Faster FHEW Bootstrapping with Adaptive Key Update

Qi Zhang[1,2][0000−0003−2904−451X], Mingqiang Wang[1][0000−0001−9221−4230], and Xiaopeng Cheng[1][0009−0001−1842−8983]

[1] School of Mathematics, Shandong University, Jinan 250100, China
zhang_qi@mail.sdu.edu.cn, wangmingqiang@sdu.edu.cn*, chengxiaopeng@mail.sdu.edu.cn

[2] Zhongtai Securities Institute for Financial Studies, Shandong University, Jinan 250100, China

Abstract. Lee et al. proposed a new bootstrapping algorithm based on homomorphic automorphism, which merges the empty sets of ciphertexts by adjusting the window size. This algorithm supports arbitrary secret key distributions with no additional runtime costs while using small evaluation keys. However, our implementation reveals that once the window size exceeds a certain threshold, the time required for bootstrapping remains relatively constant. This observation prompts the question of how to further reduce the running time. To address this challenge, we introduce a new trick called Adaptive Key Update (AKU). With AKU and automorphism techniques, we propose a new bootstrapping algorithm for Gaussian secret keys that requires only n external products and no switching for blind rotation. Building on this, we employ window size optimization and key switching techniques to further improve the algorithm. The improved algorithm provides a useful trade-off between key storage and computational efficiency, depending on the choice of the window size. Compared to the current fastest FHEW bootstrapping method for Gaussian secret keys (the LLWW+ method proposed by Li et al.), our AKU-based algorithm reduces the number of key switching by 76% and decreases the running time of bootstrapping by 20.7%. At this time, the practical runtime for bootstrapping is approximately equal to that of performing only n external products.

Keywords: Automorphism · Key Switching · Blind Rotation · Bootstrapping · Fully Homomorphic Encryption (FHE).

1 Introduction

Homomorphic encryption (HE) allows computations to be performed on encrypted data without revealing the underlying plaintext, ensuring data confidentiality throughout the process. This unique capability makes HE an attractive tool for a wide range of privacy-preserving applications, allowing data to be processed without ever exposing sensitive information. Most HE schemes rely on the

* Corresponding author.

difficulty of computational problems such as Learning With Errors (LWE) [27] or Ring Learning With Errors (RLWE) [24], where ciphertexts contain small noise to guarantee security. However, as computations on ciphertexts accumulate, the noise also increases, which can increase the probability of decryption failure, and limit the number of operations that can be performed on the encrypted data. Since Craig Gentry's pioneering introduction of the first fully homomorphic encryption (FHE) scheme [17], HE has seen significant advancements. These advancements include the development of several novel schemes that significantly improve Gentry's original construction [8, 15, 11, 14, 12]. Additionally, HE has found practical applications in various domains, such as private information retrieval [4], private set intersection [10], privacy-preserving genome-wide association studies [5], and logistic regression learning [19, 31]. A major community initiative has also been established to standardize homomorphic encryption [2].

Most existing FHE constructions adhere to the paradigm initially proposed by Gentry [17], which consists of two main components: 1) the construction of a somewhat homomorphic encryption (SHE) scheme, typically supporting only a limited number of homomorphic operations due to the accumulation of noise during computation; and 2) the design of a bootstrapping algorithm that homomorphically evaluates the decryption function, thereby refreshing the ciphertext and mitigating the noise buildup. Early research efforts [24, 9, 28, 29, 7, 8, 15, 18] primarily focused on developing SHE schemes capable of evaluating their own (augmented) decryption circuits. These schemes were later transformed into FHE schemes through Gentry's bootstrapping theorem [17], which remains foundational in the field. However, since bootstrapping is a critical and computationally intensive process in nearly all FHE constructions, it has become a significant bottleneck in achieving practical performance for real-world applications. As a result, the development of efficient bootstrapping techniques has become one of the central challenges in FHE research, with ongoing efforts to optimize this procedure being essential for enhancing the overall efficiency and scalability of FHE systems.

The FHEW FHE scheme [14, 3] and its TFHE variant [13] are among the most well-known methods for performing bit-level homomorphic computations on encrypted data. These schemes enable efficient processing of encrypted binary data and have been the foundation for subsequent innovations in homomorphic encryption. Three classic approaches have been developed for FHEW-like bootstrapping: 1) AP bootstrapping method: Originally proposed by Alperin-Sheriff and Peikert [3], this method was efficiently instantiated in the ring setting by the FHEW cryptosystem [14]. It is particularly effective for LWE secret keys that follow Gaussian or uniform distributions, but it requires relatively large evaluation keys. 2) GINX bootstrapping method: Introduced by Gama et al. [16], the GINX method has been further optimized for ternary secret keys, reducing the computational overhead by approximately half. We refer to this optimized version as GINX* [6, 20]. The GINX method, especially its TFHE variant, excels in scenarios with binary LWE secret keys. 3) LMKC+ bootstrapping method: Proposed by Lee et al. [22], LMKC+ method leverages the homomorphic au-

tomorphism technique and introduced a window size optimization strategy to effectively merge ciphertexts. This method supports arbitrary secret key distributions without additional runtime costs, while using small evaluation keys.

Recently, Wang et al. [30] and Li et al. [23] proposed the WWLL+ and LLWW+ methods, respectively, both of which build upon the LMKC+ method [22]. The WWLL+ method further reduces the number of homomorphic automorphisms by employing a naive sparse approach. While this approach is effective in minimizing the number of homomorphic automorphisms, it can lead to a rapid increase in noise, potentially degrading the performance of homomorphic computations. On the other hand, the LLWW+ method achieves similar reductions in the number of homomorphic automorphisms by merging symmetric sets, and mitigating some of the noise issues associated with the naive sparse approach. However, one notable limitation of LLWW+ is that the size of the blind rotation keys is approximately twice that of the original LMKC+ method.

Efficiency aside, the use of secret keys with large entries remains interesting for both theoretical and practical reasons. Theoretically, lattice cryptography supports Gaussian-distributed secret keys with a standard deviation of $O(\sqrt{n})$, where n is the dimension of the secret vector and serves as a security parameter [24, 27]. Research [25] shows that LWE with binary secrets can be as hard as standard LWE with uniform or Gaussian secrets, but at the cost of increasing the secret dimension by a factor of $O(\log q)$ and the error magnitude by a factor of $O(\sqrt{n})$. For practical reasons, such as limiting error growth during homomorphic computation and efficiently implementing GINX bootstrapping, these theoretical results supporting binary secrets are often overlooked. Instead, practical parameters are set based on the best-known attacks to balance security and efficiency. For Gaussian secret keys, almost all bootstrapping algorithms can achieve the same level of security with a smaller value of n. In this paper, we adopt a similar approach when comparing our work to previous schemes.

Although both the WWLL+ and LLWW+ methods reduce the number of homomorphic automorphisms compared to the LMKC+ method, it is observed that, when the window size is small, the number of homomorphic automorphisms in both the LMKC+ and LLWW+ methods decreases significantly as the window size increases. However, once the window size exceeds a certain threshold (approximately 10), the number of homomorphic automorphisms required for bootstrapping stabilizes and shows little change. This observation raises an important question: Is it possible to achieve bootstrapping for keys with a Gaussian distribution that provides faster performance and lower noise both theoretically and practically, or even matches the theoretical complexity of binary GINX?

1.1 Our Results

We have answered the above questions in the affirmative, and have the following three main contributions to the algorithmic-level and implementation-level of bootstrapping schemes.

- **Algorithmic-Level Contributions**

We introduce a new trick, called Adaptive Key Update (AKU), to reduce the number of key switching during blind rotation. With AKU and automorphism techniques, we design a new bootstrapping procedure that supports the use of arbitrary secret key distributions without any performance penalty (similar to AP/LMKC+ bootstrapping) while requiring only n external products and no switching for blind rotation. One consideration is that the key size can be relatively large.
We combine AKU, automorphism, window size and key switching techniques to propose an improved bootstrapping algorithm. This algorithm provides a useful trade-off between the storage requirements for evaluation keys and computational efficiency based on the choice of the window size. By significantly reducing the number of key switching during blind rotation, our method also reduces running time compared to LLWW+ method. This method can be naturally extended to bootstrapping algorithms based on NTRU.

- **Implementation-Level Verification**
We verify our theoretical results of the proposed method through experiments based on the OpenFHE open-source homomorphic encryption library [1]. Concretely, in the case of a gate bootstrapping operation at a 128-bit security level, our scheme outperforms the current fastest FHEW bootstrapping algorithm [23] by reducing the number of key-switching operations by 76% (from 301 to 77). This results in a 20.7% reduction in gate bootstrapping time. At this point, the practical runtime for bootstrapping is approximately equal to that of performing only n external products(See Sect. 5 for details.) Since the key size increases by a factor of 2, this trade-off makes the scheme more suitable for scenarios where there is ample key storage capacity.

1.2 Techniques

The main operation in FHEW bootstrapping is the evaluation of a so-called "blind rotation". This operation takes some polynomial f_0 as an input and "rotates" it by some value encrypted within a given LWE ciphertext ($\mathbf{a} = (a_0, ..., a_{n-1}), b$) under secret key $\mathbf{s} = (s_0, ..., s_{n-1})$ (See Sect. 2 for more details). Starting with the ciphertext RLWE(f_0) of a polynomial f_0 under $\mathbf{z}(X)$, previous blind rotation algorithms work as follows: at step i, given a ciphertext RLWE(f_{i-1}) of a polynomial $f_{i-1}(X) = f_0 \cdot X^{\sum_{j \le i-1} a_j s_j}$, homomorphically compute RLWE(f_i) of an updated polynomial $f_i = f_{i-1} \cdot X^{a_i \cdot s_i} = f_0 \cdot X^{\sum_{j \le i} a_j s_j}$, using a publicly known constant a_i part of the input LWE ciphertext and an encryption $E(s_i)$ of a secret key coordinate s_i. After repeating this step n times, we obtain the encryption of RLWE($f_0 \cdot X^{\langle \mathbf{a}, \mathbf{s} \rangle}$), which is a negacyclic rotation of f_0 by $\langle \mathbf{a}, \mathbf{s} \rangle$ positions. There are three classic blind rotation algorithms, and their main ideas are as follows:

- AP works by including encryptions $E(2^j \cdot s_i)$ for all $j \in [0, \log q - 1]$ in the evaluation keys and then using c_j as a selector to pick one of them, where c_j ensures that $a_i = \sum_j c_j 2^j$. This allows using arbitrary keys s_i with no impact

on the running time, but also requires large evaluation keys due to the need to store multiple encryptions $E(2^j \cdot s_i)$ for every secret key element s_i.

- GINX works by assuming $s_i \in \{0, 1\}$ is a single bit, and using $E(s_i)$ as a selector between the original ciphertext RLWE(f_{i-1}) and a modified version RLWE$(f_{i-1} \cdot X^{a_i})$ through a homomorphic "MUX" gate. This approach requires only a single encryption $E(s_i)$ for each key element, but it is inherently suited only for binary secrets. While larger secrets can be handled using various methods, these solutions inevitably incur additional costs, either in terms of key size or computational overhead.

- LMKC+ performs blind rotation by homomorphic automorphisms in R_Q. Given RLWE$(f_{i-1}(X))$, it first applies a homomorphic autormorphsim $\psi_{1/a_i}(\cdot)$ where $\psi_a(h) := h(X^a)$, to obtain the ciphertext of $f_{i-1}(X^{a_i^{-1}})$. Next, it homomorphically multiply the ciphertext by X^{s_i} to get the cipertext of $f_{i-1}(X^{a_i^{-1}}) \cdot X^{s_i}$. Finally, it again applies the homomorphic automorphism $\psi_{a_i}(\cdot)$ to obtain the ciperhtext RLWE$(f_{i-1}(X) \cdot X^{a_i s_i})$.

In this paper, We first propose Adaptive Key Update(AKU) to optimize the blind rotation algorithm by automorphism technique. For homomorphic automorphism, consider a ciphertext $c(X) \in R_Q$ that encrypts X^{s_i} under the secret key $\boldsymbol{z}(X)$, we can easily obtain a new ciphertext $c(X^{a_i})$ that encrypts $X^{a_i s_i}$ by applying the automorphism $X \to X^{a_i}$ to $c(X)$ if $a_i \in \mathbb{Z}_{2N}^*$. The challenge, however, is that $c(X^{a_i})$ is encrypted under a different secret key $\boldsymbol{z}(X^{a_i})$, not the original secret key $\boldsymbol{z}(X)$. To allow further homomorphic computations, methods like LMKC+ and LLWW+ require key switching to convert $c(X^{a_i})$ back to the ciphertext under $\boldsymbol{z}(X)$ that encrypts $X^{a_i s_i}$. We observe that the time required for automorphism is almost negligible compared to the time required for key switching. Abstractly, AKU involves adaptively "rotating" the secret key in RLWE ciphertexts during blind rotation, with the number of bits per rotation determined by a specific value in the given LWE ciphertext. The purpose is to perform no key switching after applying an automorphism, or perform one key switching after several automorphisms, so as to reduce the total time required for blind rotation and improve the efficiency of bootstrapping.

Recall that the goal of blind rotation is to homomorphically compute $f_0(X) \cdot X^{\langle \mathbf{a}, \mathbf{s} \rangle} = f_0(X) \cdot X^{\sum_{i=0}^{n-1} a_i s_i}$. Suppose each $a_i \in \mathbb{Z}_{2N}^*$ and $a_i' \cdot a_i \equiv 1 \mod 2N$. Starting with the ciphertext RLWE$(f_0(X^{a_0'}))$ of a polynomial $f_0(X^{a_0'})$ under $\mathbf{z}(X^{a_0'})$, our blind rotation algorithm works as follows: at step i, given a ciphertext RLWE(f_{i-1}) of a polynomial $f_{i-1} = f_0(X^{a_i'}) \cdot X^{a_i' \sum_{j=0}^{j=i-2} a_j s_j}$ under secret key $\mathbf{z}(X^{a_i'})$, homomorphically compute RLWE(f_i) of an updated polynomial $f_i = f_{i-1} \cdot X^{a_i' \cdot a_{i-1} s_{i-1}}$. Then we apply the automorphism $\psi_{a_i a_{i+1}'}(\cdot)$ to get an encryption of $f_0(X^{a_{i+1}'}) \cdot X^{a_{i+1}' \sum_{j=0}^{j=i-1} a_j s_j}$ under secret key $\mathbf{z}(X^{a_{i+1}'})$. Repeating the above process and completing the loop with respect to i, we can get $f_0(X^{a_n'}) \cdot X^{a_n' \sum_{j=0}^{j=n-1} a_j s_j}$ under secret key $\mathbf{z}(X^{a_n'})$, where $a_n' = 1$. We obtain the fastest bootstrapping theoretically and practically, which has the same theoretical complexity as binary GINX. It only takes n external products and no key switching! Two considerations are that the key size can be relatively

large, and naive sparse approach may lead to a notable increase in noise, similar to what has been observed in WWLL+ method. Building on this, we employ window size optimization and key switching techniques to further improve the algorithm, which performs a key switching only when the secret key of RLWE ciphertexts rotates at least w bits during blind rotation. Finally, the algorithm provides a useful trade-off between storage requirements for evaluation keys and computational complexity, depending on the choice of the window size.

2 Preliminaries

Let N be a power of two. We denote the $2N$-th cyclotomic ring by $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$ and its quotient ring by $\mathcal{R}_Q := \mathcal{R}/Q\mathcal{R}$. We use regular letters to represent (modular) integers, such as $a \in \mathbb{Z}_q$, while bold letters represent polynomials $\mathbf{a} \in \mathcal{R}$ or vectors $\mathbf{a} \in \mathbb{Z}^n$. For two vectors $\mathbf{a}$ and $\mathbf{b}$, we denote their inner product by $\langle \mathbf{a}, \mathbf{b} \rangle$. We write the floor, ceiling and round functions as $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$ and $\lfloor \cdot \rceil$, respectively. For $q \in \mathbb{Z}$ and $q > 1$, we identify the ring $\mathbb{Z}_q$ with the representative interval $(-q/2, q/2]$. For $x \in \mathbb{Z}$, we denote the centered remainder of x modulo q by $[x]_q \in \mathbb{Z}_q$. These notations are extended to elements of $\mathcal{R}$ by applying them coefficient-wise. For $\mathbf{a} = a_0 + a_1 X + ... + a_{N-1} X^{N-1} \in \mathcal{R}$, we denote the ℓ_∞ norm of $\mathbf{a}$ as $\|\mathbf{a}\|_\infty = \mathsf{max}_{0 \le i \le N-1}\{|a_i|\}$. We use $a \leftarrow S$ to denote uniform sampling from the set S. Sampling according to a distribution χ is denoted by $a \leftarrow \chi$. We denote χ_{err} as a subgaussian distribution with parameter σ.

2.1 Basic Lattice-Based Encryption

For positive integers q and n, the basic LWE encryption of $m \in \mathbb{Z}$ under the secret key $\mathbf{s} \leftarrow \chi_{\mathsf{key}}$ is defined as

$$\mathrm{LWE}_{q,\mathbf{s}}(m) = (\mathbf{a}, b) = (\mathbf{a}, -\langle \mathbf{a}, \mathbf{s} \rangle + m + e) \in \mathbb{Z}_q^{n+1},$$

where $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ and the error term $e \leftarrow \chi_{\mathsf{err}}$. In some cases, we may omit the subscripts q and $\mathbf{s}$ when they are understood from the context.

For an integer Q and a power of two N, the basic RLWE encryption of $\mathbf{m} \in \mathcal{R}$ under the secret key $\mathbf{z} \leftarrow \chi_{\mathsf{key}}$ is defined as

$$\mathrm{RLWE}_{Q,\mathbf{z}}(\mathbf{m}) := (\mathbf{a}, -\mathbf{a} \cdot \mathbf{z} + \mathbf{e} + \mathbf{m}) \in \mathcal{R}_Q^2,$$

where $\mathbf{a} \leftarrow \mathcal{R}_Q$, and for each coefficient e_i of $\mathbf{e}$ is sampled from χ_{err}, with $i \in [0, N-1]$.

As in the case of LWE, we will often omit the subscripts Q and $\mathbf{z}$ when they are clear from the context. Gadget decomposition plays a crucial role in maintaining error control within Fully Homomorphic Encryption (FHE) schemes. There are two distinct types of gadget decomposition, which differ in the choice of the gadget vectors.

Canonical Gadget Decomposition: The gadget vector of the canonical gadget decomposition is consisted with the power of B, where $\mathbf{g} = (1, B, B^2, ..., B^{d-1})$. Here, $d = \lceil \log_B Q \rceil$ is referred to as the gadget length, and B is the gadget base. Using this gadget vector, any ring polynomial $\mathbf{t}$ can be decomposed into a sequence of polynomials $(\mathbf{t}_0, ..., \mathbf{t}_{d-1})$ such that the absolute values of the coefficients of each polynomial $\mathbf{t_i}$ are bounded by $B/2$. This decomposition satisfies the equation $\sum_i g_i \mathbf{t_i} = \mathbf{t}$.

Approximate Gadget Decomposition: Approximate gadget decomposition generalizes the canonical gadget decomposition. When $B^d < Q$, the decomposition of a ring element is no longer exact. In this case, the decomposition error is defined as $\varepsilon_{\text{gadget}}(\mathbf{t}) = \sum_i g_i \mathbf{t_i} - \mathbf{t}$, with ϵ denoting its infinite norm, i.e., $\epsilon = \| \sum_i g_i \mathbf{t_i} - \mathbf{t} \|_\infty$. In the approximate decomposition, the gadget vector is typically chosen as $\mathbf{g} = (\lceil Q/B^d \rceil, \lceil Q/B^d \rceil B, ..., \lceil Q/B^d \rceil B^{d-1})$. Each ring polynomial $\mathbf{t}$ is then decomposed into a set of polynomials $(\mathbf{t}_0, ..., \mathbf{t}_{d-1})$ where the absolute values of the coefficients of each $\mathbf{t}_i$ are still bounded by $B/2$, and the decomposition error ϵ satisfies $\epsilon \leq \frac{1}{2} \lceil Q/B^d \rceil$.

We adapt the definitions of RLWE$'$ and RGSW from [26]. Given a gadget vector $\mathbf{g}$, we define $\text{RLWE}'_{\mathbf{z}}(\mathbf{m})$ and $\text{RGSW}_{\mathbf{z}}(\mathbf{m})$ as follows

$$\text{RLWE}'_{\mathbf{z}}(\mathbf{m}) := \big(\text{RLWE}_{\mathbf{z}}(g_0 \cdot \mathbf{m}), \text{RLWE}_{\mathbf{z}}(g_1 \cdot \mathbf{m}), \cdots, \text{RLWE}_{\mathbf{z}}(g_{d_g-1} \cdot \mathbf{m})\big) \in \mathcal{R}_Q^{2d},$$
$$\text{RGSW}_{\mathbf{z}}(\mathbf{m}) := \big(\text{RLWE}'_{\mathbf{z}}(\mathbf{z} \cdot \mathbf{m}), \text{RLWE}'_{\mathbf{z}}(\mathbf{m})\big) \in \mathcal{R}_Q^{2 \times 2d}.$$

Gadget Product: The scalar multiplication between an element in $\mathcal{R}_Q$ and RLWE$'$ ciphertext

$$\odot : \mathcal{R}_Q \times \text{RLWE}' \to \text{RLWE}$$

is defined as

$$\begin{aligned} t \odot \text{RLWE}'_{\mathbf{z}}(\mathbf{m}) &= \langle (\mathbf{t}_0, \cdots, \mathbf{t}_{d_g-1}), \big(\text{RLWE}_{\mathbf{z}}(g_0 \cdot \mathbf{m}), \cdots, \text{RLWE}_{\mathbf{z}}(g_{d_g-1} \cdot \mathbf{m})\big) \rangle \\ &= \sum_{i=0}^{d_g-1} \mathbf{t}_i \cdot \text{RLWE}_{\mathbf{z}}(g_i \cdot \mathbf{m}) \\ &= \text{RLWE}_{\mathbf{z}}(\mathbf{t} \cdot \mathbf{m}) \in \mathcal{R}_Q^2. \end{aligned}$$

Let $\mathbf{e}_i$ be the error in $\text{RLWE}_{\mathbf{z}}(g_i \cdot \mathbf{m})$. Then, the error after multiplication is given by $\sum_{i=0}^{d_g-1} \mathbf{t}_i \cdot \mathbf{e}_i + \epsilon_{\mathsf{gadget}}(t)$, where the total error remains small if both $\mathbf{t}_i$ and $\mathbf{e}_i$ are small.

Lemma 1 (Kim et al. [21]). *Let B and d 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*

$$\sigma^2_{\odot,input} \leq dN \frac{B^2}{12} \sigma^2_{input} + \frac{N}{3} \|\mathbf{m}\|_2^2 \epsilon^2,$$

where σ^2_{input} is the error variance of the input RLWE$'$ *ciphertext.*

Lemma 1 is derived from [21] proposition 1 using the fact that $\epsilon \leq \frac{1}{2}\lceil \frac{q}{B^\ell} \rceil$. When applying canonical gadget decomposition, the error term $\frac{N}{3}\|\mathbf{m}\|_2^2\epsilon^2$ is absent.

External Product: The multiplication between RLWE and RGSW ciphertexts

$$\circledast : \text{RLWE} \times \text{RGSW} \to \text{RLWE}$$

is defined as

$$\begin{aligned}\text{RLWE}_{\mathbf{z}}(\mathbf{m}_1) \circledast \text{RGSW}_{\mathbf{z}}(\mathbf{m}_2) &= (\mathbf{a}, \mathbf{b}) \circledast \left(\text{RLWE}'_z(\mathbf{z} \cdot \mathbf{m}_2), \text{RLWE}'_z(\mathbf{m}_2)\right) \\ &= \mathbf{a} \odot \text{RLWE}'_{\mathbf{z}}(\mathbf{z} \cdot \mathbf{m}_2) + \mathbf{b} \odot \text{RLWE}'_{\mathbf{z}}(\mathbf{m}_2) \\ &= \text{RLWE}_{\mathbf{z}}(\mathbf{m}_1 \cdot \mathbf{m}_2 + \mathbf{e}_1 \cdot \mathbf{m}_2) \in \mathcal{R}_Q^2.\end{aligned}$$

This result represents an encryption of the product $\mathbf{m}_1 \cdot \mathbf{m}_2$ with an additional error term $\mathbf{e}_1 \cdot \mathbf{m}_2$ under the RLWE scheme. To ensure that $\text{RLWE}_{\mathbf{z}}(\mathbf{m}_1) \circledast \text{RGSW}_{\mathbf{z}}(\mathbf{m}_2) \approx \text{RLWE}_{\mathbf{z}}(\mathbf{m}_1 \cdot \mathbf{m}_2)$, it is crucial to keep the error term $\mathbf{e}_1 \cdot \mathbf{m}_2$ small. This can be achieved by using monomials $\mathbf{m}_2 = \pm X^v$ as messages.

Key Switching in RLWE: Given an key switching key $\mathsf{SWK} = \text{RLWE}'_{\mathbf{z_2}}(\mathbf{z_1})$, then the key switching algorithm $\mathsf{KS}_{\mathbf{z_1} \to \mathbf{z_2}} : \text{RLWE}_{\mathbf{z_1}}(\mathbf{m}) \to \text{RLWE}_{\mathbf{z_2}}(\mathbf{m})$ is defined by

$$\begin{aligned}\mathsf{KS}_{\mathbf{z_1} \to \mathbf{z_2}}, (\text{RLWE}_{\mathbf{z_1}}(\mathbf{m}), \mathsf{SWK}) &= \mathbf{a} \odot \text{RLWE}'_{\mathbf{z_2}}(\mathbf{z_1}) + (0, \mathbf{b}) \\ &= \text{RLWE}_{\mathbf{z_2}}(\mathbf{a} \cdot \mathbf{z_1} + \mathbf{b}) \\ &= \text{RLWE}_{\mathbf{z_2}}(\mathbf{m}),\end{aligned}$$

where $(\mathbf{a}, \mathbf{b}) = \text{RLWE}_{\mathbf{z_1}}(\mathbf{m})$.

Homomorphic Automorphism: For $t \in \mathbb{Z}_{2N}^*$, an automorphism ψ_t: $\mathcal{R} \to \mathcal{R}$ is defined as $\mathbf{a}(X) \to \mathbf{a}(X^t)$, where $\mathcal{R}$ is the ring of polynomials. The associated automorphism key is given by $\mathsf{ATK}_t = \text{RLWE}'_{\mathbf{z}}(\mathbf{z}(X^t))$, the homomorphic automorphism $\mathsf{HomAuto}_t : \text{RLWE}_{\mathbf{z}}(\mathbf{m}) \to \text{RLWE}_{\mathbf{z}}(\mathbf{m}(X^t))$ is defined by

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

where $(\mathbf{a}, \mathbf{b}) = \text{RLWE}_{\mathbf{z}}(\mathbf{m})$.

2.2 FHEW-Like Bootstrapping

We briefly explain FHEW-like bootstrapping for NAND gates [14, 26]. FHEW like NAND gate bootstrapping starts with two $\text{LWE}_{q,\mathbf{s}}$ ciphertexts with a small modulus q and adds them together. After performing blind rotation and extraction procedures, we obtain an LWE encryption of the NAND gate operation on the plaintexts, now with a higher ciphertext modulus Q. As described in [26], the core bootstrapping procedure based on an accumulator is first followed by modulus switching, which changes the parameters from (Q, N) to (Q_{ks}, N). Then,

key switching is performed, changing the parameters from (Q_{ks}, N) to (Q_{ks}, n). Finally, another modulus switching step reduces the parameters from (Q_{ks}, n) to (q, n). The full bootstrapping procedure is illustrated in Fig.1. We focus on the blind rotation part and refer to [26] for further details on the other steps of the FHEW-like bootstrapping process.

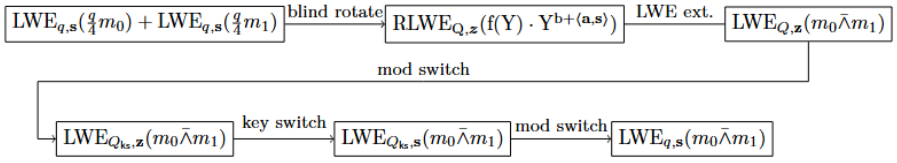

Fig. 1. NAND gate bootstrapping procedure of FHEW scheme

The core of bootstrapping algorithm can be divided into two steps:

Blind Rotation. For $q|2N$, let $Y = X^{\frac{2N}{q}}$. Blind rotation is an algorithm which takes as input a ring element $f(Y) \in \mathcal{R}_Q$, an $\mathrm{LWE}_{q,\mathbf{s}}$ ciphertext $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, and blind rotation keys $\mathsf{BRK}_{\mathbf{z},\mathbf{s}}$ corresponding to secrets $\mathbf{z}$ and $\mathbf{s}$ and outputs an RLWE ciphertext:

$$\mathrm{RLWE}_{\mathbf{z}}\left(f(Y) \cdot Y^{b+\langle \mathbf{a},\mathbf{s}\rangle}\right) \in \mathcal{R}_Q^2.$$

Three different blind rotation algorithms are introduced in [13, 14, 22]. These methods are referred to as "AP blind rotation", "GINX blind rotation", and "LMKC+ blind rotation", respectively. All of these algorithms make use of the properties of RGSW ciphertexts as described earlier.

- AP Blind Rotation. In AP blind rotation [14, 3], the blind rotation keys are generated for each element $s_i \in \mathbb{Z}_q$ of the secret $\mathbf{s}$ as follows:

$$\mathsf{BSK} = \{\mathsf{BSK}_{i,j,v} = \mathrm{RGSW}_{\mathbf{z}}(Y^{vB_r^j s_i})\}_{i,j,v}$$

for $i \in [0, n-1]$, $j \in [0, \log_{B_r}(q) - 1]$, and $v \in \mathbb{Z}_{B_r}$. In the algorithm, the accumulator acc is initialized to the trivial encryption

$$\mathsf{acc} = \mathrm{RLWE}_{Q,\mathbf{z}}(f(Y) \cdot Y^b) = (0, f(Y) \cdot Y^b).$$

Next, for each $i \in [0, n-1]$, the value a_i is decomposed in base B_r as $a_i = \sum_{j=0}^{\log_{B_r}(q)-1} a_{i,j} B_r^j$. The accumulator acc is then updated sequentially for each $a_{i,j}$ as

$$\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathrm{RGSW}_{\mathbf{z}}(Y^{a_{i,j} B_r^j s_i}).$$

The full procedure of AP blind rotation is described in Algorithm 1.

AP blind rotation supports all types of secret key distributions and offers a useful trade-off between the storage requirements for evaluation keys and computational complexity, depending on the choice of the base $B_r \geq 2$. A larger

Algorithm 1 Blind Rotation: AP [14, 3]

Input: $(\mathbf{a}, b)$, acc, $\{\mathsf{BSK}_{i,j,v}\}_{i\in[0,n-1],j\in[0,\log_{B_r}(q)-1],v\in\mathbb{Z}_{B_r}}$
Output: $\mathbf{c}$
1: **for** $(i = 0; i < n; i = i + 1)$ **do**
2: **for** $(j = 0; j < \log_{B_r}(q); j = j + 1)$ **do**
3: $a_{i,j} = \lfloor a_i / B_r^j \rfloor \pmod{B_r}$
4: $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{BKS}_{i,j,a_{i,j}}$
5: **return** $\mathbf{c} = \mathsf{acc}$

B_r allows for faster computations but increases the storage requirements for the evaluation keys. Conversely, a smaller B_r reduces the storage overhead but leads to longer computational times.

- GINX Blind Rotation. GINX blind rotation [13, 16] is more efficient than AP when the secret key $\mathbf{s}$ is binary or ternary. However, its performance degrades when using larger secret keys [26]. In the general case, each secret key element $s_i \in \mathbb{Z}_q,\ i \in [0, N-1]$, is expressed as a subset-sum $s_i = \sum_{j=0}^{|U|-1} u_j s_{i,j}$, where $s_{i,j} \in \{0, 1\}$ and $U \subset \mathbb{Z}_q$ is an appropriately chosen subset of $\mathbb{Z}_q$. To express arbitrary elements of $\mathbb{Z}_q$, one can use $U = \{1, 2, 4, \ldots, 2^{k-1}\}$. For binary and ternary secrets, $U = \{1\}$ and $U = \{1, -1\}$ can be used, respectively [26]. For any fixed set U, the blind rotation keys are generated as

$$\mathsf{BSK} = \{\mathsf{BSK}_{i,j} = \mathrm{RGSW}_{\mathbf{z}}(s_{i,j})\}_{i,j},$$

where $i \in [0, n-1]$ and $j \in [0, |U|-1]$. In the algorithm, the accumulator acc is initiated as

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}}(f(Y) \cdot Y^b) = (0, f(Y) \cdot Y^b),$$

and is updated as

$$\mathsf{acc} \leftarrow \mathsf{acc} + (Y^{a_i u_j} - 1) \cdot (\mathsf{acc} \circledast \mathrm{RGSW}_{\mathbf{z}}(s_{i,j})).$$

Algorithm 2 Blind Rotation: GINX [13, 16, 26]

Input: $(\mathbf{a}, b)$, acc, $\{\mathsf{BSK}_{i,j}\}_{i\in[0,n-1],j\in[0,|U|-1]}$
Output: $\mathbf{c}$
1: **for** $(i = 0; i < n; i = i + 1)$ **do**
2: **for** $(j = 0; j < |U|; j = j + 1)$ **do**
3: $\mathsf{acc} \leftarrow \mathsf{acc} + (Y^{a_i u_j} - 1) \cdot (\mathsf{acc} \circledast \mathsf{BSK}_{i,j})$.
4: **return** $\mathbf{c} = \mathsf{acc}$

If $s_{i,j} = 0$, the second addendum is ignored as it encrypts zero, and the value in the accumulator acc remains unchanged. If $s_{i,j} = 1$, then $\mathsf{acc} \circledast \mathrm{RGSW}_{\mathbf{z}}(1)$ is equal to acc, and the accumulator acc is updated to $Y^{a_i u_j} \cdot \mathsf{acc}$. Repeating this procedure for all $j \in [0, |U|-1]$ results in the final value $Y^{a_i s_i} \cdot \mathsf{acc}$. The full procedure for GINX blind rotation is described in Algorithm 2.

- LMKC+ Automorphism-Based Blind Rotation. Under the observation that $\mathbb{Z}_{2N}^* \cong \mathbb{Z}_{N/2} \otimes \mathbb{Z}_2$, if $a_i \in \mathbb{Z}_{2N}^*$ for all $i \in [0, n-1]$, then each a_i can be expressed in terms of the generators $\{g, -1\}$. In the case of modulus $2N$, consider the decomposition of the sum $\sum_i a_i s_i$ as follows

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

where $I_\ell^+ = \{i : a_i = g^\ell\}$ and $I_\ell^- = \{i : a_i = -g^\ell\}$, for $\ell \in [0, N/2-1]$. The

Algorithm 3 Blind Rotation: LMKC+ [22] for $q = N$

Input: $(\mathbf{a}, b)$, acc, $\{\mathsf{BSK}_i\}_{i\in[0,n-1]}$, $\mathsf{BRK}_{\mathsf{sum}}$, $\{\mathsf{ATK}_{g^u}\}_{u\in[1,w]}$, ATK_{-g}
Output: $\mathbf{c}$

$\mathbf{a} \leftarrow 2\mathbf{a} + \mathbf{1} \mod 2N$
$v \leftarrow 0$
for $(\ell = \frac{N}{2} - 1; \ell > 0; \ell = \ell - 1)$ **do**
  for $j \in I_\ell^-$ **do**
    $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_j$
  $v \leftarrow v + 1$
  if $(I_{\ell-1}^- \neq \emptyset$ or $v = w$ or $\ell = 1)$ **then**
    $\mathsf{acc} = \mathsf{HomAuto}_{g^v}(\mathsf{acc}, \mathsf{ATK}_{g^v})$
    $v \leftarrow 0$
for $j \in I_0^-$ **do**
  $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_j$
$\mathsf{acc} = \mathsf{HomAuto}_{-g}(\mathsf{acc}, \mathsf{ATK}_{-g})$;
for $(\ell = \frac{N}{2} - 1; \ell > 0; \ell = \ell - 1)$ **do**
  for $j \in I_\ell^+$ **do**
    $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_j$
  $v \leftarrow v + 1$
  if $(I_{\ell-1}^+ \neq \emptyset$ or $v = w$ or $\ell = 1)$ **then**
    $\mathsf{acc} = \mathsf{HomAuto}_{g^v}(\mathsf{acc}, \mathsf{ATK}_{g^v})$
    $v \leftarrow 0$
for $j \in I_0^+$ **do**
  $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_j$
$\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BRK}_{\mathsf{sum}}$
return $\mathbf{c} = \mathsf{acc}$

blind rotation keys are generated as

$$\mathsf{BSK} = \{\mathsf{BSK}_i = \mathrm{RGSW}_{\mathbf{z}}(X^{s_i})\}_i,$$

where $i \in [0, n-1]$, and homomorphic automorphism keys are generated as $\{\mathsf{ATK}_{g^u} = \mathrm{RLWE}'_{\mathbf{z}}(\mathbf{z}(X^{g^u}))\}_{u\in[1,w]}$, $\mathsf{ATK}_{-g} = \mathrm{RLWE}'_{\mathbf{z}}(\mathbf{z}(X^{-g}))$. In the algorithm, the accumulator acc is initiated as

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}}(f(Y) \cdot Y^b) = (0, f(X^{-g}) \cdot X^{-gb}).$$

The blind rotation algorithm proceeds by first multiplying (via the external product) the accumulator by BSK_j for all $j \in I^-_{N/2-1}$, then applying $\mathsf{HomeAuto}_g$ and repeating the process for each index set I. However, after multiplying with I^-_0, the algorithm applies $\mathsf{HomAuto}_{-g}$ instead. The final results is $\mathrm{RLWE}_{\mathbf{z}}(f(Y)\cdot Y^{b+\sum_i a_i s_i})$. By utilizing $w+1$ pre-stored automorphism keys, the number of automorphisms is reduced from n to $\frac{w-1}{w}\kappa + \frac{N}{w}\ \kappa \approx N(1-e^{-n/N})$. Since the round-to-odd operation may lead to a significant increase in noise, we focus on the Algorithm 6 in LMKC+ [22], which introduces an additional key $\mathsf{BRK}_{\mathsf{sum}} = \mathrm{RGSW}_{\mathbf{z}}(X^{-\sum_i s_i})$. The full procedure for LMKC+ blind rotation is described in Algorithm 3.

Additionally, LLWW+ method further reduces the number of automorphisms by merging the symmetric sets. For a detailed description of the specific algorithm, refer to Appendix A.

Sample Extraction. The algorithm $\mathsf{SampleExtract}$ takes as input an RLWE ciphertext $\mathbf{c} = (\mathbf{a}, \mathbf{b}) \in \mathrm{RLWE}_{Q,\mathbf{z}}(\mathbf{m})$, and outputs an $\mathrm{LWE}_{Q,\mathbf{z}}(m_0)$, where m_0 denotes the constant term of $\mathbf{m}$.

3 New Blind Rotation Algorithms

3.1 Description

In this paper, we first define a set $S = \{k \cdot 2^\theta + 1 \in \mathbb{Z}_{2N} | k = 0, ..., 2N/2^\theta - 1\}$ for $\theta \geq 1$. It is known that the elements of S form a multiplicative group. Next, consider an LWE ciphertext $(\mathbf{a}, b) \in S^{n+1}$. Define the set $A = \{w_0, w_1, ..., w_L\}$, where w_i represents the distinct elements of the vector $\mathbf{a}$, ordered such that $w_0 < w_1 < ... < w_L$. We define $w_{L+1} = 1$, $I_l = \{i : a_i = w_l\}$ and $w'_l w_l \equiv 1 \mod 2N$ for $l \in [0, L]$.

The blind rotation keys are generated as

$$\mathsf{BSK} = \{\mathsf{BSK}_{i,t} = \mathrm{RGSW}_{\mathbf{z}}(X^{s_i \cdot t})\}_{i,t},$$

where $i \in [0, n-1]$, $t \in S$. The accumulator acc is initialized to the trivial ciphertext

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}(X^{w'_1})}(f(X^{w'_1}) \cdot X^{bw'_1}) = (0, f(X^{w'_1}) \cdot X^{bw'_1}).$$

By the above definition, we have the decomposition

$$\sum_i a_i s_i = w_0 \left(\sum_{j \in I_0} s_j\right) + w_1 \left(\sum_{j \in I_1} s_j\right) + ... + w_L \left(\sum_{j \in I_L} s_j\right) \mod 2N.$$

For w_0, $j \in I_0$, we apply the automorphism $X \to X^{w'_1}$ to BSK_{j,w_0}, the corresponding ciphertext is given by

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

Input: $(\mathbf{a}, b)$, acc, $\{\mathsf{BSK}_{i,t}\}_{i\in[0,n-1],t\in S}$
Output: $\mathbf{c}$
1: **for** $(l = 0; l \le L; l = l + 1)$ **do**
2: **for** $j \in I_l$ **do**
3: $\mathsf{BSK}'_j = \psi_{w'_{l+1}}(\mathsf{BSK}_{j,w_l})$
4: $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{BSK}'_j$.
5: **if** $l < L$ **then**
6: $\mathsf{acc} = \psi_{w_{l+1}w'_{l+2}}(\mathsf{acc})$
7: **return** $\mathbf{c} = \mathsf{acc}$

$$\mathsf{BSK}'_j = \mathrm{RGSW}_{\mathbf{z}(X^{w'_1})}(X^{w_0 w'_1 s_j}).$$

The accumulator acc is updated sequentially as

$$\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{BSK}'_j,$$

for all $j \in I_0$. This process results in the ciphertext

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}(X^{w'_1})}\left(f(X^{w'_1}) \cdot X^{bw'_1 + w_0 w'_1 \left(\sum_{j\in I_0} s_j\right)}\right).$$

Finally, we apply the automorphism $X \to X^{w_1 w'_2}$ to acc, yielding the updated ciphertext

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}(X^{w'_2})}\left(f(X^{w'_2}) \cdot X^{bw'_2 + w_0 w'_2 \left(\sum_{j\in I_0} s_j\right)}\right).$$

This process continues in a similar fashion, performing continuous updates on subsequent values of w_l, where $l \in [0, L]$. Throughout the process, the exponent r in the key $\mathbf{z}(X^r)$ of RLWE ciphertext adaptively updates with w'_i. A detailed description of the full procedure for blind rotation can be found in Algorithm 4.

Algorithm 4 represents the most efficient FHEW bootstrapping scheme both theoretically and practically, exhibiting the same time complexity as the binary GINX method. It requires only n external products for its execution. Two considerations to note are that the key size can become relatively large, and the naive sparse approach may lead to a significant increase in noise, as similarly observed in the WWLL+ method. To mitigate the key size expansion and prevent the noise growth induced by the naive sparse approach [30], we propose two optimized algorithms based on the specific decomposition of the sum $\sum_i a_i s_i$.

Memory Efficient Algorithm. As discussed in LMKC+, consider the decomposition of the sum $\sum_i a_i s_i$ as follows

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

Recall that the objective of blind rotation is to homomorphically compute

$$f(Y) \cdot Y^{b+\sum_i a_i s_i} = f(X^{\frac{2N}{q}}) \cdot X^{\frac{2N}{q} b + \sum_i (\frac{2N}{q} a_i + 1) s_i - \sum_i s_i}.$$

When $\frac{2N}{q} = 2^\theta,\ \theta \geq 1$, it follows that each term $\frac{2N}{q} a_i + 1$ is odd. The accumulator acc is initialized to the trivial ciphertext

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}}(f(Y^{-g}) \cdot Y^{-bg}) = (0, f(Y^{-g}) \cdot Y^{-bg}) = (0, f(X^{-2^\theta g}) \cdot X^{-2^\theta g}).$$

The automorphism key are generated as ATK_{-g}, and the key switching keys are generated as

$$\mathsf{SWK} = \{\mathsf{SWK}_v = \mathrm{RLWE}'_{\mathbf{z}}(z(X^{g^v}))\}_v,$$

where $v \in [1, 2w-1]$. The blind rotation keys are generated as follows

$$\mathsf{BRK}_{\mathsf{sum}} = \mathrm{RGSW}\mathbf{z}(X^{-\sum_i s_i}),\ \mathsf{BSK} = \{\mathsf{BSK}_{i,t} = \mathrm{RGSW}_{\mathbf{z}(X^{g^t})}(X^{s_i})\}_{i,t},$$

where $i \in [0, n-1],\ t \in [0, w-1]$.

Building on the window size optimization method, we introduce a threshold for performing key switching. Specifically, a key switching operation is triggered when the accumulator exponent r in the key $\mathbf{z}(X^r)$ reaches or exceeds a threshold w. The exponent r is adaptively updated throughout the process. The full procedure of blind rotation is described in Algorithm 5.

Computation Efficient Algorithm. By merging the symmetric sets, if $a_i \in \mathbb{Z}_{2N}^*$ for all $i \in [0, n-1]$, we have the following decomposition

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

The key switching keys are generated as

$$\mathsf{SWK} = \{\mathsf{SWK}_v = \mathrm{RLWE}'_{\mathbf{z}}(\mathbf{z}(X^{g^u}))\}_v,$$

where $v \in [1, 2w-1]$. The blind rotation keys are generated as

Algorithm 5 Memory Efficient Blind Rotation for $\frac{2N}{q} = 2^\theta, \theta \geq 1$

Input: $(\mathbf{a}, b)$, acc, $\{\mathsf{BSK}_{i,t}\}_{i\in[0,n-1],t\in[0,w-1]}$, $\{\mathsf{SWK}_v\}_{v\in[1,2w-1]}$, $\mathsf{BRK}_{\mathsf{sum}}$
Output: $\mathbf{c}$

$\mathbf{a} \leftarrow 2^\theta \mathbf{a} + 1 \mod 2N$
$v \leftarrow 0,\ r \leftarrow 0$
for $(\ell = \frac{N}{2} - 1;\ \ell > 0;\ \ell = \ell - 1)$ **do**
  for $j \in I_\ell^-$ **do**
    $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_{j,r}$
  $v \leftarrow v + 1$
  if ($v = w$ or $\ell = 1$) **then**
    $\mathsf{acc} = \psi_{g^v}(\mathsf{acc}),\ \mathsf{acc} = \mathsf{KS}_{\mathbf{z}(X^{g^{r+v}})\to\mathbf{z}(X)}(\mathsf{acc}, \mathsf{SWK}_{r+v}),\ v \leftarrow 0,\ r \leftarrow 0$
  else if $I_{\ell-1}^- \neq \emptyset$ **then**
    $\mathsf{acc} = \psi_{g^v}(\mathsf{acc}),\ r \leftarrow r + v,\ v \leftarrow 0$
    if $r \geq w$ **then**
      $\mathsf{acc} = \mathsf{KS}_{\mathbf{z}(X^{g^r})\to\mathbf{z}(X)}(\mathsf{acc}, \mathsf{SWK}_r),\ r \leftarrow 0$
for $j \in I_0^-$ **do**
  $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_{j,0}$
$\mathsf{acc} = \mathsf{HomAuto}_{-g}(\mathsf{acc}, \mathsf{ATK}_{-g})$
for $(\ell = \frac{N}{2} - 1;\ \ell > 0;\ \ell = \ell - 1)$ **do**
  for $j \in I_\ell^+$ **do**
    $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_{j,r}$
  $v \leftarrow v + 1$
  if ($v = w$ or $\ell = 1$) **then**
    $\mathsf{acc} = \psi_{g^v}(\mathsf{acc}),\ \mathsf{acc} = \mathsf{KS}_{\mathbf{z}(X^{g^{r+v}})\to\mathbf{z}(X)}(\mathsf{acc}, \mathsf{SWK}_{r+v}),\ v \leftarrow 0,\ r \leftarrow 0$
  else if $I_{\ell-1}^+ \neq \emptyset$ **then**
    $\mathsf{acc} = \psi_{g^v}(\mathsf{acc}),\ r \leftarrow r + v,\ v \leftarrow 0$
    if $r \geq w$ **then**
      $\mathsf{acc} = \mathsf{KS}_{\mathbf{z}(X^{g^r})\to\mathbf{z}(X)}(\mathsf{acc}, \mathsf{SWK}_r),\ r \leftarrow 0$
for $j \in I_0^+$ **do**
  $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_{j,0}$
$\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BRK}_{\mathsf{sum}}$
return $\mathbf{c} = \mathsf{acc}$

$$\mathsf{BRK}_{\mathsf{sum}} = \mathrm{RGSW}_{\mathbf{z}}(X^{-\sum_i s_i}),\ \mathsf{BSK}^{\pm} = \{\mathsf{BSK}_{i,t}^{\pm} = \mathrm{RGSW}_{\mathbf{z}(X^{g^t})}(X^{\pm s_i})\}_{i,t},$$

where $i \in [0, n-1],\ t \in [0, w-1]$. The accumulator acc is initialized to the trivial ciphertext

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}}(f(Y^g) \cdot Y^{bg}) = (0, f(Y^g) \cdot Y^{bg}) = (0, f(X^{2^\theta g}) \cdot X^{2^\theta g}).$$

As with Memory Efficient Algorithm, we introduce the same threshold for performing key switching. The full procedure of blind rotation is described in Algorithm 6.

Algorithm 6 Computation Efficient Blind Rotation for $\frac{2N}{q} = 2^\theta, \theta \geq 1$

Input: $(\mathbf{a}, b)$, acc, $\{\mathsf{BSK}^{\pm}_{i,t}\}_{i\in[0,n-1],t\in[0,w-1]}$, $\mathsf{BRK}_{\mathsf{sum}}$, $\{\mathsf{SWK}_v\}_{v\in[1,2w-1]}$
Output: $\mathbf{c}$

$\mathbf{a} \leftarrow 2^\theta \mathbf{a} + 1 \mod 2N$
$v \leftarrow 0,\ r \leftarrow 0$
for $(\ell = \frac{N}{2} - 1;\ \ell > 0;\ \ell = \ell - 1)$ **do**
  for $j \in I_\ell^+$ or $k \in I_\ell^-$ **do**
    if $j \in I_\ell^+$ **then**
      $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}^+_{j,r}$
    if $k \in I_\ell^-$ **then**
      $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}^-_{k,r}$
  $v \leftarrow v + 1$
  if $v = w$ or $\ell = 1$ **then**
    $\mathsf{acc} = \psi_{g^v}(\mathsf{acc}),\ \mathsf{acc} = \mathsf{KS}_{\mathbf{z}(X^{g^{r+v}})\to\mathbf{z}(X)}(\mathsf{acc}, \mathsf{SWK}_{r+v}),\ v \leftarrow 0,\ r \leftarrow 0$
  else if $I^+_{\ell-1} \neq \emptyset$ or $I^-_{\ell-1} \neq \emptyset$ **then**
    $\mathsf{acc} = \psi_{g^v}(\mathsf{acc}),\ r \leftarrow r + v,\ v \leftarrow 0$
    if $r \geq w$ **then**
      $\mathsf{acc} = \mathsf{KS}_{\mathbf{z}(X^{g^r})\to\mathbf{z}(X)}(\mathsf{acc}, \mathsf{SWK}_r),\ r \leftarrow 0$
for $j \in I_0^+$ or $k \in I_0^-$ **do**
  if $j \in I_0^+$ **then**
    $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}^+_{j,0}$
  if $k \in I_0^-$ **then**
    $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}^-_{k,0}$
$\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BRK}_{\mathsf{sum}}$
return $\mathbf{c} = \mathsf{acc}$

3.2 Correctness

In this subsection we prove that Algorithm 4, Algorithm 5 and Algorithm 6 are correct blind rotation schemes.

Theorem 1. *Let* $S = \{k \cdot 2^\theta + 1 \in \mathbb{Z}_{2N} | k = 0, ..., 2N/2^\theta - 1\}$, *where* $\theta \geq 1$. *For the procedure described in Algorithm 4, given an* LWE *ciphertext* $(\mathbf{a}, b) \in S^{n+1}$, *an initial accumulator* $\mathsf{acc} = (0, f(X^{w'_1}) \cdot X^{bw'_1})$, *and the blind rotation keys* $\mathsf{BSK} = \{\mathrm{RGSW}_{\mathbf{z}}(X^{s_i \cdot t})\}_{i\in[0,n-1],t\in S}$, *the resulting ciphertext* $\mathbf{c}$ *will belong to the set* $\mathrm{RLWE}_{\mathbf{z}}\left(f(X) \cdot X^{b+\langle \mathbf{a}, \mathbf{s}\rangle}\right)$.

A proof of this theorem will be given in Appendix B.

Theorem 2. *Given Algorithm 5, on input a ciphertext* $(\mathbf{a}, b) \in \mathrm{LWE}_{q,\mathbf{s}}(m)$, *an accumulator* $\mathsf{acc} = (0, f(Y^{-g}) \cdot Y^{-bg})$, *keys* $\mathsf{SWK} = \{\mathrm{RLWE}'_{\mathbf{z}}(\mathbf{z}(X^{g^v}))\}_{v\in[1,2w-1]}$, $\mathsf{BSK} = \{\mathrm{RGSW}_{\mathbf{z}(X^{g^t})}(X^{s_i})\}_{i\in[0,n-1],t\in[0,w-1]}$ *and* $\mathsf{BRK}_{\mathsf{sum}} = \mathrm{RGSW}_{\mathbf{z}}(X^{-\sum_i s_i})$, *outputs a ciphertext* $\mathbf{c} \in \mathrm{RLWE}_{\mathbf{z}}\left(f(Y) \cdot Y^{b+\langle \mathbf{a}, \mathbf{s}\rangle}\right)$.

A proof of this theorem will be given in Appendix C.

Theorem 3. *Given Algorithm 6, on input a ciphertext* $(\mathbf{a}, b) \in \mathrm{LWE}_{q,\mathbf{s}}(m)$, *an accumulator* $\mathsf{acc} = (0, f(Y^g) \cdot Y^{bg})$, *keys* $\mathsf{SWK} = \{\mathrm{RLWE}'_{\mathbf{z}}\ (\mathbf{z}(X^{g^v}))\}_{v\in[1,2w-1]}$,

$\mathsf{BSK}^{\pm} = \{\mathrm{RGSW}_{\mathbf{z}(X^{g^t})}(X^{\pm s_i})\}_{i\in[0,n-1],t\in[0,w-1]}$ *and* $\mathsf{BRK}_{\mathsf{sum}} = \mathrm{RGSW}_{\mathbf{z}}(X^{-\sum_i s_i})$, *outputs a ciphertext* $\mathbf{c} \in \mathrm{RLWE}_{\mathbf{z}}(f(Y)\cdot Y^{b+\langle \mathbf{a},\mathbf{s}\rangle})$.

The proof of this theorem follows in a similar manner in Theorem 2.

4 Analysis and Comparisons

To compare the computational complexity and bootstrapping key size of several blind rotation schemes, we use the following parameters:

- q, LWE modulus;
- n, lattice parameter for the LWE scheme;
- Q, RLWE/RGSW modulus used in the core bootstrapping procedure based on an accumulator;
- N, ring dimension for RLWE/RGSW;
- B_{ep}, gadget base for $\circledast$, which breaks integers mod Q into d_{ep} digits;
- B_{ks}, gadget base, which breaks integers mod Q_{ks} into d_{ks} digits;
- B_r, gadget base in AP, which breaks integers mod q into d_r digits;
- w, window size of automorphisms;
- σ^2, error variance of a fresh RLWE ciphertext.

The comparison of computational complexity, key size, and error is summarized in Table 1. For the purpose of evaluating the performance of the blind rotation algorithms, we quantify their time complexity based on the number of $\mathcal{R}\odot\mathrm{RLWE}'$ products executed. A single RLWE $\circledast$ RGSW operation requires two $\odot$ multiplications, whereas key switching only involves one $\odot$ multiplication. Consequently, both the $\circledast$ product and key switching can be effectively expressed in terms of the $\odot$ operation. The $\odot$ operation itself serves as a fundamental abstraction in FHEW, as well as in its torus variant TFHE [13]. In prior studies of FHEW-like homomorphic encryption schemes, an alternative complexity measure often used is the number of NTT/FFT operations performed by the algorithm. Since each $\odot$ operation requires $(d+1)$ NTT operations (with d being the dimension of a gadget vector), it is straightforward to convert the count of $\odot$ products to the number of required NTT operations.

In a similar fashion, we assess the memory consumption of the various blind rotation algorithms by evaluating the total number of RLWE' ciphertexts. The blind rotation keys in all algorithms consist of a series of RGSW and RLWE' ciphertexts, with each RGSW ciphertext encompassing two RLWE' ciphertexts.

To estimate the variance σ^2_{acc} introduced by the blind rotation procedure, we follow an approach described in [26, 14]. The overall error for algorithms utilizing blind rotation, such as FHEW/TFHE bootstrapping [26, 14, 13] can be conveniently estimated using this variance. According to Lemma 1, the error variance induced by a single $\odot$ operation under canonical gadget decomposition is expressed as $\sigma^2_{\odot} = d_{\mathsf{ep}} N \frac{B^2_{\mathsf{ep}}}{12}\sigma^2$. For approximate gadget decomposition, the error variance introduced by the $\odot$ and $\circledast$ operations are, respectively, given as $\sigma'^2_{\odot} = d_{\mathsf{ep}} N \frac{B^2_{\mathsf{ep}}}{12}\sigma^2 + \frac{1}{3}\epsilon^2_{\mathsf{ep}}$ and $\sigma'^2_{\circledast} = d_{\mathsf{ep}} N \frac{B^2_{\mathsf{ep}}}{6}\sigma^2 + \frac{N\sigma^2+1}{3}\epsilon^2_{\mathsf{ep}}$ for Gaussian secrets.

In AP, the operation $\circledast$ is executed by encrypting the monomial with RGSW, which results in an additive error characterized by a variance of $2{\cdot}\sigma^2_{\odot}$. In LMKC+ and LLWW+ methods, the homomorphic automorphism operation due to key switching introduces an additive error with variance $\sigma^2_{\odot}$. Thus the variance σ^2_{acc} can be estimated as $\sigma^2_{\odot}$ multiplied by the number of $\odot$ operations. In GINX, due to the preprocessing of RGSW ciphertexts before $\circledast$ multiplications, each $\circledast$ introduces an additive error with variance $4 \cdot \sigma^2_{\odot}$.

Table 1. Complexity, key size, and error variance of each blind rotation technique. Key size (# keys) is the number of RLWE$'$ ciphertexts, and computational complexity (# mult) is the number of $\mathcal{R}_Q \odot \text{RLWE}'$. $\kappa \approx N(1 - e^{-n/N})$.

Method	# keys	# mult	σ^2_{acc}
AP	$2d_r(B_r-1)n$	$2d_r\left(1-\frac{1}{B_r}\right)n$	$2d_r\left(1-\frac{1}{B_r}\right)n\sigma^2_{\odot}$
GINX*	$4n$	$2n$	$8n\sigma^2_{\odot}$
LMKC+	$2n+w+3$	$2n+\frac{w-1}{w}\kappa+\frac{N}{w}+2$	$(n+1)\sigma'_{\circledast}+(\frac{w-1}{w}\kappa+\frac{N}{w})\sigma'^2_{\odot}$
LLWW+	$4n+w+2$	$2n+\frac{w-1}{w}\kappa+\frac{N}{2w}+2$	$(n+1)\sigma'^2_{\circledast}+(\frac{w-1}{w}\kappa+\frac{N}{2w})\sigma'^2_{\odot}$
Algorithm 5	$2nw+2w+1$	$2n+\frac{N}{w}+2$	$(n+1)\sigma'^2_{\circledast}+\frac{N}{w}\sigma'^2_{\odot}$
Algorithm 6	$4nw+2w+1$	$2n+\frac{N}{2w}+2$	$(n+1)\sigma'^2_{\circledast}+\frac{N}{2w}\sigma'^2_{\odot}$

As shown in Table 1, the total key size required by our algorithm for blind rotation and key switching is approximately w times larger than the original size prior to the improvement. However, the value of w required to achieve the optimal runtime for each algorithm may vary. Furthermore, the LWE key switching operation introduces an additional overhead. On the whole, the overall increase in bootstrapping key size remains modest.

Regarding time complexity, we observe that the theoretical complexity of blind rotation in Algorithm 5 and Algorithm 6 is higher than that of GINX*. However, this theoretical analysis should not be taken to mean that the algorithm with the smaller complexity expression will necessarily exhibit better runtime performance in practice. This discrepancy arises because different sets of parameters are used to achieve the same security level. For instance, while ternary GINX* exhibits the smallest theoretical expressions for computational complexity in this analysis, our proposed blind rotation algorithm outperforms GINX* in practice for two main reasons. First, our algorithm experiences less noise growth compared to GINX*, allowing it to operate with a smaller parameter set. Second, by leveraging Gaussian secrets, our algorithm can achieve the same level of security with a smaller value of n, without incurring any performance penalties.

5 Implementation

In this section, we present the implementation results of our new blind rotation algorithms. In our implementation, we optimize performance by selecting

the appropriate window size to reduce the number of key switching operations. We compare our approach with the AP, GINX, LMKC+, and LLWW+ blind rotation techniques.

5.1 Parameter Sets

Similar to [12, 14, 26], the parameters are set to ensure a decryption failure upper bound of less than 2^{-32} while achieving the highest possible efficiency under each respective algorithm to ensure a fair comparison. Table 2 presents optimized parameter sets from the OpenFHE library for FHEW schemes. Based on these criteria, we propose the following 128-bit secure parameter sets: STD_128 for AP/GINX with ternary secrets, STD128_LMKC+ for LMKC+/Algorithm 5 with Gaussian secrets, and 128_OURS for Algorithm 6 with Gaussian secrets.

Table 2. Optimized paramater sets for FHEW schems [22]

Parameter set	key	n	q	N	Q	Q_{ks}	B_{ep}	B_{ks}	B_{r}	w
STD_128	ternary	503	1024	1024	2^{27}	2^{14}	512	32	32	$\times$
STD128_LMKC+	$\sigma = 3.19$	447	1024	1024	2^{28}	2^{14}	1024	32	$\times$	10
128_OURS	$\sigma = 3.19$	447	1024	1024	2^{28}	2^{14}	1024	32	$\times$	5

Theoretical estimates. We use the approach from [14] to write the error of a refreshed ciphertext as a Gaussian of standard deviation

$$\beta = \sqrt{\frac{q^2}{Q_{\mathsf{ks}^2}} \left(\frac{Q_{\mathsf{ks}}^2}{Q^2} \sigma_{\mathsf{acc}}^2 + \sigma_{\mathsf{ms1}}^2 + \sigma_{\mathsf{ks}}^2 \right) + \sigma_{\mathsf{ms2}}^2} \ ,$$

where σ_{ms1}^2, σ_{ks}^2, and σ_{ms2}^2 denote the error variances introduced by modulus switching from Q to $Q_{\mathbf{ks}}$, key switching from $\boldsymbol{z}$ to $\boldsymbol{s}$, and modulus switching from $Q_{\mathbf{ks}}$ to q, respectively. We have

$$\sigma_{\mathrm{ms1}}^2 = \frac{\|\boldsymbol{z}\|^2 + 1}{12},\ \sigma_{\mathrm{ks}}^2 = \sigma^2\, N d_{\mathrm{ks}},\ \sigma_{\mathrm{ms2}}^2 = \frac{\|\boldsymbol{s}\|^2 + 1}{12}.$$

We assume $\|\boldsymbol{z}\| \leq \sqrt{2N/3}$ and $\|\boldsymbol{s}\| \leq \sqrt{2n/3}$ for ternary secrets [14], and $\|\boldsymbol{z}\| = \sqrt{N\sigma^2}$ and $\|\boldsymbol{s}\| = \sqrt{n\sigma^2}$ for Gaussian secrets. The ciphertext $\mathrm{LWE}_{q,\mathbf{s}}(q/4 \cdot m)$ exhibits the largest noise, characterized by a standard deviation of $\sqrt{2}\beta$. Decryption fails when the noise in $\mathrm{LWE}_{q,\mathbf{s}}(q/4 \cdot m)$ exceeds $q/8$. As a result, the decryption failure probability per NAND operation is given by 1-erf($\frac{q/8}{2\beta}$).

5.2 Runtime Results

To ensure a fair comparison of the bootstrapping algorithms, we implemented all of them using identical libraries and computing environments. The evaluation was conducted on a system with an 11th Gen Intel(R) Core(TM) i5-1135G7 processor running at 2.40 GHz, under Ubuntu 24.04.1 LTS. The code

was compiled using Clang 12, with the following CMake flags: NATIVE SIZE = 32,WITH_OPENMP = OFF, WITH_ NATIVEOPT = ON.

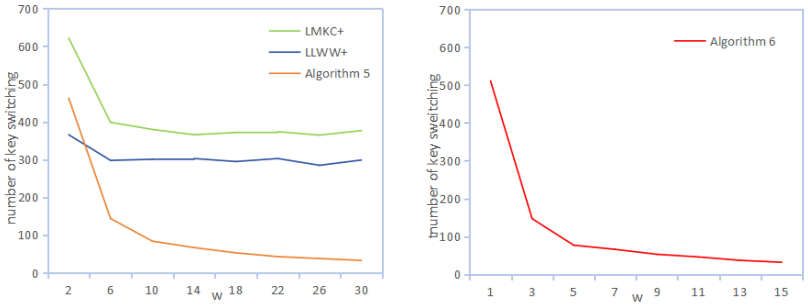

Fig. 2. The window size and corresponding number of key switching for different of bootstrapping algorithms

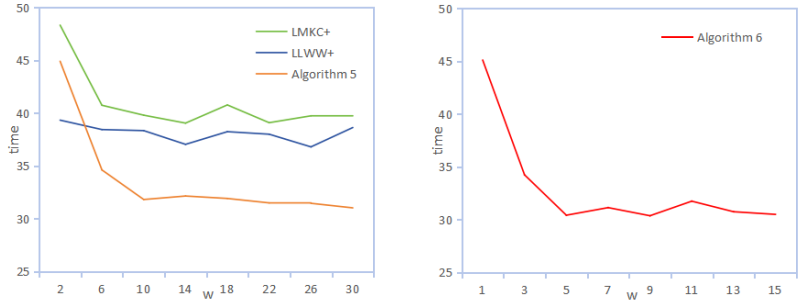

Fig. 3. The window size and corresponding time for different of bootstrapping algorithms

As shown in the figures, Figure 2 illustrates the relationship between window size and the number of key switching, while Figure 3 shows the correlation between window size and bootstrapping time for NAND gate evaluation in FHEW. For LMKC+ and LLWW+ methods, we observe that the number of key switching decreases as the window size w increases when w is small. However, once all empty sets are merged, the window size no longer influences the number of key switching. In contrast, for Algorithm 5, the number of key switching consistently decreases with increasing window size w. The bootstrapping time stabilizes once w exceeds 10, which can be attributed to the fact that, for $w \geq 10$, the number of key switching operations becomes very small, meaning the runtime is pri-

marily determined by the inherent randomness of the algorithm. At this point, the practical runtime required for bootstrapping is approximately equal to that for performing only n external products. Specifically, for Algorithm 6, the bootstrapping time remains relatively constant once the window size exceeds 5.

Table 3. Timing results (average of 400), the number of key switching, bootstrapping key size, and failure probability for FHEW bootstrapping (NAND gate)

Method	Number	Time[ms]	Key size[MB]	Fail.prob
AP [14, 3]	×	55.49	1316.12	2^{-84}
GINX* [6, 20]	×	36.33	153.41	2^{-66}
LMKC+ [22]	380	39.79	92.1	2^{-34}
LLWW+ [23]	301	38.38	110.41	2^{-35}
Ours(Algorithm 5)	84	31.81	257.27	2^{-37}
Ours(Algorithm 6)	**77**	**30.41**	257.07	2^{-36}

We set the window size $w = 10$ for LMKC+, LLWW+, and Algorithm 5, and $w = 5$ for Algorithm 6. It is important to note that the entire bootstrapping key consists of both the evaluation keys for blind rotation and key switching keys. The size of the LWE key switching key is given by $(n+1)NB_{\mathsf{sk}}d_{\mathsf{ks}}\log Q_{\mathsf{ks}}$ bits. The resulting values for the number of key switching, bootstrapping time, key size, and the probability of decryption failure are summarized in Table 3. The results indicate that Algorithm 6 performs only 77 key switching, representing a reduction of 80% and 74% compared to LMKC+ and LLWW+, respectively. Moreover, the running time of bootstrapping is 30.41 ms, which corresponds to a 20.7% reduction in bootstrapping time compared to LLWW+.

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: WAHC 2022 – 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 53–63 (2022)
2. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., et al.: Homomorphic Encryption Standard. Protecting privacy through homomorphic encryption pp. 31–62 (2021)
3. Alperin-Sheriff, J., Peikert, C.: Faster Bootstrapping with Polynomial Error. In: CRYPTO 2014 - 34th Annual Cryptology Conference. pp. 297–314. Springer (2014)
4. Angel, S., Chen, H., Laine, K., Setty, S.: PIR with Compressed Queries and Amortized Query Processing. In: SP 2018 - 2018 IEEE Symposium on Security and Privacy. pp. 962–979. IEEE (2018)
5. Blatt, M., Gusev, A., Polyakov, Y., Goldwasser, S.: Secure Large-Scale Genome-Wide Association Studies Using Homomorphic Encryption. Proceedings of the National Academy of Sciences **117**(21), 11608–11613 (2020)

6. Bonte, C., Iliashenko, I., Park, J., Pereira, H.V.L., Smart, N.P.: FINAL: Faster FHE Instantiated with NTRU and LWE. In: ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security. pp. 188–215. Springer Nature (2022)
7. Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: CRYPTO 2012 - 32nd Annual Cryptology Conference. pp. 868–886. Springer (2012)
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. ACM Transactions on Computation Theory **6**(3), 1–36 (2014)
9. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In: CRYPTO 2011 - 31st Annual Cryptology Conference. pp. 505–524. Springer (2011)
10. Chen, H., Laine, K., Rindal, P.: Fast Private Set Intersection from Homomorphic Encryption. In: ACM CCS 2017 - 24th ACM SIGSAC Conference on Computer and Communications Security. pp. 1243–1255. ACM (2017)
11. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic Encryption for Arithmetic of Approximate Numbers. In: ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security. pp. 409–437. Springer (2017)
12. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster Fully Homomorphic Encryption: Bootstrapping in Less than 0.1 Seconds. In: ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security. pp. 3–33. Springer (2016)
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast Fully Homomorphic Encryption over the Torus. Journal of Cryptology **33**(1), 34–91 (2020)
14. Ducas, L., Micciancio, D.: FHEW: Bootstrapping Homomorphic Encryption in Less than a Second. In: EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 617–640. Springer (2015)
15. Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive p. 144 (2012)
16. Gama, N., Izabachène, M., Nguyen, P.Q., Xie, X.: Structural Lattice Reduction: Generalized Worst-Case to Average-Case Reductions and Homomorphic Cryptosystems. In: EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 528–558. Springer (2016)
17. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: STOC 2009 - 41st Annual ACM Symposium on Theory of Computing. pp. 169–178. ACM (2009)
18. Gentry, C., Sahai, A., Waters, B.: Homomorphic Encryption from Learning with Errors: Conceptually Simpler, asymptotically-faster, attribute-based. In: CRYPTO 2013 - 33rd Annual Cryptology Conference. pp. 75–92. Springer (2013)
19. Han, K., Hong, S., Cheon, J.H., Park, D.: Logistic Regression on Homomorphic Encrypted Data at Scale. In: AAAI 2019 - The Thirty-Third AAAI Conference on Artificial Intelligence. pp. 9466–9471. AAAI Press (2019)
20. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General Bootstrapping Approach for RLWE-Based Homomorphic Encryption. IEEE Transactions on Computers **73**(1), 86–96 (2024)
21. 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 p. 767 (2023)

22. 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: EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 227–256. Springer (2023)
23. Li, Z., Lu, X., Wang, Z., Wang, R., Liu, Y., Zheng, Y., Zhao, L., Wang, K., Hou, R.: Faster NTRU-based Bootstrapping in Less than 4 Ms. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(3), 418–451 (2024)
24. Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings. Journal of the ACM **60**(6), 1–35 (2013)
25. Micciancio, D.: On the Hardness of Learning with Errors with Binary Secrets. Theory of Computing **14**(1), 1–17 (2018)
26. Micciancio, D., Polyakov, Y.: Bootstrapping in FHEW-like Cryptosystems. In: WAHC 2021 - 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 17–28. Association for Computing Machinery (2021)
27. Regev, O.: On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. Journal of the ACM **56**(6), 1–40 (2009)
28. Smart, N.P., Vercauteren, F.: Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In: PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography. pp. 420–443. Springer (2010)
29. Stehlé, D., Steinfeld, R.: Faster Fully Homomorphic Encryption. In: ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security. pp. 377–394. Springer (2010)
30. Wang, R., Wen, Y., Li, Z., Lu, X., Wei, B., Liu, K., Wang, K.: Circuit Bootstrapping: Faster and Smaller. In: EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 342–372. Springer (2024)
31. Xiang, B., Zhang, J., Deng, Y., Dai, Y., Feng, D.: Fast Blind Rotation for Bootstrapping FHEs. In: CRYPTO 2023 - 43rd Annual International Cryptology Conference. pp. 3–36. Springer (2023)

A LLWW+ Basic Blind Rotation Algorithm

Algorithm 7 Blind Rotation: LLWW+ [23] for $\frac{2N}{q} = 2^\theta, \theta \geq 1$

Input: $(\mathbf{a}, b)$, $\mathsf{acc} = (0, Y^{gb} \cdot f(Y^g))$, $\{\mathsf{BSK}^{\pm}{}_i = \mathrm{RGSW}_{\mathbf{z}}(X^{\pm s_i})\}_{i\in[0,n-1]}$, $\mathsf{BRK}_{\mathsf{sum}} = \mathrm{RGSW}_{\mathbf{z}}(X^{-\sum_i s_i})$, $\{\mathsf{ATK}_{g^u} = \mathrm{RLWE}'_{\mathbf{z}}(\mathbf{z}(X^{g^v}))\}_{v\in[1,w]}$
Output: $\mathbf{c}$

$\mathbf{a} = 2^\theta \mathbf{a} + \mathbf{1} \mod 2N$
$v \leftarrow 0$
for $(\ell = \frac{N}{2} - 1; \ell > 0; \ell = \ell - 1)$ **do**
  for $j \in I_\ell^+$ or $j \in I_\ell^-$ **do**
    if $j \in I_\ell^+$ **then**
      $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_j^+$
    if $j \in I_\ell^-$ **then**
      $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_j^-$
  $v \leftarrow v + 1$
  if $(I_{\ell-1} \neq \emptyset$ or $v = w$ or $\ell = 1)$ **then**
    $\mathsf{acc} = \mathsf{HomAuto}_{g^v}(\mathsf{acc}, \mathsf{ATK})$;
    $v \leftarrow 0$
for $j \in I_0^+$ or $j \in I_0^-$ **do**
  if $j \in I_0^+$ **then**
    $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_j^+$
  if $j \in I_0^-$ **then**
    $\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BSK}_j^-$
$\mathsf{acc} = \mathsf{acc} \circledast \mathsf{BRK}_{\mathsf{sum}}$
return $\mathbf{c} = \mathsf{acc}$

B The correctness of Algorithm 4

Proof. We begin by considering the initial accumulator acc, which can be treated as a trivial ciphertext under the secret key $\mathbf{z}(X^{w'_1})$. For $l = 0$, each $j \in I_0$, we apply the automorphism $X \to X^{w'_1}$ to the ciphertext BSK_{j,w_0} to obtain

$$\mathsf{BSK}'_j = \mathrm{RGSW}_{\mathbf{z}(X^{w'_1})}(X^{w_0 w'_1 s_j}).$$

Initially, the accumulator is defined as an $\mathsf{acc} = (0, f(X^{w'_1}) \cdot X^{bw'_1})$. The accumulator acc is updated sequentially as

$$\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{BSK}'_j,$$

for all $j \in I_0$. This process results in the ciphertext

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}(X^{w'_1})}\left(f(X^{w'_1}) \cdot X^{bw'_1 + w_0 w'_1 (\sum_{j\in I_0} s_j)}\right).$$

Next, we apply the automorphism $X \to X^{w_1 w_2'}$ to acc, yielding the updated ciphertext

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}(X^{w_2'})}\left(f(X^{w_2'}) \cdot X^{bw_2' + w_0 w_2'(\sum_{j\in I_0} s_j)}\right).$$

For $l < L-1$, let $j \in I_l$. Suppose, after the previous steps, we have the following expression for the accumulator

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}(X^{w'_{l+2}})}\left(f(X^{w'_{l+2}}) \cdot X^{bw'_{l+2} + w_0 w'_{l+2}(\sum_{j\in I_0} s_j) + \ldots + w_l w'_{l+2}\left(\sum_{j\in I_l} s_j\right)}\right).$$

Then for $l+1$, we apply the automorphism $X \to X^{w'_{l+2}}$ to $\mathsf{BSK}_{j,w_{l+1}}$, obtaining

$$\mathsf{BSK}'_j = \mathrm{RGSW}_{\mathbf{z}\left(X^{w'_{l+2}}\right)}(X^{w_{l+1} w'_{l+2} s_j}).$$

The accumulator acc is updated sequentially as

$$\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{BSK}'_j,$$

for all $j \in I_l$. Thus, we obtain the ciphertext

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}(X^{w'_{l+2}})}\left(f(X^{w'_{l+2}}) \cdot X^{bw'_{l+2} + w_0 w'_{l+2}(\sum_{j\in I_0} s_j) + \ldots + w_l w'_{l+2}\left(\sum_{j\in I_{l+1}} s_j\right)}\right).$$

Next, we apply the automorphism $X \to X^{w_{l+2} w'_{l+3}}$ to acc, yielding:

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}(X^{w'_{l+3}})}\left(f(X^{w'_{l+3}}) \cdot X^{bw'_{l+3} + w_0 w'_{l+3}(\sum_{j\in I_0} s_j) + \ldots + w_l w'_{l+3}\left(\sum_{j\in I_{l+1}} s_j\right)}\right).$$

Finally, for $l = L$, after executing the steps in line 4 for the last time, the resulting ciphertext is

$$\mathsf{acc} = \mathrm{RLWE}_{\mathbf{z}}\left(f(X) \cdot X^{b + w_0(\sum_{j\in I_0} s_j) + \ldots + w_L\left(\sum_{j\in I_L} s_j\right)}\right) = \mathrm{RLWE}_{\mathbf{z}}\left(f(X) \cdot X^{b + \langle \mathbf{a}, \mathbf{s}\rangle}\right).$$

C The correctness of Algorithm 5

Proof. For the sets $I_0^-, ..., I_{\frac{N}{2}-1}^-$, let $v_1, \ldots, v_k$ be the exponents of the k automorphisms that need to be applied after each non-empty set. We are given that $v_1 + v_2 + ... + v_k = \frac{N}{2} - 1$. We decompose each exponent as $v_i = v_i' + w \cdot v_i''$, where $v_i' = v_i \bmod w$, and $v_i'' = \lfloor v_i'/w \rfloor$. Let $\hat{I}_1, ..., \hat{I}_k$ be the non-empty sets. For writing convenience, the vectors (d_1, d_2, d_3) represent the ciphertext $f(Y^{-g^{d_1}}) \cdot Y^{-bg^{d_1}} \cdot X^{d_2}$ under the secret key $\mathbf{z}(X^{g^{d_3}})$. Initially, the accumulator is defined as $\mathsf{acc} = (0, f(Y^{-g})) \cdot Y^{-bg}$.

One of the following two cases will occur: $I^-_{\frac{N}{2}} \neq \emptyset$ or $I^-_{\frac{N}{2}} = \emptyset$. The proof for both cases is similar, so, without loss of generality, we can assume that $I^-_{\frac{N}{2}} \neq \emptyset$. Then we have $\hat{I}_k = I^-_{\frac{N}{2}-1}$, and the sum is expressed as

$$\sum_{j\in I_0^-} s_j + \cdots + g\left(\sum_{j\in I^-_{N/2-1}} s_j\right) = g^{v_1}\left(\sum_{j\in \hat{I}_1} s_j + g^{v_2}\left(\sum_{j\in \hat{I}_2} s_j + \cdots + g^{v_k}\left(\sum_{j\ \hat{I}_k} s_j\right)\right)\right) \mod 2N.$$

For v_k, after the first application of the steps in line 5 for all $j \in I^-_\ell$, the resulting ciphertext is

$$\left(1, \sum_{j\in\hat{I}_k} X^{s_j}, 0\right).$$

Algorithm 5 then performs the steps in line 8 v''_k times, resulting in

$$\left(1 + wv''_k, g^{wv''_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right), 0\right).$$

By performing lines 6 and 10, the accumulator is updated to

$$\left(1 + v_k, g^{v_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right), v'_k\right).$$

At this point, we don't perform key switching like LMKC+, and $r = v'_k$. We have

$$\mathsf{BSK}_j = \mathrm{RGSW}_{\mathbf{z}(X^{g^{v'_k}})}(X^{s_j}).$$

After executing the steps in line 5 for all $j \in I^-_\ell$ (with the updated value of ℓ), we obtain the following ciphertext

$$\left(1 + v_k, \sum_{j\in\hat{I}_{k-1}} X^{s_j} + g^{v_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right), v'_k\right).$$

As the algorithm progresses, for v_{k-1}, if $v''_{k-1} \neq 0$, the ciphertext becomes

$$\left(1 + v_k + v_{k-1}, g^{v_{k-1}}\left(\sum_{j\in\hat{I}_{k-1}} X^{s_j} + g^{v_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right)\right), v'_{k-1}\right).$$

Otherwise, the result is

$$\left(1 + v_k + v_{k-1}, g^{v_{k-1}}\left(\sum_{j\in\hat{I}_{k-1}} X^{s_j} + g^{v_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right)\right), v'_k + v'_{k-1}\right)$$

or

$$\left(1+v_k+v_{k-1}, g^{v_{k-1}}\left(\sum_{j\in\hat{I}_{k-1}} X^{s_j}+g^{v_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right)\right), 0\right).$$

Complete the for loop with respect to i. For v_i, suppose that $v_i'' \neq 0$ (in the event that this assumption does not hold, we proceed by considering the previous non-zero v_i''; if all $v_i'' = 0$, the algorithm is obviously correct). It is straightforward to observe, for $i \geq 2$, we have

$$\left(1+v_k+...+v_i, g^{v_i}\left(\sum_{j\in\hat{I}_i} X^{s_j}+...+g^{v_{k-1}}\left(\sum_{j\in\hat{I}_{k-1}} X^{s_j}+g^{v_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right)\right)\right), v_i'\right).$$

Specially, when $i = 1$, we get the final result

$$\begin{aligned}&\left(1+v_k+...+v_1, g^{v_1}\left(\sum_{j\in\hat{I}_1} X^{s_j}+...+g^{v_{k-1}}\left(\sum_{j\in\hat{I}_{k-1}} X^{s_j}+g^{v_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right)\right)\right), 0\right)\\ =&\left(\frac{N}{2}-1, \sum_{j\in I_0^-} s_j+\cdots+g\left(\sum_{j\in I^-_{N/2-1}} s_j+g\left(\sum_{j\in I^-_{N/2-2}} s_j\right)\right), 0\right).\end{aligned} \tag{1}$$

If $v_{i-1}'' \neq 0$, the situation reduces to the previous case with $v_i'' \neq 0$. When

$$v_{i-1}'' = v_{i-2}'' = ... = v_{i-j'}'' = 0,\ v_{i-j'-1}'' \neq 0,\ 1 \leq j' \leq i-2.$$

For $v_{i-j''},\ 1 \leq j'' \leq j'$, we have

$$\left(1+v_k+...+v_{i-j''}, g^{v_{i-j''}}\left(\sum_{j\in\hat{I}_{i-j''}} X^{s_j}+...+g^{v_k}\left(\sum_{j\in\hat{I}_k} X^{s_j}\right)\right), r\right),$$

where r is the sequential sum from v_i' to $v'_{i-j''}$, following the rule that once the sum exceeds w, it resets to 0 and continues adding. Then situation reduces to the previous case with $v_i'' \neq 0$. When

$$v_{i-1}'' = v_{i-2}'' = ... = v_1'' = 0,$$

we get the final result (1) when $l = 0$.

Next, we execute the steps from line 13 to line 15 and repeat the above process for I_i^+, where $i = 0, ..., \frac{N}{2}-1$. The procedure is analogous to that for I_i^- with $i \in [0, N/2-1]$, and thus is omitted for brevity.