IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 3, pp. 418–451. DOI:10.46586/tches.v2024.i3.418-451

Faster NTRU-based Bootstrapping in less than 4 ms

Zhihao Li[1,2], Xianhui Lu[1,2](✉), Zhiwei Wang[1,2], Ruida Wang[1,2], Ying Liu[1,2], Yinhang Zheng[1,2], Lutan Zhao[1,2], Kunpeng Wang[1,2] and Rui Hou[1,2]

[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
luxianhui@iie.ac.cn

Abstract. Bootstrapping is a critical technique in constructing fully homomorphic encryption (FHE), which serves to refresh the noise in FHE ciphertexts, enabling an arbitrary number of homomorphic operations. Among published results, the TFHE-rs library [Zam22] offers the fastest bootstrapping implementation on CPU platforms by taking advantage of AVX-512 instructions.
In this paper, we improve the efficiency of the bootstrapping algorithm based on the NTRU problem. First, we introduce the approximate gadget decomposition method tailored for NTRU ciphertext, reducing the number of NTT operations required for external products. Second, by integrating the approximate decomposition and key unrolling techniques, we improve the performance of CMux-based blind rotation. Third, for the automorphism-based blind rotation method, we present a hybrid window size technique that reduces the number of automorphisms by 34% compared to recent work [XZD+23](in Crypto23).
Subsequently, we implement the proposed bootstrapping algorithm on the CPU platform with AVX instructions. Experimental results demonstrate that our method only takes 3.8ms, which achieves a 1.8× speedup compared to the TFHE-rs library. Finally, we propose an efficient FPGA accelerator based on the CMux method, which not only achieves the best performance but also exhibits high throughput advantages. Our accelerator can improve performance by 2× compared to state-of-the-art FPGA implementations (e.g., FPT).

Keywords: Fully Homomorphic Encryption · Bootstrapping · NTRU · Key Unrolling · Hybrid Window Size · AVX Instruction · FPGA Accelerator

1 Introduction

Fully homomorphic encryption (FHE) is a powerful cryptographic technology for privacy-preserving computation, allowing arbitrary computations on encrypted data without the need for decryption. In 2009, Gentry [Gen09] introduced the first FHE scheme based on ideal lattices. Specifically, the scheme proposed a specialized bootstrapping procedure capable of transforming a scheme with limited homomorphic properties into a fully homomorphic one by evaluating the decryption circuit. Bootstrapping is considered more intricate both in theory and practice when compared to other homomorphic operations. Thus, it has emerged as the primary efficiency bottleneck in FHE.

At present, existing FHE schemes are primarily based on three assumptions: Ring Learning With Errors (LWE/RLWE), NTRU, and Approximate Greatest Common Divisor (AGCD). Among these, AGCD-based schemes necessitate extensive parameters, rendering

Received: 2024-01-15 Accepted: 2024-03-15 Published: 2024-07-18

them impractical for real-world applications. Conversely, (R)LWE and NTRU-based schemes showcase significant promise due to their robust algebraic structure and efficient polynomial evaluation employing the Fast Fourier Transform (FFT) and Number Theoretic Transform (NTT). Broadly, these FHE schemes can be categorized into three classes based on their data type. The first class facilitates modular arithmetic within a finite field by employing techniques like Single Instruction Multiple Data (SIMD) mode for packing multiple data into a single ciphertext. Noteworthy examples include the RLWE-based BGV [Bra12], BFV [FV12] schemes, and NTRU-based [BLLN13, LATV12] schemes. The second class, exemplified by the CKKS scheme [CKKS17], supports approximate homomorphic encryption and enables packing operations over complex vectors. However, bootstrapping for both classes often entails extremely computationally expensive techniques like homomorphic bit extraction or homomorphic modular reduction.

The last class focuses on evaluating boolean circuits, which are well-suited for tasks like comparison and decision diagram computation. This class includes (R)LWE-based schemes such as FHEW schemes [DM15, MP21, LMK+23] and TFHE scheme [CGGI20], as well as NTRU-based schemes [BIP+22, XZD+23], which offer faster bootstrapping for an LWE ciphertext. In more detail, given an LWE ciphertext $\mathsf{ct} = (\mathbf{a}, b)$, the bootstrapping process can be divided into two steps. First, it homomorphically evaluates the RLWE or NTRU ciphertext of $X^{b + \sum_{i=0}^{n-1} a_i s_i \bmod q}$ through a procedure known as blind rotation. Next, a refreshed LWE ciphertext can be extracted using a predefined test polynomial. The two steps can be integrated during the implementation of algorithm. Currently, there are three strategies employed for performing the blind rotation procedure.

- The first one is the AP method [DM15] that uses a_i as a selector to pick all the evaluation keys that encrypt $E(a \cdot s_i)$. It can support arbitrary key distributions, and requires large blind rotation keys to store multiple encryptions $E(a \cdot s_i)$ for every secret key element s_i. These keys are accumulated through external products to generate the final result.

- The second one is to design a special CMux gate proposed by [CGGI16] that can use $E(s_i)$ as a selector between the original accumulator acc and a modified one $\mathsf{acc} \cdot X^{a_i}$. This approach requires n iterations and is more suitable for binary or ternary secret key distributions.

- Recently, Lee et al. [LMK+23] proposed the third method by utilizing the ring automorphism technique, which can also support arbitrary key distribution. Compared to the AP method, this alternative is more efficient and has smaller key sizes.

The comparisons of different schemes in terms of assumptions, key distributions, and blind rotation strategies are listed in Table 1.

When it comes to computational efficiency, the TFHE-rs library [Zam22] stands as the state-of-the-art CPU implementation, relying on the binary CMux gate approach. This implementation stands out by integrating techniques like approximate gadget decomposition and advanced vector extensions (AVX) instructions, which prove instrumental in improving the efficiency of blind rotation. It's important to note that real-world applications often involve thousands of gates, leading to significant performance overhead in the TFHE-based mode. As a result, hardware acceleration has garnered widespread attention as a promising solution for performance enhancement. Extensive efforts have been devoted to exploring hardware-accelerated bootstrapping across various platforms, encompassing GPUs [MAAM20], FPGAs [GNT+21], and ASICs [JLJ22].

Table 1: Comparisons of bootstrapping schemes.

Schemes	Assumption	Key distribution	Methods
[DM15]	RLWE	Gaussian	AP
[Per21]	AGCD	Gaussian	AP
[CGGI16]	RLWE	Binary	CMux
[BMMP18]	RLWE	Binary	CMux
[MP21]	RLWE	Ternary	CMux
[BIP$^+$22]	NTRU	Ternary	CMux.
[LMK$^+$23]	RLWE	Gaussian	Auto.
[XZD$^+$23]	NTRU	Gaussian	Auto.

1.1 Our Techniques and Contributions

Our main contribution lies in the efficient implementation of two NTRU-based gate bootstrapping schemes, achieved through algorithmic-level and implementation-level optimizations. At the algorithmic level, we significantly improve the efficiency of blind rotation by incorporating some advanced techniques as follows.

- **Approximate gadget decomposition with NTRU ciphertext.** We utilize the approximate gadget decomposition to accelerate the external product and reduce the key sizes of blind rotation for the NTRU accumulator. Compared to the exact gadget decomposition, approximate decomposition can reduce the decomposition length and the number of polynomial multiplications by introducing an approximation factor.
- **Key unrolling for NTRU-based CMux gate bootstrapping.** We apply the key unrolling technique [BMMP18] to NTRU-based CMux gate bootstrapping, which can further improve the efficiency of blind rotation. The new CMux gate can get the accumulator as $\mathrm{NTRU}_{f,Q}(X^{a_{2i}s_{2i}+a_{2i+1}s_{2i+1}})$ instead of $\mathrm{NTRU}_{f,Q}(X^{a_i s_i})$, and the entire blind rotation process only need to performs $n/2$ such CMux gates. Consequently, we can reduce the number of NTTs by almost half in blind rotation.
- **Improved automorphism-based blind rotation.** We propose a new automorphism-based blind rotation algorithm with NTRU accumulator by merging the consecutive empty sets and symmetric sets. This approach enables us to effectively reduce the number of required automorphisms to approximately $\frac{3}{5}n$.

As demonstrated in Table 9, our methods outperform existing schemes in terms of computational complexity, regardless of whether CMux or automorphism methods are employed. In addition, for automorphism-based blind rotation, our scheme also outperforms other schemes in terms of key size. At the implementation level, we provide the state-of-the-art CPU and FPGA implementations by optimizing the underlying operators and data flow.

- **CPU implementation with AVX instructions.** We use AVX instructions to improve building blocks of blind rotation, such as the approximate decomposition, NTT, and Hadamard multiplications, enabling them to be computed in parallel. In particular, NTT can obtain 16 parallelisms within 32bit word length under AVX-512 instruction, and the running time of the proposed CMux method is only 3.8 ms, which is 1.8 times faster than the state-of-the-art TFHE-rs [Zam22] library.

- **FPGA implementation.** We introduce a pioneering FPGA accelerator tailored for CMux-based bootstrapping to fully exploit the inherent parallelism in FHE computations. This accelerator incorporates a high-throughput (I)NTT design, schedule-optimized vector chaining, and a carefully designed memory architecture. To ensure a fair comparison with state-of-the-art FPGA implementation FPT [VBDV22], we meticulously fine-tuned the scheme's parameters. The experimental results demonstrate that, under the same decryption failure rate, our implementation outperforms FPT with a 2$\times$ speedup.

In this paper, we focus on the gate bootstrapping mode of the FHEW and TFHE framework. It first performs homomorphic addition on the LWE ciphertext, and then refreshes the ciphertext through the bootstrapping procedure. The bootstrapping procedure involves a large number of NTT and Hadamard multiplication, whereas the first step is the addition of vector, whose computational cost is negligible compared to bootstrapping, as shown in the FHEW [DM15] and TFHE [CGGI16] schemes. Thus, we significantly improve the gate bootstrapping mode, which is 1.8 times faster than TFHE-rs implementation.

1.2 Related Work

The NTRU problem and the corresponding cryptosystem date back to the work by Hoffstein, Pipher, and Silverman [HPS98]. One of the earliest FHE schemes is [LATV12], and its scale-invariant version YASHE [BLLN13]. However, the security of NTRU-based cryptography has been controversial in recent years. Typically, Ducas et al. [DvW21] provide a tighter prediction for fatigue point as $Q \in O(N^{2.484})$. Consequently, the sublattice attack has rendered NTRU-based FHE schemes vulnerable, since these schemes require an exponentially large modulus Q relative to the polynomial parameter N.

Afterward, Bonte et al. [BIP+22] and independently Kluczniak et al. [Klu22] presented a similar gate bootstrapping algorithm in the FHEW/TFHE framework based on NTRU and LWE assumptions. Since the blind rotation incurs only polynomial error growth, the modulus of these schemes satisfies the safety bounds. Additionally, their results show that the NTRU-based bootstrapping is more effective and uses smaller key sizes than the original TFHE scheme, which directly benefits from a single polynomial as opposed to a pair of polynomials in RLWE-based schemes. In recent work, Xiang et al. [XZD+23] proposed a bootstrapping method based on NTRU that utilizes ring automorphisms and extends the sample extraction technique to NTRU ciphertext, thereby improving upon the NTRU-to-LWE key switching method presented in [BIP+22].

The key unrolling technique, introduced in [BMMP18], aims to reduce the number of iterations required for the CMux gate in the TFHE scheme. Given an unrolling factor m, the technique is to compute m dimensions of the secret key s simultaneously, which can reduce the number of iterations from n to n/m. The original BKU technique in [BMMP18] works with $m = 2$, while Joye et al. [JP22a] extends it to generalized unrolling factors, as well as arbitrary secret key distributions. It is important to note that while this technique can improve efficiency, it also leads to increased noise growth and larger blind rotation key sizes. Thus, there exists a trade-off between achieving practical scheme efficiency and considering factors like noise and key sizes.

Strong data dependency of blind rotation and high memory overloads is challenging in hardware implementation. Morshed et al. [MAAM20] ports the blind rotation to GPU and leverages the parallel processing capabilities of GPU with their multitude of cores for boolean and arithmetic circuits. This enables efficient execution of the bootstrapping process and improves overall performance. Gener et al. [GNT+21] introduces the first FPGA-based programmable vector engine for bootstrapping. However, the engine was developed without algorithmic optimization and thus exhibits high computing latency. Moreover, [VBDV22] develops the pipelined FFTs method on FPGA that naturally supports a streaming architecture and reduces the latency. However, this work sacrifices the decryption failure rate for the algorithm level. Jiang et al. proposed MATCHA [JLJ22], the first customized ASIC accelerator for bootstrapping. The accelerator adopts and extends the bootstrapping unrolling scheme proposed by [BMMP18].

1.3 Paper Organization

The rest of the paper is organized as follows. We provide the necessary background knowledge and some general tools in FHE schemes in Section 2. In Section 3, we show the NTRU-based approximate decomposition and improved CMux-based bootstrapping algorithm. In Section 4, we demonstrate some optimization techniques for improving the automorphism-based bootstrapping algorithm. In Section 5, we present some details of the algorithm parameters and implementation, as well as some experimental results. In Section 6, we provide an FPGA implementation based on the improved CMux gate method. Finally, we conclude the paper in Section 8.

2 Background

2.1 Notation

The lower-case bold letters indicate vectors, e.g., $\mathbf{a}$, and upper-case bold letters indicate matrices like $\mathbf{A}$. The inner product between two vectors is denoted by $\langle \mathbf{a}, \mathbf{b} \rangle$. For a real number r, we write the floor, ceiling, and round functions as $\lfloor r \rfloor$ $\lceil r \rceil$ $\lfloor r \rceil$, respectively. We denote the infinity norm $||\mathbf{u}||$ for a vector $\mathbf{u}$ and $\mathbb{Z}_q$ the integer ring $\mathbb{Z}/q\mathbb{Z}$ and the scope is $[-q/2, q/2) \cap \mathbb{Z}$, and sometimes $[x]_Q$ is used to denote $x \mod Q$. We use $\leftarrow$ to denote randomly choosing an element from uniform and Gaussian distributions.

Let N be a power of 2, we denote the $2N$-th cyclotomic ring by $\mathbb{Z}[X]/(X^N + 1)$, and the quotient ring is $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$ with coefficients in $\mathbb{Z}_Q$. For a polynomial s, we denote $\boldsymbol{\phi}(s) = (s_0, ..., s_{N-1}) \in \mathbb{Z}_q^N$ as the vector of coefficient. Furthermore, for a random variable $a \in \mathbb{Z}_q$, we denote $\mathsf{Var}(a)$ as the variance of a. Similarly, for a ring element $a = a_0 + a_1 X \cdots + a_{N-1} X^{N-1} \in \mathcal{R}$, we define $\mathsf{Var}(a) = \mathsf{Var}(\phi(a))$ as the variance among the coefficients of the polynomial a. Finally, the function $\mathsf{min}(x, y)$ is defined to return the value of the smaller number.

2.2 Gaussian Distribution

Gaussian Distribution. Given the Gaussian function $\rho_{\sigma,c}(x) = \exp\left(-\frac{|x-c|^2}{2\cdot\sigma^2}\right)$, where $\sigma, c \in \mathbb{R} \geq 0$, the Gaussian distribution is defined over $\mathbb{Z}$ as $\rho_{\sigma,c}(\mathbb{Z}) = \sum_{i=-\infty}^{\infty} \rho_{\sigma,c}(i)$. Here each element in $\mathbb{Z}$ is sampled with probability proportional to its probability mass function value under a Gaussian distribution over $\mathbb{R}$.

Discrete Gaussian Distribution. The discrete Gaussian distribution with standard deviation σ and mean c is a distribution on $\mathbb{Z}$ with the probability of $x \in \mathbb{Z}$ given by $\mathcal{D}_{\delta,c} = \rho_{\sigma,c}(x)/\rho_{\sigma,c}(\mathbb{Z})$. If c is omitted, then it is implicitly set to 0.

Subgaussian Distribution. The α-subgaussian for distribution V over $\mathbb{R}$ if the moment generating function satisfies $\mathbb{E}[\exp(tV)] \leq \frac{1}{2}\exp(\alpha^2 t^2)$ for all $t \in \mathbb{R}$, where $\mathbb{E}$ is the expectation function. It is easy to see that the variance of V satisfies $\mathsf{Var}(V) \leq \alpha^2$. Subgaussian random variables have an important property, i.e., Pythagorean additivity. For two random variables, α-subgaussian X and β-subgaussian Y, let a and $b \in \mathbb{Z}$, the random variable $a \cdot X + b \cdot Y$ satisfies $a^2 \cdot \alpha^2 + b^2 \cdot \beta^2$-subgaussian.

2.3 Digit Decomposition

For a fixed modulus Q and the decomposition base B, we define the decomposition length as $d = \lceil \log_B Q \rceil$ and the gadget vector as $\mathfrak{g} = \left(B^0, B, \cdots, B^{d-1}\right)$. Given a polynomial $a \in \mathcal{R}_Q$, we define the signed decomposition in base B as

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

where each term belongs to $[-B/2, B/2]$. It is easy to see that $\left\langle \mathfrak{g}^{-1}(a), \mathfrak{g} \right\rangle \equiv a \mod Q$.

2.4 Hard Problems and Message Encoding

We recall the learning with errors (LWE) [Reg09], Ring learning with errors (RLWE) [LPR13], and NTRU [HPS98] problems, which are instantiated in FHE schemes.

Definition 1. (Decisional LWE Problem [Reg09]). For positive integers q and n, and a noise distribution χ over $\mathbb{Z}$, the $Decision - LWE_{q,n,\chi}$ problem is to distinguish the distribution between $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + \ e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ and a pair uniformly chosen at random from $\mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ is chosen uniformly at random, $e \leftarrow \chi_\delta$ is chosen from the Gaussian distribution, and $\mathbf{s}$ is the secret key that is chosen from $\mathbb{Z}^n$.

Definition 2. (Decisional RLWE Problem [LPR13]). For positive integers Q and N, and a noise distribution χ over $\mathcal{R}$, the $Decision - RLWE_{Q,N,\chi}$ problem is to distinguish the distribution between $(a, as + e) \in \mathcal{R}_Q \times \mathcal{R}_Q$ and a pair uniformly chosen at random from $\mathcal{R}_Q \times \mathcal{R}_Q$, where a is uniformly random in $\mathcal{R}_Q$, the error $e \leftarrow \chi_\delta^N$, and s is the polynomial secret key that is chosen from $\mathcal{R}$.

Definition 3. (Decisional NTRU Problem [HPS98]). For positive integers Q and N, and a noise distribution χ over $\mathcal{R}$, the $Decision - NTRU_{Q,N,\chi}$ problem is to distinguish the distribution between $g/f \in \mathcal{R}_Q$ and a uniform chosen at random from $\mathcal{R}_Q$, where $g \leftarrow \chi_\delta^N$ is the noise polynomial, and the secret key f is invertible in $\mathcal{R}_Q$.

In the homomorphic encryption scheme, there are two forms of encoding messages, known as least significant bit (LSB) encoding, and most significant bit (MSB) encoding. In this paper, we use MSB encoding by default, and its encoding and decoding functions are described as follows

$$\mathsf{Encode}: \varphi = \left\lfloor \frac{q}{t} \right\rfloor \cdot m + e, \quad \mathsf{Decode}: \left\lfloor \frac{t}{q} \cdot \varphi \right\rceil \bmod t,$$

where m is the message, t is the plaintext modulus, and q is the ciphertext modulus. Note that we sometimes omit the encoding forms in ciphertexts for simplicity.

2.5 NTRU Ciphertext and Homomorphic Operation

In this subsection, we recall the NTRU encryption, and some homomorphic building blocks such as external products, key-switching and ring automorphisms.

2.5.1 NTRU Encryption

Definition 4. The NTRU ciphertext can be defined as

$$\mathrm{NTRU}_{f,Q}(\mu) = (g + \mu)/f \in \mathcal{R}_Q,$$

where the error polynomial g is taken from a Subgaussian Distribution, the secret key f is from $\{-1, 0, 1\}^N$ with the variance $\mathsf{Var}(f) = 1/\sqrt{2}$.

The structure of NTRU ciphertext allows for homomorphic addition and scalar multiplication operations. Given the NTRU samples ct_1 and ct_2 with the same secret key f, their terms can be added together to obtain:

$$\mathsf{ct}_1 + \mathsf{ct}_2 = (g_1 + g_2 + (\mu_1 + \mu_2))/f \in \mathcal{R}_Q.$$

Moreover, the multiplication between a ciphertext and a scalar z can be obtained directly from multiplying polynomials.

2.5.2 NTRU-based External Product

Bonte et al. [BIP+22] proposed a GSW-like NTRU ciphertext encryption, called NGS, in definition 5.

Definition 5. Given the gadget decomposition vector $\mathfrak{g} = (1, B, \cdots, B^{d-1})$, the NGS ciphertext is defined as

$$\mathrm{NGS}_{f,Q}(m) = \mathbf{g}/f + \mathfrak{g} \cdot m \in \mathcal{R}_Q^d,$$

where $d = \lceil \log_B Q \rceil$, $\mathbf{g} = (g_0, \ldots, g_{d-1})$, and $g_0, ..., g_{d-1}$ are the error polynomials.

Lemma 1 (**External Product**)**.** *Input an NTRU ciphertext* $\mathsf{ct} = (g+\mu)/f \in \mathrm{NTRU}_{f,Q}(\mu)$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}))$*, and an NGS ciphertext* $\mathbf{CT} \in \mathrm{NGS}_{f,Q}(m)$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathbf{CT}))$*, the external product* $\odot$ *outputs a new NTRU ciphertext* $\mathsf{ct}' \in \mathrm{NTRU}_{f',Q}(\mu m)$*, and its variance satisfies* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \leq Nd\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{CT})) + \mathsf{Var}(\mathsf{err}(\mathsf{ct}))$*.*

Proof.

$$\begin{aligned} \mathsf{ct}' &= \mathsf{ct} \odot \mathbf{CT} \\ &= \langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{CT} \rangle \\ &= (\langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{g} \rangle)/f + \mathsf{ct} \cdot m \\ &= (\langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{g} \rangle + g \cdot m + \mu m)/f \in \mathcal{R}_Q, \end{aligned}$$

where the noise term is $g' = \langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{g} \rangle + g \cdot m$. Let gadget decomposition be $\mathfrak{g}^{-1}(\mathsf{ct}) = (c_0, ..., c_{d-1})$, where each term c_i is viewed as uniformly distributed over $[-B/2, B/2]$. Thus, the variance of the noise is $\mathsf{Var}(g') \leq Nd\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{CT})) + \mathsf{Var}(\mathsf{err}(\mathsf{ct}))$.

$\square$

2.5.3 NTRU-based Key-switching and Automorphism

The key-switching technique can change secret keys in homomorphic encryption schemes. Lemma 2 shows the key-switching algorithm of NTRU ciphertext.

Lemma 2 (**NTRU Key Switching**)**.** *Input an NTRU ciphertext* $\mathsf{ct} = (g + \mu)/f' \in \mathrm{NTRU}_{f,Q}(\mu)$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}))$*, and the switching key* $\mathbf{KSK} \in \mathrm{NGS}_{f,Q}(f'/f)$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathbf{KSK}))$*, the NTRU key-switching algorithm computes the external product*

$$\mathsf{NTRU.KeySwitch}(\mathsf{ct}) = \mathsf{ct} \odot \mathbf{KSK},$$

which outputs a new NTRU ciphertext $\mathsf{ct}' \in \mathrm{NTRU}_{f,Q}(\mu)$*, and its variance satisfies* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \leq Nd\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{KSK})) + \mathsf{Var}(\mathsf{err}(\mathsf{ct}))$*.*

The proof of the lemma and the noise analysis can be referred to Appendix B.

For the polynomial ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$, where N is a power of two, there are N automorphisms as

$$\psi_j : \mathcal{R} \to \mathcal{R},\ \mathrm{c}(X) \to \mathrm{c}(X^j)$$

for $j \in \mathbb{Z}_{2N}^*$. The automorphism operation can be applied to RLWE and NTRU ciphertexts. Formally, given an NTRU sample $\mathsf{ct} = (g + \mu)/f \in \mathcal{R}_Q$, and a switching key $\mathbf{KSK}_j \in \mathrm{NGS}_{f(X),Q}(f(X)/f(X^j))$, we define the automorphism $\mathsf{HomAuto}_j(\mathsf{ct}, \mathbf{KSK}_j)$ based on NTRU ciphertexts as follows:

- Let $\psi_j(\mathsf{ct}(X)) = (g(X^j) + \mu(X^j))/f(X^j) \in \mathcal{R}_Q$.
- Apply the NTRU key switching from the secret key $f(X^j)$ to $f(X)$.

The first step outputs an NTRU encryption of $m(X^j)$ under the secret key $f(X^j)$. In the second step, NTRU key-switching is used to switch the secret key from $f(X^j)$ to $f(X)$. Note that the automorphism ψ_j is a permutation on the coefficients of the elements of $\mathcal{R}$ and does not introduce an additional error, since the automorphism is a canonical preserving mapping. Furthermore, compared to key switching, the time required for the permutation is negligible.

Finally, we introduce the associated operations of LWE ciphertexts in Appendix A, including sample extraction, key switching, and modulus switching, which are necessary modules for the construction of the bootstrapping algorithm.

2.6 Number Theoretic Transform (NTT)

The Number Theoretic Transform (NTT) is a mathematical algorithm utilized in the domains of number theory. It can convert a polynomial from its coefficient representation to the NTT representation, facilitating efficient polynomial multiplication and convolution operations. Typically, NTT is a special case of the Fast Fourier Transform on finite fields [Pol71]. In lattice-based cryptography, such as RLWE and NTRU, the NTT plays a crucial role as a fundamental computational tool, reducing the computation complexity of polynomial multiplication from $O(N^2)$ to $O(N \log N)$.

In details, for the $2N$-th cyclotomic ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N + 1)$, where Q is a prime number satisfying $Q \equiv 1 \pmod{2N}$, there exists a $2N$-th primitive root of unity $\zeta \in \mathbb{Z}_Q$ that satisfies $X^N + 1 = X^N - \zeta^N \pmod{Q}$. By using the Chinese Remainder Theorem (CRT), the polynomial $X^N + 1$ can be split into N polynomials

$$\mathbb{Z}_Q[X]/(X^N+1) \to \mathbb{Z}_Q[X]/(X-\zeta) \times \mathbb{Z}_Q[X]/(X-\zeta^3) \times \cdots \times \mathbb{Z}_Q[X]/(X-\zeta^{2N-1}).$$

Thus, for the polynomial $a(X) \in \mathcal{R}_Q$, one can get a new vector as,

$$(a(X) \bmod (X-\zeta), a(X) \bmod (X-\zeta^3), \cdots, a(X) \bmod (X-\zeta^{2N-1})).$$

Then, we define the NTT representation of the polynomial $a(X)$ as

$$\mathrm{NTT}(a) = \left(a(\zeta^1), a(\zeta^3), \cdots, a(\zeta^{2N-1})\right) \in \mathbb{Z}_Q^N.$$

On the other hand, the INTT step computes the inverse isomorphism by using the negative powers of primitive roots, and we can define the INTT operation as

$$\mathrm{INTT}(a) = \frac{1}{N}\left(a(\zeta^{-1}), a(\zeta^{-3}), \cdots, a(\zeta^{-(2N-1)})\right) \in \mathbb{Z}_Q^N.$$

In this way, given two polynomials $a(x)$ and $b(x)$, we can efficiently compute $\mathrm{INTT}(\mathrm{NTT}(a) \cdot \mathrm{NTT}(b))$, and the computation complexity is $O(N \cdot \log_2 N)$. The detailed NTT algorithm is described in Appendix C.

3 Improved CMux-based Bootstrapping

The FHEW/TFHE-like bootstrapping includes functional bootstrapping (FBS), multi-valued bootstrapping (MVBS), circuit bootstrapping (CBS), etc. Here, the functional bootstrapping can be used to evaluate a NAND gate [DM15, DvW21], which is known as gate bootstrapping (GBS). Input two LWE ciphertexts, the NAND gate bootstrapping first adds them together and performs the blind rotation and extraction procedures. After that, one can utilize the modulus switchings and key switching to obtain the refreshed NAND result, as shown in Figure 1.

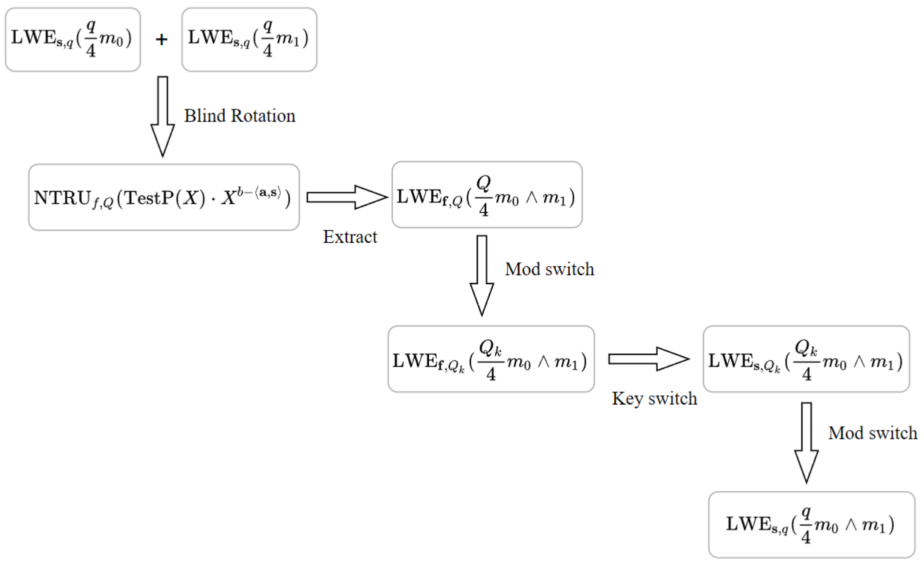

$\mathrm{LWE}_{\mathbf{s},q}(\frac{q}{4}m_0)$ + $\mathrm{LWE}_{\mathbf{s},q}(\frac{q}{4}m_1)$
Blind Rotation
$\mathrm{NTRU}_{f,Q}(\mathrm{TestP}(X)\cdot X^{b-\langle \mathbf{a},\mathbf{s}\rangle})$
Extract
$\mathrm{LWE}_{\mathbf{f},Q}(\frac{Q}{4}m_0 \wedge m_1)$
Mod switch
$\mathrm{LWE}_{\mathbf{f},Q_k}(\frac{Q_k}{4}m_0 \wedge m_1)$
Key switch
$\mathrm{LWE}_{\mathbf{s},Q_k}(\frac{Q_k}{4}m_0 \wedge m_1)$
Mod switch
$\mathrm{LWE}_{\mathbf{s},q}(\frac{q}{4}m_0 \wedge m_1)$

Figure 1: Bootstrapping for evaluating NAND gate with LWE and NTRU ciphertexts

In this section, we introduce some optimization techniques to improve the CMux-based blind rotation with NTRU accumulator, including NTRU-based approximate gadget decomposition and key unrolling techniques. In addition, we also present a comparative analysis with existing methods to demonstrate the advantages of the proposed scheme.

3.1 Approximate Gadget Decomposition

The approximate gadget decomposition was originally introduced for RLWE ciphertext in the TFHE scheme [CGGI20]. Compared to exact decomposition, approximate decomposition can improve the efficiency of the external product in blind rotation. Note that existing NTRU-based bootstrapping schemes [BIP+22, Klu22, XZD+23] employ exact decomposition to reduce noise growth, and do not explore the integration of the approximate decomposition technique with the NTRU accumulator. We develop the approximate gadget decomposition for NTRU ciphertext to improve the efficiency of blind rotation. Notably, the approximate gadget decomposition and approximate external product for NTRU ciphertext are formally given in Definition 6 and Lemma 3.

Definition 6. Given the decomposition base B and a auxiliary modulus P, the approximate gadget decomposition for a polynomial c is defined by $\mathfrak{g}_A^{-1}(\mathrm{c}) = (\mathrm{c}_0, ..., \mathrm{c}_{d'-1})$, such that $\mathrm{c} = \sum_{i=0}^{d'} \mathrm{c}_i \cdot PB^j$, where $c_i \in [-B/2, B/2]$. Thus, for the approximate gadget vector $\mathfrak{g}' = (P, P\cdot B, \cdots, P\cdot B^{d'-1})$, we have $\langle \mathfrak{g}_A^{-1}(\mathrm{c}), \mathfrak{g}' \rangle = c + \epsilon$, where $\epsilon \le P/2$. Then, for a ternary message $m \in \pm X^k$, we define the new NGS ciphertext as

$$\mathrm{NGS}'_{f,Q}(m) = (g_0/f + P \cdot m, \dots, g_{d'-1}/f + P \cdot B^{d'-1} m) \in \mathcal{R}_Q^{d'}.$$

Lemma 3 (**Approximate External Product**). *Input an NTRU ciphertext* $\mathsf{ct} = (g + \mu)/f$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}))$*, and an* NGS' *ciphertext* $\mathbf{CT} = \mathrm{NGS}'_{f,Q}(m)$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathbf{CT}))$*, the approximate external product* $\odot_A$ *outputs a new NTRU ciphertext , and its variance satisfies* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \le Nd\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{CT})) + \frac{NP^2}{24} + \mathsf{Var}(\mathsf{err}(\mathsf{ct}))$.

Proof. Let approximate external product be

$$\begin{aligned} \mathsf{ct} \odot_A \mathbf{CT} &= \langle \mathfrak{g}_A^{-1}(\mathsf{ct}), \mathbf{CT} \rangle \\ &= (\langle \mathfrak{g}_A^{-1}(\mathsf{ct}), \mathbf{g} \rangle)/f + (\mathsf{ct} + \epsilon) \cdot m \\ &= (\langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{g} \rangle + g \cdot m + \mu m)/f + \epsilon m, \\ &= (\langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{g} \rangle + g \cdot m + \epsilon m f + \mu m)/f, \end{aligned} \tag{1}$$

where the error term is $g' = \langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{g} \rangle + g \cdot m + \epsilon m f$. Since $\mathsf{Var}(f) = 1/2$ and $\|m\|_2^2 \le 1$, the variance of the noise satisfies

$$\mathsf{Var}(g') \le Nd\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{CT})) + \frac{NP^2}{24} + \mathsf{Var}(\mathsf{err}(\mathsf{ct})). \tag{2}$$

□

Remark . *Note that we do not use the approximate gadget decomposition technique for NTRU ciphertext in key-switching procedure, as it would lead to significant noise growth. Specifically, the key-switching key discussed in Section 2.5 encrypts the polynomial* $\frac{f}{f'}$ *instead of the ternary message* $\pm X^k$*. If approximate decomposition were utilized, the noise term for* $\epsilon m f$ *in Equation 2 would expand by a factor of* N*, i.e.,* $\frac{N^2P^2}{24}$ *compared with the original external product.*

3.2 Improved CMux-based Bootstrapping with Binary Secret Key

In this subsection, we present the improved CMux-based blind rotation algorithm with NTRU accumulator. We recall that the blind rotation procedure can homomorphically evaluate the encryption of $X^{\frac{2N}{q}(b + \sum_{i=0}^{n-1} a_i s_i)}$, where q is the LWE ciphertext modulus. For the binary secret key distribution, [BMMP18] introduces the unrolling factor $m = 2$. Let $Y = X^{\frac{2N}{q}}$ that has an order of exactly q, the accumulation process can be rewritten as:

$$Y^{\sum_{i=0}^{n-1} a_i s_i} = Y^{\sum_{i=0}^{(n-1)/2} a_{2i}s_{2i} + a_{2i+1}s_{2i+1}}.$$

Furthermore, since the secret key satisfies $s_i \in \{0, 1\}$, we have the following facts

$$\begin{aligned} &Y^{a_{2i}s_{2i} + a_{2i+1}s_{2i+1}} \\ =&Y^{a_{2i}+a_{2i+1}} \cdot s_{2i}s_{2i+1} + Y^{a_{2i}} \cdot s_{2i}(1 - s_{2i+1}) + Y^{a_{2i+1}} \cdot s_{2i+1}(1 - s_{2i}) + (1 - s_{2i})(1 - s_{2i+1}) \\ =&(Y^{a_{2i}+a_{2i+1}} - 1) \cdot s_{2i}s_{2i+1} + (Y^{a_{2i}} - 1) \cdot s_{2i}(1 - s_{2i+1}) + (Y^{a_{2i+1}} - 1) \cdot s_{2i+1}(1 - s_{2i}) + 1. \end{aligned}$$

Then, we generate the bootstrapping key **BRK** as follows.

$$\begin{cases} \mathbf{BRK}_{i,0} = \mathrm{NGS}'(1),\ \mathbf{BRK}_{i,1} = \mathrm{NGS}'(0),\ \mathbf{BRK}_{i,2} = \mathrm{NGS}'(0), \text{ if } (s_{2i} = 1, s_{2i+1} = 1); \\ \mathbf{BRK}_{i,0} = \mathrm{NGS}'(0),\ \mathbf{BRK}_{i,1} = \mathrm{NGS}'(1),\ \mathbf{BRK}_{i,2} = \mathrm{NGS}'(0), \text{ if } (s_{2i} = 1, s_{2i+1} = 0); \\ \mathbf{BRK}_{i,0} = \mathrm{NGS}'(0),\ \mathbf{BRK}_{i,1} = \mathrm{NGS}'(0),\ \mathbf{BRK}_{i,2} = \mathrm{NGS}'(1), \text{ if } (s_{2i} = 0, s_{2i+1} = 1); \\ \mathbf{BRK}_{i,0} = \mathrm{NGS}'(0),\ \mathbf{BRK}_{i,1} = \mathrm{NGS}'(0),\ \mathbf{BRK}_{i,2} = \mathrm{NGS}'(0), \text{ if } (s_{2i} = 0, s_{2i+1} = 0), \end{cases}$$

for $i \in [0, n/2]$. After that, the CMux gate with key unrolling can be expressed as

$$\begin{aligned}\mathsf{acc} \leftarrow & (Y^{a_{2i}+a_{2i+1}} - 1) \cdot \mathbf{BSK}_{i,0} \odot_A \mathsf{acc} + (Y^{a_{2i}} - 1) \cdot \mathbf{BSK}_{i,1} \odot_A \mathsf{acc} \\ & + (Y^{a_{2i+1}} - 1) \cdot \mathbf{BSK}_{i,2} \odot_A \mathsf{acc} + \mathsf{acc}.\end{aligned} \tag{3}$$

For the details of implementation, we first use the approximate decomposition for the input acc, and then apply d' NTT transformations to these polynomials. To minimize the number of INTT operations in the blind rotation, we precompute a table that contains all the NTT transformations of $Y^i - 1$, where $0 \leq i \leq q - 1$. Then, we utilize the LWE ciphertext to retrieve the corresponding NTT representations for $Y^{a_{2i}+a_{2i+1}} - 1$, $Y^{a_{2i}} - 1$, and $Y^{a_{2i+1}} - 1$, which can be used for Hadamard multiplication with the bootstrapping keys in the NTT domain. Afterward, only an INTT transformation is performed on the accumulated acc. Algorithm 1 presents the detailed bootstrapping process.

Algorithm 1 Improved CMux-based Bootstrapping with NTRU and LWE Ciphertexts

```
Input:
    An LWE sample $\mathsf{ct} = (\mathbf{a}, b = -\langle \mathbf{a}, \mathbf{s} \rangle - \lfloor \frac{q}{t} \rfloor \cdot m + e) \in \mathrm{LWE}^n_{\mathbf{s},q}(m)$.
    The bootstrapping key $\mathbf{BRK}$.
    A special bootstrapping key $\mathbf{BRK}' = \mathrm{NGS}'_{f,Q}(1/f)$.
    An LWE key switching key $\mathsf{ksk}_{\mathbf{s}}(\phi(f))$ as shown in Section A.
Output:
    An LWE sample $\mathsf{ct}' \in \mathrm{LWE}^n_{\mathbf{s},q}(f(m))$.
 1: Set $\mathrm{TestP}(X) = \sum_{i=0}^{N-1} \frac{Q}{t} \cdot f(\lfloor \frac{t}{q} \cdot i \rceil) \cdot X^i$, and $Y = X^{\frac{2N}{q}}$.
 2: Let $\mathsf{acc} = (\mathrm{TestP}(X) \cdot Y^b) \odot_A \mathbf{BSK}'$
 3: for $(i = 0; i < (n-1)/2; i = i + 1)$ do
 4:     $\mathsf{acc} = (Y^{a_{2i}+a_{2i+1}} - 1) \cdot \mathbf{BSK}_{i,0} \odot_A \mathsf{acc} + (Y^{a_{2i}} - 1) \cdot \mathbf{BSK}_{i,1} \odot_A \mathsf{acc}$
            $+ (Y^{a_{2i+1}} - 1) \cdot \mathbf{BSK}_{i,2} \odot_A \mathsf{acc} + \mathsf{acc}$
 5: end for
 6: $\mathsf{ct}' = \mathsf{SampleExtract}(\mathsf{acc})$
 7: $\mathsf{ct}' = \mathsf{ModSwitch}_{Q \to Q_k}(\mathsf{ct}')$
 8: $\mathsf{ct}' = \mathsf{LWE.KeySwitch}(\mathsf{ct}')$
 9: $\mathsf{ct}' = \mathsf{ModSwitch}_{Q_k \to q}(\mathsf{ct}')$
10: return $\mathsf{ct}'$
```

Theorem 1. *Input an LWE ciphertext* ct *with error variance* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}))$, *Algorithm 1 outputs a refreshed LWE ciphertexts as* $\mathsf{ct}' \in \mathsf{LWE}^n_{\mathbf{s},q}(f(m))$, *and its variance satisfies* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \leq \frac{q^2}{Q_k^2} \cdot [\frac{Q_k^2}{Q^2} \cdot \mathsf{Var}(\mathsf{err}(\mathsf{BR})) + \frac{2+N}{24} + N d_k \cdot \mathsf{Var}(\mathsf{err}(\mathsf{ksk}))] + \frac{2+n}{24}$.

Proof. Let's first focus on the correctness of the algorithm. Let the special bootstrapping key be $\mathbf{BSK}' = \frac{\mathbf{g}+\mathfrak{g}'}{f}$, where $\mathbf{g}$ is the noise term from Gaussian distribution, we can compute the initial acc as

$$\begin{aligned}\mathsf{acc} &= (\mathrm{TestP}(X) \cdot Y^b) \odot_A \mathbf{BSK}' \\ &= \left\langle \mathfrak{g}_A^{-1}(\mathrm{TestP}(X) \cdot Y^b), \frac{\mathbf{g} + \mathfrak{g}'}{f} \right\rangle \\ &= \frac{\langle \mathfrak{g}_A^{-1}(\mathrm{TestP}(X) \cdot Y^b), \mathbf{g} \rangle + \epsilon f + \mathrm{TestP}(X) \cdot Y^b}{f},\end{aligned}$$

which can be regarded as a ciphertext $\mathrm{NTRU}_{f,Q}(\mathrm{TestP}(X) \cdot Y^b)$. After that, line 4 performs the binary CMux gate with key unrolling operation. It is easy to see that this step yields the ciphertext $\mathsf{acc} = \mathrm{NTRU}_{f,Q}(Y^{a_{2i}s_{2i}+a_{2i+1}s_{2i+1}})$ as shown in Equation 3.2. Further, this process is iterated $n/2$ times, which results in a ciphertext as

$$\begin{aligned}\mathsf{acc} &= \mathrm{NTRU}_{f,Q}\left(\mathrm{TestP}(X)\cdot Y^{b+\sum_{i=0}^{n-1}a_is_i}\right)\\ &= \mathrm{NTRU}_{f,Q}\left(\mathrm{TestP}(X)\cdot Y^{-(\lfloor\frac{q}{t}\rfloor\cdot m+e)}\right).\end{aligned}$$

Note that the test polynomial $\mathrm{TestP}(X) = \sum_{i=0}^{N-1}\frac{Q}{t}\cdot f(\lfloor\frac{t}{q}\cdot i\rceil)\cdot X^i$ can refresh the noise of the LWE ciphertext while evaluating a lookup table $f:\mathbb{Z}_t\to\mathbb{Z}_t$. Thus, the ciphertext $\mathrm{LWE}^N_{f,Q}(f(m))$ is obtained by the NTRU-based sample extraction under the secret key f in line 6 of the algorithm. Then, the LWE key switching is performed to switch the dimension from N to n, and the modulus switching can convert the modulus to the original modulus q. Finally, the algorithm outputs the ciphertext $\mathsf{ct}' = \mathrm{LWE}^n_{\mathbf{s},q}(f(m)) \in \mathbb{Z}_q^{n+1}$.

Noise analysis. Now, let's analyze the noise growth in the bootstrapping process. Firstly, after performing the external product with the key $\mathbf{BRK}'$, the noise term is $\langle \mathfrak{g}_A^{-1}(\mathrm{TestP}(X)\cdot Y^b), \mathbf{g}\rangle + \epsilon f$, and the variance of noise for the initial accumulator acc is

$$\mathsf{Var}(\mathsf{acc}) \le Nd'\frac{B^2}{12}\cdot\mathsf{Var}(\mathsf{err}(\mathbf{BRK}')) + \frac{NP^2}{24}.$$

Due to the fact that $||X^i-1||_2^2\cdot\mathsf{Var}(\mathsf{err}(\mathbf{BRK})) \le 2\cdot\mathsf{Var}(\mathsf{err}(\mathbf{BRK}))$, the variance of noise for the binary CMux gate with key unrolling is

$$\mathsf{Var}(\mathsf{err}(\mathsf{acc})) \le \frac{Nd'B^2}{2}\cdot\mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{NP^2}{8} + \mathsf{Var}(\mathsf{err}(\mathsf{acc})).$$

Note that this step is performed $n/2$ times in blind rotation, thus the variance of noise in blind rotation satisfies

$$\begin{aligned}\mathsf{Var}(\mathsf{err}(\mathsf{BR})) &\le \frac{nNd'B^2}{4}\cdot\mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{nNP^2}{16} + \frac{Nd'B^2}{12}\cdot\mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{NP^2}{24}\\ &\le \frac{(3n+1)Nd'B^2}{12}\cdot\mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{(3n+2)NP^2}{48}.\end{aligned}$$

After that, we need to perform the modulus switching from Q to Q_k, and the variance of the error is

$$\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \le \frac{Q_k^2}{Q^2}\cdot\mathsf{Var}(\mathsf{err}(\mathsf{BR})) + \frac{2+N}{24}.$$

In addition, after the key-switching for LWE ciphertext, we have

$$\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \le Nd_k\cdot\mathsf{Var}(\mathsf{err}(\mathsf{ksk})) + \mathsf{Var}(\mathsf{err}(\mathsf{ct}')).$$

Finally, by performing modulus switching form Q_k to q, we can conclude that the variance of the error generated by the bootstrapping process is

$$\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \le \frac{q^2}{Q_k^2}\cdot[\frac{Q_k^2}{Q^2}\cdot\mathsf{Var}(\mathsf{err}(\mathsf{BR})) + \frac{2+N}{24} + Nd_k\cdot\mathsf{Var}(\mathsf{err}(\mathsf{ksk}))] + \frac{2+n}{24}. \quad (4)$$

□

Finally, we present the comparisons of computational cost among different schemes based on the CMux method in Table 3. It is easy to see that the proposed scheme involves the least number of NTTs compared to other schemes under the binary secret key distribution. In addition, our techniques can be extended to the ternary secret key distributions, as demonstrated in [JP22b]. We omit the details and comparisons related to ternary distribution as they are similar to those presented in Table 3.

Table 2: Comparison of noise variance of CMux-based blind rotation for different schemes. Here, KU is the key unrolling method, σ^2_{BR} is the variance of blind rotation key, and $d' < d$.

Schemes	Assumption	Method	Noise variance
[CGGI16]	RLWE	CMux	$\frac{nNd'B^2}{3} \cdot \sigma^2_{\mathsf{BR}} + \frac{nNP^2}{12}$
[BMMP18]	RLWE	CMux & KU	$\frac{nNd'B^2}{2} \cdot \sigma^2_{\mathsf{BR}} + \frac{nNP^2}{8}$
[MP21]	RLWE	CMux	$\frac{nNdB^2}{3} \cdot \sigma^2_{\mathsf{BR}}$
[BIP+22]	NTRU	CMux	$\frac{nNd'B^2}{6} \cdot \sigma^2_{\mathsf{BR}}$
Algorithm 1	NTRU	CMux & KU	$\frac{(3n+1)NdB^2}{12} \cdot \sigma^2_{\mathsf{BR}} + \frac{(3n+2)NP^2}{48}$

Table 3: Comparison of computational cost of CMux-based blind rotation for different schemes, where HMs is the Hadamard multiplications, and $d' < d$.

Schemes	# NTTs/FFTs	# HMs
[CGGI16]	$2n(d'+1)$	$2n(2d'+1)$
[BMMP18]	$n(d'+1)$	$3n(2d'+1)$
[MP21]	$2n(d+1)$	$2n(2d+1)$
[BIP+22]	$n(d+1)$	$n(d+1)$
Algorithm 1	$n/2(d'+1)+d'$	$(3/2n+1)d' + \frac{3}{2}n$

4 Improved Automorphism-based Bootstrapping

In this section, we focus on the automorphism-based blind rotation and improve the efficiency of the algorithm. Currently, there are two blind rotation algorithms based on automorphisms technique. One is the LMK scheme [LMK+23], which is based on the RLWE assumption, and the other is the XZD scheme [XZD+23], which is based on the NTRU assumption. We introduce two improvements for the external product and the automorphism evaluation, respectively, as summarized as follows:

1. **External Product**: In automorphism-based bootstrapping algorithm, we use the NTRU-based accumulator, while utilizing the approximate gadget decomposition to accelerate the external product. For more details, please refer to Section 3.1.

2. **Automorphism**: We optimize the window size technique proposed by [LMK+23]. By using w pre-stored automorphism keys, we can reduce the number of NTRU automorphisms from n, as mentioned in [XZD+23], to $\frac{\omega-1}{\omega}\kappa + \frac{N}{\omega}$, where $\kappa = N(1 - e^{-n/N})$ in the average case. Furthermore, we can merge the symmetric sets by introducing auxiliary bootstrapping keys, which can further reduce the number of NTRU automorphisms to $\frac{\omega-1}{\omega}\kappa + \frac{N}{2\omega}$. We detail these two techniques in this Section.

4.1 Optimization for Automorphisms

Firstly, let us recall the idea of the automorphism-based blind rotation algorithm [LMK+23]. Given an LWE ciphertext with all-odd terms with the modulus $q = 2N$, each term can map to $\pm g^k$ by applying the isomorphism $\mathbb{Z}^*_{2N} \cong \mathbb{Z}_{N/2} \otimes \mathbb{Z}_2$. In this setting, we can define N set $I^+_l = \{j : a_j = g^l\}$ and $I^-_l = \{k : a_k = -g^l\}$, for $l \in [0, N/2-1]$. Then, the blind rotation can be expressed as through the decomposition

$$\sum_i a_i s_i = \left(\sum_{j\in I_0^+} s_j + \cdots + g\left(\sum_{j\in I_{N/2-1}^+} s_j - g\left(\sum_{k\in I_0^-} s_k + \cdots + g\left(\sum_{k\in I_{N/2-1}^-} s_k\right)\right)\right)\right) (\bmod 2N). \tag{5}$$

In this equation, one need to perform N automorphisms operations. Here, we introduce two approaches to reduce the number of automorphisms based on NTRU ciphertext.

Approach 1. Merge the empty sets for NTRU ciphertext. We note that the scheme [XZD$^+$23] does not utilize the set partitioning strategy and window size techniques mentioned above. Instead, the scheme iterates over each $a_i s_i$ individually as shown in Algorithm 4 of Appendix D. Our approach 1 is to directly substitute the RLWE accumulator in [LMK$^+$23] with the NTRU accumulator in blind rotation and use the window size technique to reduce the number of automorphism.

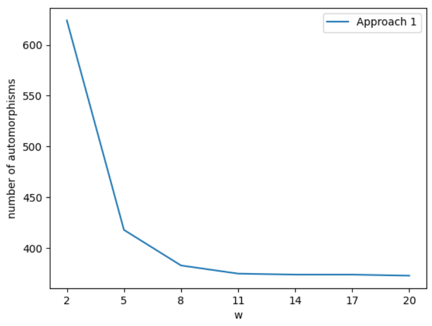
In detail, when an empty set I_l exists, only the automorphism $\mathsf{HomAuto}_g$ needs to be performed without the external products for s_j. Thus, we can merge it into the neighboring set and evaluate a single automorphism $\mathsf{HomAuto}_{g^2}$ for NTRU-based accumulator. Similarly, the window size technique was introduced to handle the case of consecutive empty sets. It actually provides a trade-off between reducing the number of automorphism computations by adding additional storage for $\mathbf{KSK}_g, ..., \mathbf{KSK}_{g^\omega}$. Thus, given the window size ω, our NTRU-based method need to perform $\frac{\omega-1}{\omega}\kappa + \frac{N}{\omega}$ automorphisms compared to n in [XZD$^+$23], where $n = N/2$. Please see Section 4.1 of [LMK$^+$23] for more details.

Approach 2. Merge the symmetric sets. We note that for a fixed l, the two sets $I_l^+ = \{j : a_j = g^l\}$ and $I_l^- = \{k : a_k = -g^l\}$ are symmetric. We can merge these two sets and perform single automorphism operation by introducing the auxiliary bootstrapping key $\mathbf{BSK}^- = \mathrm{NGS}'_{f,Q}(X^{-s_i})_{i\in[0,n-1]}$. Specifically, we evaluate $X^{\sum_{j\in I_l^+} s_j - \sum_{k\in I_l^-} s_k}$ using $\mathrm{NGS}'_{f,Q}(X^{s_j})_{j\in I_l^+}$ and $\mathrm{NGS}'_{f,Q}(X^{-s_k})_{k\in I_l^-}$ when performing external products, and followed by an automorphism $\mathsf{HomAuto}_g$ operation. Thus, the new blind rotation process can be expressed as:

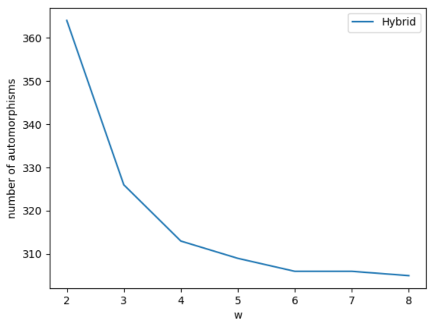
$$\sum_i a_i s_i = \left(\sum_{j\in I_0^+} s_j - \sum_{k\in I_0^-} s_k + g\left(\sum_{j\in I_1^+} s_j - \sum_{k\in I_1^-} s_k + \cdots + g\left(\sum_{j\in I_{N/2-1}^+} s_j - \sum_{k\in I_{N/2-1}^-} s_k\right)\right)\right) (\bmod 2N), \tag{6}$$

where the number of automorphism can be reduced from N for Equation 5 to $\frac{N}{2}$. In addition, we can also combine the window size technique to build the hybrid approach that further reduce the number of automorphism to $\frac{\omega-1}{\omega}\kappa + \frac{N}{2\omega}$. Then, we can determine the specific number of automorphism based on the chosen value of ω.

Monte Carlo Simulation. To gain a more intuitive understanding of the effect of window size on the number of automorphisms, we employ the Monte Carlo method [Jam80] to determine the optimal window size. Specifically, we consider the gate bootstrapping parameters $N = 1024$ and $n = 465$, as described in Section 5. The simulation can be simplified as a stochastic process, where we randomly place n balls into N (Approach 1) and $N/2$ (Hybrid Approach) bins. Since the LWE ciphertext a_i are uniformly distributed, each bin has an equal probability of receiving a ball. The simulation results, as shown in Figure 2, demonstrate that the number of automorphisms n_{aut} monotonically decreases concerning the window size, but there exists an asymptotic lower bound. Given that enlarging the window size only slightly increases the key size, we choose a sufficiently large



(a) Approach 1



(b) Hybrid Approach

Figure 2: The window size and corresponding number of automorphisms for approach 1 and hybrid approach.

ω to minimize the number of automorphisms. The test data indicates that Approach 1 allows us to reduce n_{aut} from 465 to 373 on average case, by using a window size of $\omega = 20$. In the case of the hybrid approach, we can further reduce n_{aut} to 305 with a window size of $\omega = 8$.

To summarize, our hybrid approach yields a 34% reduction in the number of automorphisms compared to the original method proposed in [XZD+23]. Compared to [LMK+23], which utilizes a window size of $\omega = 10$, our hybrid approach achieves an 18% reduction in automorphisms. The complete algorithm for the hybrid approach can be found in Section 4.2.

4.2 The Construction

In this subsection, we present the proposed hybrid approach as follows:

- $\mathsf{BRKGen}(\mathbf{s}, f)$**.** Given an LWE secret key $\mathbf{s} = (s_0, ..., s_{n-1}) \in \chi^n$, and an NTRU secret key $f \in \mathcal{R}_Q$. For all $i \in [0, n-1]$, the blind rotation keys are generated as follows:

$$\mathbf{BRK}_i^+ = \mathrm{NGS}'_{f,Q}(X^{s_i}),\ \mathbf{BRK}_i^- = \mathrm{NGS}'_{f,Q}(X^{-s_i}), \mathbf{BRK}' = \mathrm{NGS}'_{f,Q}(1/f).$$

  Then, it generates a set of key-switching keys for automorphism as follows:

$$\mathbf{KSK}_{g^v} = \mathrm{NGS}_{f,Q}\left(\frac{f(X^{g^v})}{f(X)}\right), v \in [1, \omega]$$

  The algorithm outputs $\mathbf{EVK} = (\mathbf{BRK}_i^+, \mathbf{BRK}_i^-, \mathbf{BRK}', \mathbf{KSK}_{g^v})$ as the evaluation key for blind rotation.

- $\mathsf{BootStrap}(\mathsf{ct}, \mathbf{EVK})$**.** Takes as input an LWE ciphertext $\mathsf{ct} = (\mathbf{a}, b)$, and the evaluation key $\mathbf{EVK}$, Algorithm 2 outputs a refreshed LWE ciphertext ct'.

Correctness. Firstly, we need to perform the modulus switching operation that yields an LWE ciphertext $(\mathbf{a}', b')$ with all odd terms as shown in Appendix A.3. Then, we can obtain an NTRU ciphertext that

$$\mathrm{NTRU}_{f,Q}(\mathrm{TestP}(X^{-g}) \cdot X^{-gb'})$$

Algorithm 2 Efficient automorphism-based Bootstrapping with NTRU and LWE

```
Input:
    An LWE ciphertext $\mathsf{ct} = (\mathbf{a}, b = -\langle \mathbf{a}, \mathbf{s} \rangle - \lfloor \frac{q}{t} \rceil \cdot m + e) \in \mathsf{LWE}^n_{\mathbf{s},2N}(m)$ with an odd number of all entries.
    An evaluation key $\mathbf{EVK}$.
    An LWE key switching key $\mathsf{ksk}_{\mathbf{s}}(\phi(f))$ as shown in Section A.
Output:
    An LWE sample $\mathsf{ct}' \in \mathrm{LWE}^n_{\mathbf{s},q}(f(m))$.
1:  $(\mathbf{a}', b') = \mathsf{ModSwitch}_{q \to 2N, \mathsf{odd}}(\mathsf{ct})$.
2:  Set $\mathrm{TestP}(X) = \sum_{i=0}^{N-1} \frac{Q}{t} \cdot f(\lfloor \frac{t}{q} \cdot i \rceil) \cdot X^i$, and let $\mathsf{acc} = (\mathrm{TestP}(X^{-g}) \cdot X^{-gb'}) \odot_A \mathbf{BRK}'$
3:  $v \leftarrow 0$
4:  for $(l = N/2 - 1; l > 0; l = l - 1)$ do
5:      for all $j \in I_l^+$ and $k \in I_l^-$ do
6:          if $a'_j = g^l$, let $\mathsf{acc} = \mathsf{acc} \odot_A \mathbf{BRK}_j^+$. end if
7:          if $a'_k = -g^l$, let $\mathsf{acc} = \mathsf{acc} \odot_A \mathbf{BRK}_k^-$. end if
8:      end for
9:      $v \leftarrow v + 1$
10:     if $(I_{\ell-1} \neq \emptyset$ or $v = w$ or $l = 1)$
11:         $\mathsf{acc} = \mathsf{HomAuto}_{g^v}(\mathsf{acc}, \mathbf{KSK}_{g^v})$
12:         $v \leftarrow 0$
13:     end if
14: end for
15: for all $j \in I_0^+$ and $k \in I_0^-$ do
16:     if $a'_j = 1$, let $\mathsf{acc} = \mathsf{acc} \odot_A \mathbf{BRK}_j^+$. end if
17:     if $a'_k = -1$, let $\mathsf{acc} = \mathsf{acc} \odot_A \mathbf{BRK}_k^-$. end if
18: end for
19: $\mathsf{ct}' = \mathsf{SampleExtract}(\mathsf{acc})$
20: $\mathsf{ct}' = \mathsf{ModSwitch}_{Q \to Q_k}(\mathsf{ct}')$
21: $\mathsf{ct}' = \mathsf{LWE.KeySwitch}(\mathsf{ct}')$
22: $\mathsf{ct}' = \mathsf{ModSwitch}_{Q_k \to q}(\mathsf{ct}')$
23: return $\mathsf{ct}'$
```

by using external product with $\mathbf{BRK}'$. For $l = 2N - 1$, we perform the external product for all the terms a'_j and a'_k that satisfy the $a'_j = g^l$ and $a'_k = -g^l$ together with $\mathbf{BRK}_j^+$ and $\mathbf{BRK}_k^-$. Then, by homomorphically evaluating $\mathsf{HomAuto}_g : X \to X^g$ in line 7, we have

$$\mathsf{acc} = \mathrm{NTRU}_{f,Q}\left(\mathrm{TestP}(X) \cdot X^{b'} \cdot X^{g\left(\sum_{j \in I^+_{N/2-1}} s_j - \sum_{k \in I^-_{N/2-1}} s_k\right)}\right),$$

By repeating the above process and combining the window size technique, we can get the following result according to Equation 6.

$$\begin{aligned}\mathsf{acc} &= \mathrm{NTRU}_{f,Q}\left(\mathrm{TestP}(X) \cdot X^{b'} \cdot X^{\sum_{i=0}^{n-1} a'_i s_i}\right) \\ &= \mathrm{NTRU}_{f,Q}\left(\mathrm{testP}(X) \cdot X^{b' + \sum_{i=0}^{n-1} a'_i s_i}\right) \\ &= \mathrm{NTRU}_{f,Q}\left(\mathrm{TestP}(X) \cdot X^{-\left(\lfloor \frac{q}{t} \rceil \cdot m + e\right)}\right).\end{aligned}$$

After performing the sample extraction, key-switching, and modulus switching, we can obtain the LWE ciphertext $\mathsf{ct}' = \mathrm{LWE}^n_{\mathbf{s},q}(f(m)) \in \mathbb{Z}_q^{n+1}$ as described in Theorem 1.

Noise analysis. Firstly, by performing the approximate gadget decomposition $\mathsf{acc} = \mathrm{TestP}(X) \odot_A \mathbf{BRK}'$, and the noise variance of the noise is

$$\mathsf{Var}(g') \leq Nd' \frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{NP^2}{24}.$$

For each loop, there are l external products with approximate gadget decompositions $\odot_A$ and one key-switching with exact gadget decomposition $\odot$. Thus, we have:

$$\begin{aligned}\mathsf{Var}(\mathsf{err}(\mathsf{acc})) &\leq lNd' \frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{lNP^2}{24} + Nd\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{KSK})) \\ &\leq (ld' + d)N\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{lNP^2}{24}.\end{aligned}$$

As previously analyzed, the blind rotation process involves n external products and $\frac{\omega-1}{\omega}\kappa + \frac{N}{2\omega}$ automorphisms. Thus, we can get the variance of the noise during the blind rotation as

$$\mathsf{Var}(\mathsf{err}(\mathsf{BR})) \leq \left(nd' + d\left(\frac{\omega-1}{\omega}\kappa + \frac{N}{2\omega}\right)\right) N\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{nNP^2}{24}.$$

Furthermore, after modulus switching and key-switching, the variance of the error in bootstrapping process is

$$\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \leq \frac{q^2}{Q_k^2} \cdot [\frac{Q_k^2}{Q^2} \cdot \mathsf{Var}(\mathsf{err}(\mathsf{BR})) + \frac{2+N}{24} + Nd_k \cdot \mathsf{Var}(\mathsf{err}(\mathsf{ksk}))] + \frac{2+n}{24}.$$

Finally, we give a detailed comparison among different automorphism-based blind rotation methods in Table 5. The result demonstrates that our hybrid method involves the smaller number of NTTs and Hadamard multiplications compared to other schemes.

Table 4: Comparison of noise varianceof automorphism-based blind rotation among different schemes. Here, σ^2_{BR} is the variance of blind rotation key, and $d' < d$.

Schemes	Methods	Noise variance
[LMK+23]	Auto. & Window Size	$d(n + (\frac{\omega-1}{\omega}\kappa + \frac{N}{\omega}))N\frac{B^2}{6} \cdot \sigma^2_{\mathsf{BR}}$
[XZD+23]	Auto.	$d(2n+1)N\frac{B^2}{12} \cdot \sigma^2_{\mathsf{BR}}$
Approach 1	Auto. & Window Size	$(nd' + d(\frac{\omega-1}{\omega}\kappa + \frac{N}{\omega}))N\frac{B^2}{12} \cdot \sigma^2_{\mathsf{BR}} + \frac{nNP^2}{24}$
Hybrid Approach	Auto. & Window Size	$(nd' + d(\frac{\omega-1}{\omega}\kappa + \frac{N}{2\omega}))N\frac{B^2}{12} \cdot \sigma^2_{\mathsf{BR}} + \frac{nNP^2}{24}$

Table 5: Comparison of computational cost of automorphism-based blind rotation among different schemes, where HMs is the Hadamard multiplications, and $d' < d$.

Schemes	# NTTs	# HMs
[LMK+23]	$(2n+424)(d+1)$	$d(4n+848)$
[XZD+23]	$(n+465)(d+1)$	$d(n+465)$
Approach 1	$n(d'+1) + 375(d+1)$	$d'n + 375d$
Hybrid Approach	$n(d'+1) + 305(d+1)$	$d'n + 305d$

5 Parameters, Implementations, and Comparisons

In this section, we start by presenting the parameter settings, error growth, and decryption failure rates, providing a comprehensive understanding of the experimental setup. Building upon this parameter setting, we describe the implementation details of the algorithm. Finally, we present the experimental results that demonstrate the efficiency and performance improvements achieved by our approach.

5.1 Parameters

Firstly, we give the symbolic notations for the parameters in Appendix D, which includes dimension, modulus, distribution, security parameters, etc. Then, we use the parameters **128B** and **128G** to indicate the binary and Gaussian distributions that are used in Algorithm 1 and 2, respectively. In Table 6 lists the detailed bootstrapping parameters for the NTRU and NGS ciphertexts. Specifically, we generate the secret key for NTRU from the ternary distribution on $\{-1, 0, 1\}$, where 0 occurs with probability 1/2, 1 and −1 occur with probability 1/4. As [BIP+22] mentioned, the ternary distribution approximates a discrete Gaussian with standard deviation $\sigma = 1/\sqrt{2}$.

In addition, the modulus of NTRU ciphertext satisfies $Q < N^{2.484}$, which is smaller than the fatigue point in [DvW21] to avoid sublattice attacks on NTRU problems. The approximation factor P, gadget decomposition base B, and length d' are used in NTRU-based external products. Then, we also present the LWE parameters in Table 7. It is worth noting that different key distributions correspond to different LWE dimensions at the same security level.

Table 6: Bootstrapping parameters for NTRU/NGS ciphertext.

Parameters	Key distrib.	λ	N	σ	P	Q	B	d'
128B	Ternary	128	1024	$1/\sqrt{2}$	2^5	$\approx 2^{19.9}$	2^3	5
128G	Ternary	128	1024	$1/\sqrt{2}$	2^4	$\approx 2^{19.9}$	2^4	4

Table 7: Bootstrapping parameters for LWE ciphertext.

Parameters	Key distrib.	λ	n	σ	q	Q_k	B_k	d_k
128B	Binary	128	512	3.19	512	2^{14}	2^7	2
128G	Gaussian	128	465	3.19	2048	2^{14}	2^7	2

Table 8: Bootstrapping parameters for other schemes.

Scheme	Key distrib.	λ	n	N	σ	Q	B	d	d'
[MP21]	Binary	128	512	1024	3.19	$\approx 2^{25}$	2^7	4	–
[BMMP18, CGGI20]	Binary	128	636	1024	2^7	2^{32}	2^6	–	3
[LMK+23]	Gaussian	128	458	1024	3.19	$\approx 2^{25}$	2^7	4	–
[XZD+23]	Gaussian	128	465	1024	3.19	$\approx 2^{19.9}$	2^4	5	–

To evaluate the security of NTRU ciphertext, we use the NTRU estimator offered by Ducas and van Woerden [DvW21] to find the BKZ block size β for the Dense Sublattice Discovery (DSD) attack. In detail, we use the cost model $T(d, \beta) := 2^{0.292\cdot\beta+16.4+\log_2(8\cdot 2N)}$ to estimate the concrete security of NTRU. On the other hand, we use the LWE estimator [APS15] to estimate the security level for the LWE sample, which calculates the complexity of primal attacks via the shortest vector problem, decoding, and dual-lattice attacks. In particular, we can get the fact that the parameters of NTRU and LWE provide at least a 128-bit security level. Finally, to facilitate comparisons, we provide the parameters for other schemes under 128-bit security level in Table 8. Note that the parameters not listed in the table are the same as those provided by our scheme.

Noise Growth and Failure Probability of Decryption. In previous sections, we present a theoretical analysis of noise growth. Recall that during blind rotation, modulus switching, and key-switching operations, the noise growth causes the final error that follows a Gaussian distribution with a standard deviation of

$$\sigma = \sqrt{\frac{q^2}{Q_k^2} \cdot \left(\frac{Q_k^2}{Q^2} \cdot \sigma_{\mathsf{acc}}^2 + \sigma_{ks}^2\right) + \sigma_{ms}^2}.$$

Among these operations, σ_{acc}^2 is the primary source of noise growth during bootstrapping. By using the approximate decomposition technique mentioned in Section 3.1, we can reduce the length of the gadget decomposition under the same magnitude of noise growth. For instance, for the NTRU/NGS parameters of the [XZD+23] scheme, we can use the approximate decomposition technique to reduce the decomposition length from $d = 5$ to $d' = 4$, as shown in parameter **128G**.

Furthermore, the NAND gate bootstrapping is instantiated in the functional bootstrapping of our experiments, as shown in Figure 1, which requires the plaintext modulus to be set to $t = 4$. In this way, the probability of decryption failure can be calculated using the following formula:

$$1 - \mathsf{erf}\left(\frac{q/8}{2\sigma}\right), \tag{7}$$

where erf is the Gaussian function. According to the analysis of the noise growth and the setting of the parameters, we can get the failure probability of decryption for parameters **128B** and **128G** are 2^{-31} and 2^{-35}, respectively, which are close to the result of [XZD+23].

5.2 Performance Analysis and Comparison

5.2.1 C Implementation

We implemented the gate bootstrapping in the OpenFHE library v1.0.4 [BBB+22]. The evaluation environment was a commodity desktop computer system with an Intel(R) Core(TM) i5-11500 CPU 2.70 GHz and 32 GB of RAM, running Ubuntu 22.04.2 LTS with a single thread at a single CPU core. The compiler was clang 11.3.0. Table 9 summarizes the runtimes for these similar schemes, and each result is an average of 5,000 executions. Our CMux-based algorithm only takes 39ms to evaluate a gate bootstrapping, which is about 2.7 times faster than the FHEW [MP21] scheme and 1.4 times faster than the FINAL [BIP+22] scheme. On the other hand, for the automorphisms-based algorithm, our approach also achieves 2.4× and 1.3× speedups compared to the [LMK+23] and [XZD+23] schemes, respectively.

Additionally, we also provide a comparison of key sizes in Table 9. Notably, the utilization of NTRU-based accumulator and approximate decomposition techniques helps reduce the key sizes compared to RLWE-based accumulator and exact decomposition. However, due to using the key unrolling technique in Algorithm 1, our result is 1.5× larger than the traditional method, i.e. $\frac{3d'nN}{2}\log Q$ bits. In Algorithm 2, we requires some additional blind rotation keys and automorphism keys in hybrid method, totaling $(2nd' + \omega d)N \log Q$ bits. Finally, the bootstrapping procedure also requires $nd_k B_k N \log Q_k$ bits for LWE key-switching keys, which can be performed with the same strategy. Thus, we omit this comparison in this Table.

Table 9: Comparison of key sizes and running times, where the CMux-based methods use the binary secret key, and the NTT is used to accelerate polynomial multiplication in all schemes.

Schemes	Assumption	Key sizes (MB)	Times (ms)
FHEW [MP21]	RLWE	13.5	106
FINAL [BIP+22]	NTRU	6.5	57
Our work I	NTRU	9.3	39
LMK [LMK+23]	RLWE	12.7	112
XZD [XZD+23]	NTRU	17.9	62
Our work II	NTRU	9.2	46

5.2.2 Implementation with AVX Instructions

It should be noted that some advanced implementations introduce AVX instructions to improve performance: Examples of this include TFHE-pp [MBM+21] works with AVX-512, and TFHE-rs [Zam22] with AVX-2 and AVX-512. The AVX instructions can be used to effectively vectorize NTT, Hadamard multiplication, and gadget decomposition for efficient parallel computation. Since the OpenFHE library does not yet fully support these accelerated instructions, we developed a new library that incorporates the Intel AVX-2 and AVX-512 instructions [Int21] for our CMux-based method.

The computational efficiency with the AVX instruction is significantly impacted by the size of the modulus and registers. The standard method for performing the modular reduction operation in NTT requires twice the width for each coefficient in the vector registers to accommodate intermediate full products. We use Montgomery's algorithm[Mon85] to efficiently perform the modular reduction, where the high and low parts of the product are computed individually. These results only need to be stored in single precision within 32 bits, reducing the register space of coefficients and achieving $16\times$ parallelism in the NTT with the AVX-512 instruction.

Our algorithms also benefit from vectorized operations. In TFHEpp and TFHE-rs implementations, the modulus Q is typically chosen as either $Q = 2^{32}$ or $Q = 2^{64}$, whereas our scheme chooses a NTT-friendly modulus $Q \approx 2^{19.9}$, as shown in Section 2.6. In addition, TFHE-like schemes use FFT to perform polynomial multiplication, where intermediate computation only requires floating-point addition and multiplication without any modulo operations. However, since the intermediate results of the FFT need to be stored in double precision in the registers, the parallelism of the FFT is not as efficient as the NTT utilized by AVX-512 and AVX2 instructions. In Table 10, we show the parallelisms of NTT and FFT using the AVX instruction.

Table 10: Number of coefficients processed per Intel AVX instruction with different size moduli for NTT and FFT, where $N = 1024$, and subscript indicates the modulus.

Transformation Type	NTT_{16}	NTT_{32}	NTT_{64}	FFT_{32}
AVX2	16	**8**	4	4
AVX-512	32	**16**	8	8

Finally, Table 11 shows the experimental results using AVX instructions. From the table, we can see that the proposed CMux-based bootstrapping algorithm takes only 5.5

Table 11: Comparison of bootstrapping runtimes with AVX instructions.

Implementation	Instruction	Poly. multiplication	Times (ms)
TFHE-rs [Zam22]	AVX-2	FFT	10.2
Our work	AVX-2	NTT	5.5
TFHEpp [MBM$^+$21]	AVX-512	FFT	9
TFHE-rs [Zam22]	AVX-512	FFT	6.8
Our work	AVX-512	NTT	3.8

ms for the AVX-2 instruction and 3.8 ms for the AVX-512 instruction. Compared to the C implementation, the AVX2 and AVX-512 instructions achieve 7.1× and 10.2× speedups, respectively, by vectorizing NTT, Hadamard multiplication, and gadget decomposition operations. In comparison to the TFHE-like implementation, our CMux-based approach is 1.9 times faster than the AVX2 implementation. When compared to the state-of-the-art CPU implementation in TFHE-rs [Zam22] with AVX-512 instructions, we can also achieve around 1.8× speedup. The remarkable performance improvement can be attributed to the optimization of algorithm and the inherent parallelism advantage of NTT over FFT.

6 FPGA Implementation

To improve FHE and bootstrapping performance, a widely adopted strategy is to offload computationally intensive FHE computations to acceleration platforms equipped with substantial hardware resources. This plays a crucial role in overcoming the computational bottlenecks that hinder the practical adoption of FHE. In this context, we seamlessly incorporate the methodology presented in this paper into the field-programmable gate array (FPGA) platform to design an efficient hardware accelerator. This integration demonstrates remarkable performance boost over previous endeavors, making the bootstrapping scheme a significant advancement towards practical applications.

Since the CMux-based blind rotation involves less computational cost compared to automorphisms-based method, we implement Algorithm 1 on the FPGA and conduct a detailed breakdown of the execution path for the blind rotation procedure. Figure 3 illustrates the typical workflow of blind rotation. It commences with the accumulator of the NTRU ciphertext and proceeds through a sequence of operations, such as approximate gadget decomposition, NTT, Hadamard multiplication and accumulation (MAC), and INTT, and finally generates a new NTRU ciphertext. It's noteworthy that the entire blind rotation process needs to perform $n/2$ iterations. Subsequently, we provide a comprehensive description of the FPGA implementation in accordance with this workflow.

6.1 Overall Architecture

Figure 4 illustrates the overall architecture of the FPGA accelerator, called NFP, which consists of four key components:

- HOST CPU: The HOST CPU plays a crucial role in dispatching data and tasks to the FPGA. As the central control unit of the entire accelerator system, it can allocate computing tasks and data with high efficiency.
- High Bandwidth Memory (HBM): HBM has high bandwidth characteristics to ensure rapid data transmission to the computing module. Specifically, our accelerator design

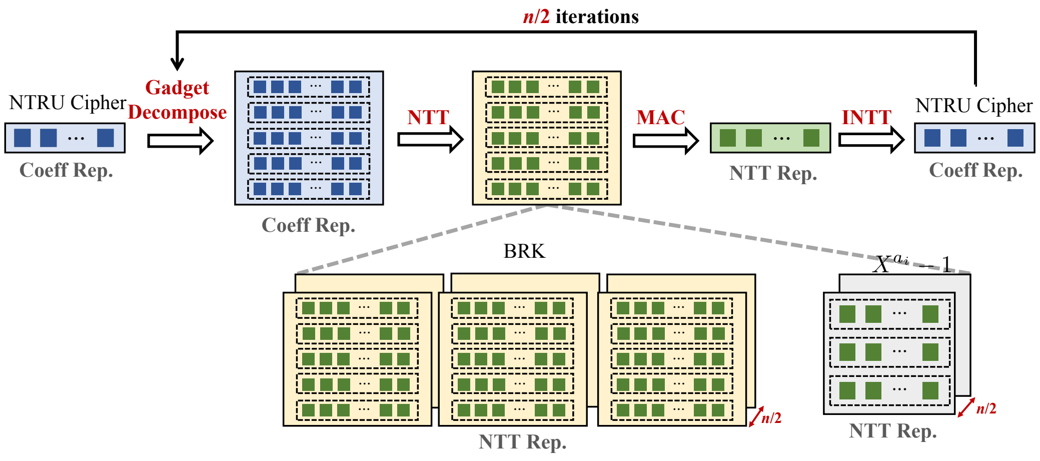


Figure 3: The typical workflow of blind rotation

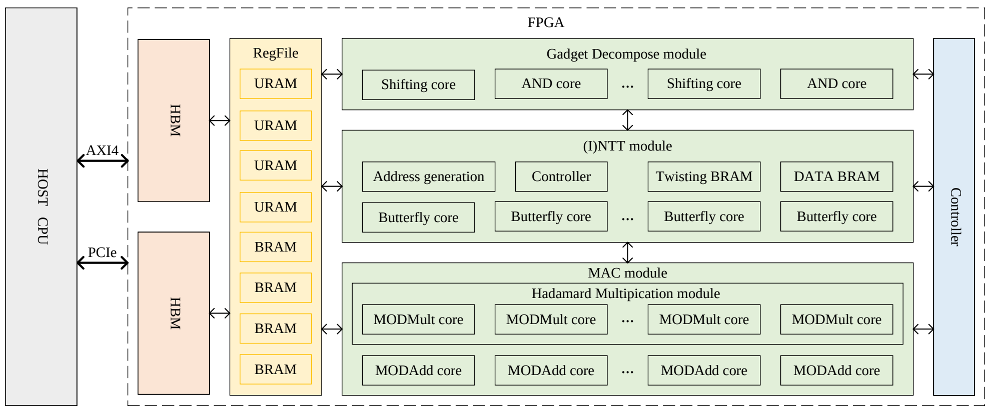


Figure 4: Overall architecture of NFP

utilizes HBM to store auxiliary data (e.g., NTT table for $Y^i - 1$) which are used in the blind rotation process.

- On-chip Register File: The on-chip register file acts as a bridge connecting the HBM and the computing module, facilitating seamless data exchange during the calculation process. Additionally, it provides sufficient memory space for storing blind rotation keys (BRK).
- Computing Modules: The computing modules execute key operators according to the tasks scheduled by the host CPU, mainly including gadget decomposition, (I)NTT and MAC modules.

Next, we provide an in-depth explanation of the key functional units, vector chaining, and memory design.

6.2 Functional Units

Our accelerator design focuses on decomposing coarse-grained gate bootstrapping into fine-grained operation-level computational units. This design maximizes parallelism and

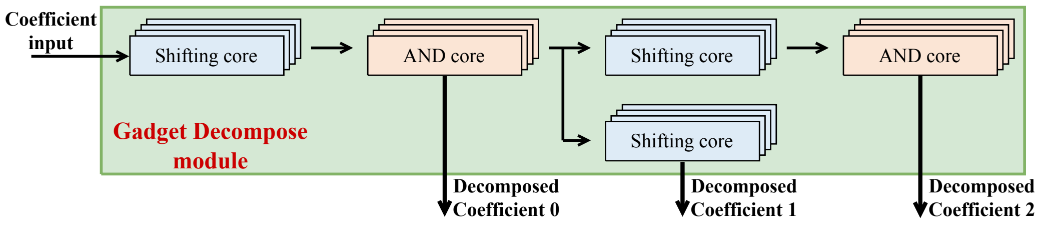


Figure 5: DSP-friendly Gadget decomposition implementation

enhances the flexibility of the acceleration scheme. The core operations primarily include modular multiplication, modular addition (subtraction), (I)NTT, etc. By integrating and scheduling these core operators, we can achieve efficient execution of bootstrapping on FPGA.

DSP-friendly Gadget Decomposition. The gadget decomposition involves intensive division operations, which not only leads to a significant increase in computation time, but also consumes additional DSP resources. To tackle this challenge, we employ computationally efficient shift and bitwise AND operations, as depicted in Figure 5. Given a polynomial with coefficient representation, the gadget decomposition module generates three output coefficients through a combination of shifting and AND operations. This approach ensures computational accuracy while significantly reducing the computational load on the FPGA's computing cores.

High-throughput NTT/INTT Modules. The NTT plays a crucial role in enhancing the efficiency of polynomial arithmetic operations in FHE. The overall performance of the accelerator is greatly impacted by the efficiency of NTT operations. Thus, we prioritize the design of a high-throughput NTT module to ensure the efficient execution of NTT calculations.

A typical NTT implementation consists of a series of interconnected butterfly units, encompassing fundamental operations such as modular multiplication and modular addition (subtraction). The greater the number of butterfly units, the higher the throughput, and the lower the computational latency for NTT calculations. It's essential to emphasize that these modular operations are associated with relatively high computational costs.

As illustrated in Figure 6, to fully utilize the finite resources of the FPGA and optimize computational latency, we introduce dedicated computing cores for modular multiplication and addition (subtraction) within the NTT module. Consequently, the NTT module does not share these cores with other modules, eliminating any competition for these cores when NTT and other modules operate concurrently. Additionally, to enhance the throughput of NTT calculations, we optimize the butterfly units and minimize resource usage (e.g., LUTs, DSPs) as much as possible for this implementation. In our case, the optimized NTT module can instantiate 256 processing elements (PEs).

6.3 Vector Chaining

To enhance the execution efficiency of accelerator for blind rotation procedure, we introduce a schedule-optimized vector chaining mechanism. By interconnecting these computation units as shown in Figure 7, the intermediate results of each key operation within the blind rotation process can flow seamlessly to the subsequent computation unit. This eliminates the need for frequent read and write operations on the on-chip memory. As a result, we reduce memory access time and alleviate the read-write pressure on the register file.

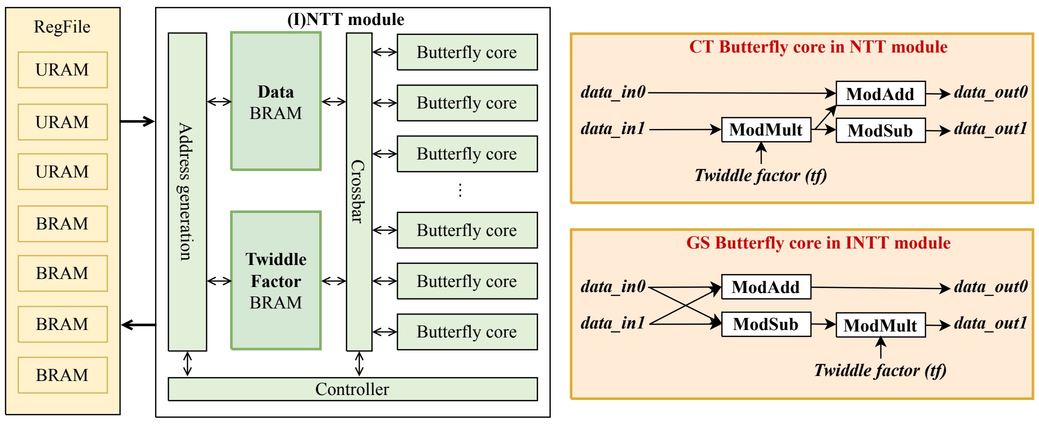


Figure 6: Architecture of high-throughput (I)NTT module

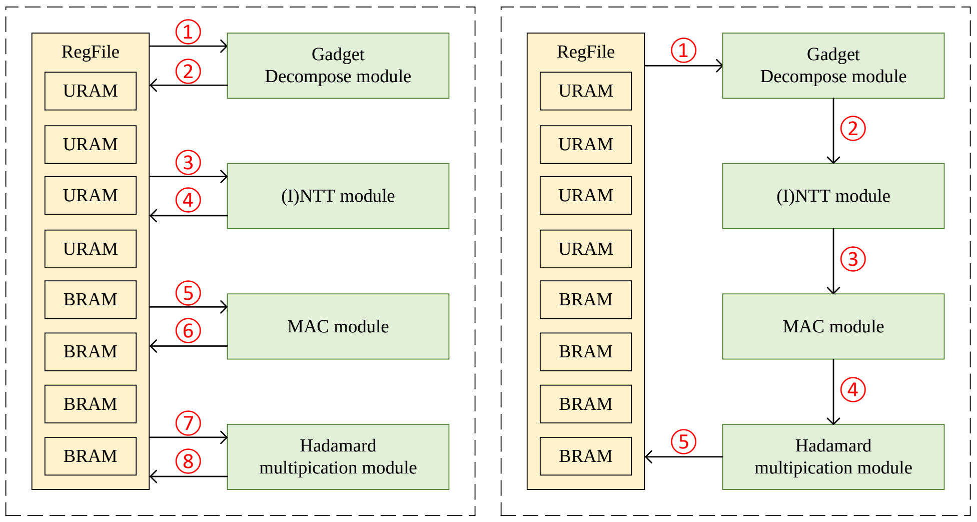


Figure 7: Schedule-optimized vector chaining design

6.4 Memory Design

Our accelerator is a vector processor equipped with specialized computational and storage modules tailored for FHE operations. Within the computational module, we instantiate 256 computing units, each capable of handling 512 operands per cycle. Both data processing and storage operate at a rate of 512 data/cycle. The accelerator features multiple register files, designed to match the bandwidth of the computing cores. One end of these register files is directly connected to the High-Bandwidth Memory (HBM), with data being organized within the register files and then supplied to the designated computing cores. Due to the pipelined nature of the computing cores, the theoretical throughput can potentially reach up to 512 data/cycle. However, despite the high throughput capabilities of the accelerator, the storage of FHE ciphertext and key sizes remains a significant challenge.

To address this issue, we utilize HBM as off-chip storage, which provides a generous storage capacity of up to 8GB and features a rapid transfer rate of 16GB/s, perfectly aligning with the storage requirements of our accelerator. Specifically, the HBM is connected to the host CPU through a PCIe interface, which enables efficient read and write operations in the expanded HBM memory space. This approach proves especially beneficial when handling substantial data volumes like ciphertext and keys. For smaller parameters, the host CPU utilizes the AXI4-Lite interface to access the cache space within the on-chip register file. This facilitates the transmission of computational tasks and system parameters such as Montgomery reduction. In essence, the host CPU establishes crucial connections to the FPGA through the PCIe and AXI4-Lite interfaces, enabling efficient data access and transfer.

Although high-bandwidth memory (HBM) provides ample storage capacity, its memory access time is both random and relatively long, typically around 30 cycles. We introduce a register file between the HBM and the computing cores. The register file acts as a cache and proactively accesses the HBM to fetch the required data for calculations. By doing so, the compute module can efficiently access the register file, retrieving the necessary source operands in just a single cycle. This approach minimizes the impact of HBM's access latency on overall computation performance.

Analysis. During the blind rotation process, two types of auxiliary data are required: blind rotation keys **BRK** and the NTT table for $Y^i - 1$. It's important to note that not all of this precomputed data is used in one iteration; instead, three or nine of them are utilized per iteration. Therefore, we store precomputed data on the HBM and employ a data prefetching technique to mitigate the delay in transferring data from the HBM to on-chip memory. Specifically, during the execution of the NTT operation for acc, we proactively read the corresponding NTT representation of $Y^i - 1$ into the register file. As a result, the computation for MAC can be performed directly, without waiting for data retrieval.

6.5 Evaluation

6.5.1 Methodology

Software and hardware configuration. We build a real-world experimental environment based on an x86 CPU system. On the host side, we integrate Xilinx's extensive developing toolkit. The Verilog RTL code is compiled into bit files utilizing Xilinx Vivado (version 2022.1) and subsequently loaded onto the Xilinx Alveo U280 FPGA. The host system is equipped with a C++-coded runtime environment, which works in close collaboration with the FPGA. Furthermore, communication between the host and the Alveo U280 takes place via the PCIe interface.

Benchmarks. We focus on evaluating the performance of FPGA accelerator under different metrics, such as hardware resource consumption and execution time. To emphasize its superiority, we also conduct a thorough comparison with existing ASIC accelerators like MATCHA [JLJ22] and STRIX [PCK+23], as well as typical FPGA designs such as XHEC [NOMP22], YKP [YKP22], and FPT [VBDV22].

6.5.2 Performance Results

In terms of FPGA acceleration performance, NFP achieves remarkable efficiency in blind rotation operations. With the **128B** parameter setting, our experimental results demonstrate a latency of only 0.92ms, which brings around 4.1× performance improvement compared to the CPU implementation. In contrast to FPGA accelerators like YKP [YKP22], NFP not only utilizes fewer resources but also achieves a speedup of up to 2× (1.88ms vs. 0.92ms).

It's worth mentioning that FPT[VBDV22] introduces a rapid FPGA implementation that achieves a runtime of just 0.58ms through the use of the pipelined FFT technique. However, this technique comes at the cost of increased approximation error, resulting in a decryption failure rate of approximately 2^{-15}. To ensure a fair comparison with

[VBDV22], we adjust the bootstrapping parameters to align the decryption failure rates and security levels as closely as possible, as explained in Appendix E. While maintaining a similar decryption failure rate to FPT, NFP utilizes almost identical hardware resources but achieves up to 2× performance improvement (0.58ms vs. 0.29ms).

Besides, we can see that NFP exhibits a performance level that is roughly half that of MATCHA and STRIX. Nevertheless, it's worth noting that current ASIC accelerator implementations still face significant hurdles, particularly in terms of development expenses and cycle duration. In conclusion, the notable enhancements in performance achieved by NFP stem from a combination of algorithmic refinements and the incorporation of numerous computational and memory enhancements within our accelerator design.

Table 12: Comparison bootstrapping of hardware implementations, where FR is the decryption failure rate, throughput is calculated by number of bootstrapping operations per second.

Works	FR	Resource LUT / FF / DSP / SRAM	Latency ms	Thr.[†] GBS/s	Thr./DSP GBS/s
MATCHA [JLJ22]	-	36.96 mm^2 16nm PTM [SYC+12]	0.20	5000	-
STRIX [PCK+23]	-	141.37 mm^2 28nm TSMC	0.16	6250	-
XHEC [NOMP22]	-	520K / 659K / 4096 / 167Mb	-	-	0.63
YKP [YKP22]	-	842K / 662K / 7202 / 338Mb	3.76	265	0.036
	-	442K / 342K / 6910 / 409Mb	1.88	531	0.076
FPT [VBDV22]	2^{-15}	595K / 1024K / 5980 / 14Mb	0.58	1724	0.288
NFP	2^{-31}	891K / 217K / 4508 / 34 Mb	0.92	1087	0.241
	2^{-15}	891K / 217K / 4508 /33 Mb	0.29	3448	0.765

7 Application

In this section, we briefly discuss practical applications of the proposed NTRU-based bootstrapping. Our bootstrapping can be utilized to instantiate logical gates such as AND, XOR, and OR gates, which in turn can be used to construct adders and multipliers. For example, a full adder takes two binary inputs, A_i and B_i, along with a carry bit C_{i-1} from the previous stage. It produces two outputs: the sum bit S_i and the carry bit C_i for the next stage as follows.

$$S_i = A_i \oplus B_i \oplus C_{i-1},$$

$$C_i = A_i B_i + C_{i-1}(A_i \oplus B_i).$$

This functionality can be achieved using 5 gate bootstrappings, and the TFHE-rs library requires $5 \times 6.8 = 34$ms to achieve this task. However, our technique significantly improves this time to just $5 \times 3.8 = 19$ms with CPU implementation and $5 \times 0.29 = 1.45$ms with FPGA implementation.

In addition, we focus on a hybrid homomorphic encryption framework. It sends data from the client to the server using symmetric ciphertexts and lets the server homomorphically decrypt the symmetric ciphertexts to obtain the FHE ciphertext, which has the advantage of reducing the size of transmitted ciphertexts and supporting homomorphic computations. In the schemes [TCBS23] and [WWL+23], the homomorphic evaluation of AES-128 ciphertexts was performed using the FBS and CBS modes of the FHEW/TFHE-like scheme. In contrast, we can use the GBS evaluation mode with NTRU-based bootstrapping. As mentioned in [MG20], AES circuits can consist of 33616 binary gates. Consequently, we can convert an AES-128 ciphertext into an NTRU-based FHEW/TFHE-like ciphertext in 2.1 minutes with a CPU implementation.

8 Conclusion

In this paper, we present two improved bootstrapping schemes based on GSW-like NTRU ciphertexts, aiming to elevate the performance of FHE. The first scheme employs a CMux-based blind rotation method, leveraging techniques such as approximate gadget decomposition and key unrolling to improve the efficiency of blind rotation. The second scheme involves automorphism-based blind rotation, utilizing a hybrid window size method, which significantly reduces the number of required automorphisms.

To further boost performance, we introduce an advanced implementation utilizing AVX-512 instructions on CPUs. Experimental results demonstrate that our method can execute a NAND gate bootstrapping operation in just 3.8ms, achieving approximately 1.8$\times$ performance gain compared to the state-of-the-art TFHE-rs implementation. Finally, we propose a more efficient FPGA accelerator that accelerates the entire blind rotation process from both computational and memory design perspectives. We conclude that the proposed technique can improve the efficiency of applications with the gate bootstrapping mode.

Acknowledgments

We are grateful for the helpful comments from the anonymous reviewers. This work was supported by the Huawei Technologies Co., Ltd and CAS Project for Young Scientists in Basic Research (Grant No. YSBR-035).

References

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[BBB$^+$22] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[BIP$^+$22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder VL Pereira, and Nigel P Smart. Final: Faster fhe instantiated with ntru and lwe. *Cryptology ePrint Archive*, 2022.

[BLLN13] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding: 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings 14*, pages 45–64. Springer, 2013.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference,* *Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38*, pages 483–512. Springer, 2018.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology–CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings 31*, pages 505–524. Springer, 2011.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *international conference on the theory and application of cryptology and information security*, pages 3–33. Springer, 2016.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pages 409–437. Springer, 2017.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015.

[DvW21] Léo Ducas and Wessel van Woerden. Ntru fatigue: how stretched is overstretched? In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV 27*, pages 3–32. Springer, 2021.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[GNT$^+$21] Serhan Gener, Parker Newton, Daniel Tan, Silas Richelson, Guy Lemieux, and Philip Brisk. An fpga-based programmable vector engine for fast fully homomorphic encryption over the torus. In *SPSL: Secure and Private Systems for Machine Learning (ISCA Workshop)*, 2021.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings*, pages 267–288. Springer, 1998.

[Int21] Intel Corporation. *Intel® Architecture Instruction Set Extensions Programming Reference*, 2021.

[Jam80] Frederick James. Monte carlo theory and practice. *Reports on progress in Physics*, 43(9):1145, 1980.

[JLJ22] Lei Jiang, Qian Lou, and Nrushad Joshi. Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 235–240, 2022.

[JP22a] Marc Joye and Pascal Paillier. Blind rotation in fully homomorphic encryption with extended keys. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, pages 1–18. Springer, 2022.

[JP22b] Marc Joye and Pascal Paillier. Blind rotation in fully homomorphic encryption with extended keys. In *Cyber Security, Cryptology, and Machine Learning: 6th International Symposium, CSCML 2022, Be'er Sheva, Israel, June 30–July 1, 2022, Proceedings*, pages 1–18. Springer, 2022.

[Klu22] Kamil Kluczniak. Ntru-v-um: Secure fully homomorphic encryption from ntru with small modulus. *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 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*, pages 1219–1234, 2012.

[LMK$^+$23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. 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*, pages 227–256. Springer, 2023.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.

[MAAM20] Toufique Morshed, Md Momin Al Aziz, and Noman Mohammed. Cpu and gpu accelerated fully homomorphic encryption. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 142–153. IEEE, 2020.

[MBM$^+$21] Kotaro Matsuoka, Ryotaro Banno, Naoki Matsumoto, Takashi Sato, and Song Bian. Virtual secure platform: A five-stage pipeline processor over tfhe. In *USENIX Security Symposium*, pages 4007–4024, 2021.

[MG20] Kalikinkar Mandal and Guang Gong. Can lwc and pec be friends?: Evaluating lightweight ciphers in privacy-enhancing cryptography. In *Fourth Lightweight Cryptography Workshop. NIST*, 2020.

[Mon85] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[MP21] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in fhew-like cryptosystems. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 17–28, 2021.

[NOMP22] Kevin Nam, Hyunyoung Oh, Hyungon Moon, and Yunheung Paek. Accelerating n-bit operations over tfhe on commodity cpu-fpga. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[PCK+23] Adiwena Putra, Yi Chen, John Kim, Joo-Young Kim, et al. Strix: An end-to-end streaming architecture with two-level ciphertext batching for fully homomorphic encryption with programmable bootstrapping. *arXiv preprint arXiv:2305.11423*, 2023.

[Per21] Hilder Vitor Lima Pereira. Bootstrapping fully homomorphic encryption over the integers in less than one second. 2021.

[Pol71] John M Pollard. The fast fourier transform in a finite field. *Mathematics of computation*, 25(114):365–374, 1971.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[SYC+12] Saurabh Sinha, Greg Yeric, Vikas Chandra, Brian Cline, and Yu Cao. Exploring sub-20nm finfet design with predictive technology models. In *Proceedings of the 49th Annual Design Automation Conference*, pages 283–288, 2012.

[TCBS23] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. A homomorphic aes evaluation in less than 30 seconds by means of tfhe. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 79–90, 2023.

[VBDV22] Michiel Van Beirendonck, Jan-Pieter D'Anvers, and Ingrid Verbauwhede. Fpt: a fixed-point accelerator for torus fully homomorphic encryption. *arXiv preprint arXiv:2211.13696*, 2022.

[WWL+23] Benqiang Wei, Ruida Wang, Zhihao Li, Qinju Liu, and Xianhui Lu. Fregata: Faster homomorphic evaluation of aes via tfhe. In *International Conference on Information Security*, pages 392–412. Springer, 2023.

[XZD+23] Binwu Xiang, Jiang Zhang, Yi Deng, Yiran Dai, and Dengguo Feng. Fast blind rotation for bootstrapping fhes. Springer-Verlag, 2023.

[YKP22] Tian Ye, Rajgopal Kannan, and Viktor K Prasanna. Fpga acceleration of fully homomorphic encryption over the torus. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2022.

[Zam22] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.

A Useful Algorithms for LWE ciphertext

In this section, we present the sample extract, key-switching, modulus switching for LWE ciphertext.

A.1 Sample Extraction

The original sample extraction algorithm can extract some LWE samples from an RLWE ciphertext in the TFHE scheme [CGGI20]. Recently, Xiang et al. [XZD+23] extend this technique to the NTRU ciphertext. In particular, the NTRU ciphertext $\mathsf{ct} = (g+m)/f \in \mathcal{R}_Q$ can be viewed an RLWE ciphertext $(\mathsf{ct}, 0) \in \mathcal{R}_Q^2$, and the decryption process is written as

$$0 - \mathsf{ct} \cdot f = (g+m)/f \cdot f = g+m,$$

by using the secret key f. We define the sample extraction as

$$\mathsf{SampleExtraction}(\mathsf{ct}) = (a_0, -a_{N-1}, -a_{N-2}, ..., -a_1, 0) \in \mathrm{LWE}^N_{\phi(f),Q}(m_0),$$

where the noise does not increase in the process.

A.2 Key Switching for LWE ciphertext

We show the key-switching algorithm for LWE ciphertext in Lemma 4.

Lemma 4 (**LWE Key Switching**). *Input an LWE ciphertext* $\mathsf{ct} = (\mathbf{a}, b) \in \mathsf{LWE}^N_{\mathbf{z},Q}(m)$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}))$*, and the switching keys* $\mathsf{ksk}_{i,j,v} \in \mathsf{LWE}^n_{\mathbf{s},Q}\left(vz_iB_k^j\right)$*, where* $v \in \{0, \dots, B_k - 1\}$*, for all* $0 \le i \le N-1$*,* $0 \le j \le d_k - 1$*, and let* $d_k = \lceil \log_{B_k} Q_k \rceil$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathsf{ksk}))$*, the LWE key switching algorithm computes*

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

which outputs a new LWE ciphertext $\mathsf{ct}' \in \mathsf{LWE}^n_{\mathbf{s},Q}(m)$*, and its variance satisfies* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \le \mathsf{Var}(\mathsf{err}(\mathsf{ct})) + Nd_k \cdot \mathsf{Var}(\mathsf{err}(\mathsf{ksk}))$*, where* $d_k = \lceil \log_{B_k} Q \rceil$.

Proof. Let $\mathrm{ksk}_{i,j,v} = (\mathbf{a}'_{i,j,v}, \mathbf{a}'_{i,j,v} \cdot \mathbf{s} + vz_iB_k^j + e_{i,j,v})$ for some $\mathbf{a}'_{i,j,v} \in \mathbb{Z}_q^n$ and $e_{i,j,v} \in \chi_\delta$, the output ciphertext is

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

It outputs a new LWE ciphertext under the secret key $\mathbf{s}$, where $\mathbf{a}' = -\sum_{i,j} \mathbf{a}'_{i,j,a_{i,j}}$ and $b' = b - \mathbf{a} \cdot \mathbf{z} + \mathbf{a}' \cdot \mathbf{s} - \sum_{i,j} e_{i,j,a_{i,j}}$. Thus, the variance of the noise satisfies $\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \le \mathsf{Var}(\mathsf{err}(\mathsf{ct})) + Nd_k \cdot \mathsf{Var}(\mathsf{err}(\mathsf{ksk}))$. $\square$

A.3 Modulus Switching

The modulus switching technique can change the modulus of LWE ciphertext [BV11] without affecting the message as shown in Lemma 5.

Lemma 5 (**LWE Modulus Switching**). *Input an LWE ciphertext* $\mathsf{ct} = (\mathbf{a}, b) \in \mathrm{LWE}_{\mathbf{s},Q}(m)$ *with error variance* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}))$*, the modulus switching algorithm computes*

$$\mathsf{LWE.KeySwitch}(\mathsf{ct}) = (\lfloor \frac{q}{Q} \cdot \mathbf{a} \rceil, \lfloor \frac{q}{Q} \cdot b \rceil),$$

which outputs the LWE ciphertext ct' *under modulus* q*, and its variance satisfies* $\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \le (\frac{q}{Q})^2 \cdot \mathsf{Var}(\mathsf{err}(\mathsf{ct})) + \frac{n+2}{24}$.

Proof. Let the integers $Q > q > t$, the output ciphertext is

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

By checking the decryption function, we can get

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

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

Round-to-Odd Modulus Switching. Lee et al. [LMK+23] proposed a special modulus switching method to generate LWE ciphertexts with all entries odd as follows

$$\mathsf{ct}' = \mathsf{ModSwitch}_{\mathsf{odd}}(\mathsf{ct}) = (\lfloor \frac{q}{Q} \cdot \mathbf{a}\rceil_{\mathsf{odd}}, \lfloor \frac{q}{Q} \cdot b\rceil_{\mathsf{odd}}),$$

where $\lfloor\cdot\rceil_{\mathsf{odd}}$ outputs the nearest odd integer for the input value. The correctness of this step can be obtained directly from Lemma 5. For the noise growth, since the error introduced by rounding at this point follows the standard deviation of a uniform distribution in $[-1, 1]$, we can get variance is $\mathsf{Var}(\mathsf{err}(\mathsf{ct}')) \leq (\frac{q}{Q})^2 \cdot \mathsf{Var}(\mathsf{err}(\mathsf{ct})) + \frac{||\mathbf{s}||_2^2+1}{6}$.

□

B Correctness of Key Switching for NTRU Ciphertext

We show the proof of Lemma 2.

Proof. Let $\mathbf{KSK} = (\mathbf{g}/f + \mathfrak{g} \cdot f/f') \in \mathcal{R}_Q^d$, we have

$$\begin{aligned}\mathsf{ct}' &= \mathsf{NTRU.KeySwitch}(\mathsf{ct})\\ &= \left\langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{KSK}\right\rangle\\ &= \frac{\left\langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{g}\right\rangle)}{f} + \frac{g+\mu}{f} \cdot \frac{f}{f'}\\ &= \frac{\left\langle \mathfrak{g}^{-1}(\mathsf{ct}), \mathbf{g}\right\rangle) + g + \mu}{f'}\end{aligned}$$

As mentioned before the external product, the new ciphertext satisfies $\mathsf{ct}' \in \mathrm{NTRU}_{f,Q}(\mu)$, and the variance of the noise is $\mathsf{Var}(g') \leq Nd\frac{B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{KSK})) + \mathsf{Var}(\mathsf{err}(\mathsf{ct}))$. □

C NTT Algorithm

The NTT is shown in Algorithm 3. We omit the INTT algorithm since it is symmetric.

Algorithm 3 Algorithm for Number Theoretic Transform

Input:

A coefficient vector $\mathbf{a} = (a_0, a_1, ..., a_{N-1})$ for $a(X) \in \mathcal{R}_Q$.

A table ζ_{rev} computed by powers of ζ and stored in bit-reversed order, where $\zeta_{rev}[i] = \zeta^{\text{bit-reverse}(i)} \mod Q$.

Output:

A NTT vector of $\mathbf{a} \in \mathbb{Z}_Q^N$ in bit-reversed order.

```
1: $t = N$
2: for $(m = 1; m < 2N; m = 2m)$ do
3:     $t = t/2$
4:     for $(i = 0; i < m; i++)$ do
5:         $j_1 = 2 \cdot i \cdot t$
6:         $j_2 = j_1 + t - 1$
7:         for $(j = j_1; j \leq j_2; j++)$ do
8:             $U = a_j$
9:             $V = a_{j+t} \cdot \zeta[m+i] \pmod Q$
10:            $a_j = U + V \pmod Q$
11:            $a_{j+t} = U - V \pmod Q$
12:        end for
13:    end for
14: end for
15: **return** NTT($a$).
```

D Automorphism-based Blind Rotation in [XZD+23]

In this Section, we describe the NTRU-based blind rotation that was proposed in [XZD+23] scheme, which output an NTRU ciphertext NTRU$(X^{b+\sum_{i=0}^{n-1} a_i \cdot s_i})$. Note that, the exact gadget decomposition is used during the external products.

E Parameters for Bootstrapping

We give some symbolic and parameters as follows

- λ, Security level;
- t, Plaintext modulus for the LWE sample;
- n, Lattice dimension for the LWE sample;
- q, Ciphertext modulus for the LWE sample;
- N, Ring dimension for NTRU/NGS;
- σ, Standard deviation of Gaussian distribution;
- Q, Ciphertext modulus for the NTRU/NGS sample;
- P, Auxiliary modulus used in the approximate gadget decomposition;
- B, Gadget base for modulus Q used in the external product;
- d, Exact gadget decomposition length for modulus Q;
- d', Approximate gadget decomposition length for modulus Q;
- Q_k, Ciphertext modulus used in LWE key-switching;

Algorithm 4 Automorphism-based blind rotation with NTRU and LWE in [XZD+23]

Input: An LWE ciphertext $\mathsf{ct} = (\mathbf{a}, b = -\langle \mathbf{a}, \mathbf{s}\rangle - \lfloor \frac{q}{t} \rfloor \cdot m + e) \in \mathsf{LWE}^n_{\mathbf{s},q}(m)$, where $q < 2N$. An evaluation key $\mathbf{EVK} = (\mathbf{BRK}^+_i, \mathbf{KSK}_j)$, where $i \in [0, n]$ and $j \in [0, n-1]$.
Output: An LWE sample $\mathsf{ct}' \in \mathrm{LWE}^n_{\mathbf{s},q}(f(m))$.

```
1: for $(i = 0; i < n; i = i + 1)$ do
2:     $w_i = \frac{2N}{q} a_i + 1$
3:     $w'_i = w_i^{-1} \mod 2N$
4: end for
5: Let $w'_n = 1$ and $\mathsf{acc} = X^{\frac{2N}{q} w'_0} \cdot X^{-\frac{2N}{q} b w'_0}$
6: for $(i = 0; i < n; i = i + 1)$ do
7:     $\mathsf{acc} = \mathsf{acc} \odot \mathbf{BRK}^+_i$
8:     if $w_i w'_{i+1} \neq 1$
9:         $\mathsf{acc} = \mathsf{HomAuto}_{w_i w'_{i+1}}(\mathsf{acc}, \mathbf{KSK}_{w_i w'_{i+1}})$
10: $\mathsf{acc} = \mathsf{acc} \odot \mathbf{BRK}^+_n$
11: end for
12: return $\mathsf{ct}'$.
```

- B_k, Gadget base used in LWE key-switching;
- d_k, Exact gadget decomposition length for modulus Q digits;
- ω, Window size used in automorphism-based blind rotation.

F FPGA Parameter and Decryption Failure Rate

To compare with the scheme [YKP22] for FPGA implementation, we adjust the parameters as follows. In order to achieve the same security level as [YKP22] and [VBDV22], we set the value of n to 500. In addition, we set the unrolling factor m to 4 and the length of the approximate gadget decomposition d' to 2. As a result, the number of iterations in the blind rotation using CMux gates is reduced to $500/4 = 125$, while the number of external product executions in each CMux gate is increased to 15. This adjustment for the number of iterations and the length of the approximate gadget decomposition effectively reduces the number of NTTs required in blind rotation, which contributes to improved performance in the FPGA implementation.

Then, we compute the decryption failure rate for this parameter set. For the parameter $n = 500$, $m = 4$, and $d' = 2$, the variance of noise in new CMux gate is

$$\mathsf{Var}(\mathsf{err}(\mathsf{acc})) \leq \frac{5Nd'B^2}{2} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{5NP^2}{8} + \mathsf{Var}(\mathsf{err}(\mathsf{acc})),$$

where $B = 2^6$, and $P = 2^8$. The step is performed $n/4 = 125$ times in blind rotation, thus the variance of noise in blind rotation satisfies

$$\begin{aligned}\mathsf{Var}(\mathsf{err}(\mathsf{acc})) &\leq \frac{5nNd'B^2}{8} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{5nNP^2}{32} + \frac{Nd'B^2}{12} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{NP^2}{24} \\ &\leq \frac{(15n+2)Nd'B^2}{24} \cdot \mathsf{Var}(\mathsf{err}(\mathbf{BRK})) + \frac{(15n+4)NP^2}{96}.\end{aligned}$$

By incorporating the variance $\mathsf{Var}(\mathsf{err}(\mathsf{acc}))$ into Equation 4, we can calculate the variance of noise for bootstrapping. Finally, we can obtain the final decryption failure rate by evaluating the formula for the decryption failure rate with Equation 7, which is approximately $2^{-15.3}$.