Free-XOR Gate Bootstrapping

Chunling Chen[1,2], Xianhui Lu[1,2] ✉ *, Ruida Wang[1,2] ✉, Zhihao Li[3], Xuan Shen[1,2] and Benqiang Wei[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
{chenchunling,luxianhui,wangruida}@iie.ac.cn
[3] Ant Group, Beijing, China

Abstract. The FHEW-like gate bootstrapping framework operates in a 2-bit plaintext space, where logic gates such as NAND, XOR, and AND are implemented by adding two ciphertexts and extracting the most significant bit. However, each gate operation requires bootstrapping with a primary cost of one blind rotation, which is expensive, when processing circuit operations for applications. We propose a novel Free-XOR gate bootstrapping framework based on a single-bit plaintext space, in which the XOR operation is realized by simply adding two ciphertexts, resulting in an almost free computational cost. To form a minimal complete set for logical operations, we design an algorithm for the AND gate within this framework. The AND gate cost of our Free-XOR gate bootstrapping involves two blind rotations. However, by utilizing a single-bit plaintext space to enhance noise tolerance and swapping some operations of the bootstrapping process, we can adopt a more compact parameter setting, which in turn accelerates the speed of blind rotation. We propose an instantiation of the NTRU-based AND gate operation, which requires two blind rotations. Despite the additional rotation, the overall computational cost is marginally lower than the state-of-the-art gate bootstrapping scheme LLW+ [TCHES24], which utilizes only a single blind rotation. In addition, our approach achieves a significant reduction in key size, reducing it to 3.3 times the size of LLW+ [TCHES24].
We ultimately apply our framework to the homomorphic evaluation of symmetric ciphers, in which require a substantial number of XOR operations: 74% for AES and 62.6% for ASCON. The Free-XOR gate bootstrapping framework demonstrates significant advantages in this context. Specifically, the homomorphic computation of AES requires 31 seconds, while ASCON requires 28 seconds, representing an improvement of 1.5× over the Thunderbird implementation [TCHES24] and 4.7× over the work of BPR [TCHES24].

Keywords: Gate Bootstrapping · Free-XOR · Single-bit Plaintext Space · NTRU.

1 Introduction

Fully homomorphic encryption (FHE) schemes enable computing on encrypted data directly, thereby achieving a state where the data remain available but invisible. The first FHE scheme was proposed by Gentry in 2009 [Gen09]. FHE schemes are predominantly based on lattice-based hard problems and ensure their security by incorporating noise into the encryption process. Currently, the most popular FHE schemes are primarily constructed upon the Learning With Errors (LWE), Ring Learning With Errors (RLWE), and NTRU problems [Reg09, LPR10, KF17]. FHE schemes are typically classified into

*Xianhui Lu was supported by Chinese Academy of Sciences Project for Young Scientists in Basic Research [NO.YSBR-035].

three main categories: (1) BFV/BGV [BV11, FV12, BGV14a], which are designed to perform exact arithmetic operations on finite fields; (2) CKKS [CKKS17, CHK$^+$18], which enables approximate arithmetic computations and supports the handling of real and complex numbers; and (3) FHEW/TFHE/NTRU-based schemes [DM15a, CGGI16, CGGI17, CGGI20, BIP$^+$22a], which are friendly for Bit-Wise boolean evaluation.

In FHE, once noise accumulates to a certain threshold, it is necessary to perform a bootstrapping operation to ensure the validity and accuracy of the calculations. Bootstrapping, i.e. computing the decryption circuit homomorphically, is the only way to achieve full homomorphism. Gate bootstrapping is computing a gate operation while bootstrapping [DM15a, CGGI16, CGGI17]. The gate bootstrapping (GBS) method is considered the most effective currently.

The plaintext space of existing gate bootstrapping methods consists of 2 bits [DM15a, CGGI16, CGGI17, CGGI20]. For AND ($\wedge$) gate bootstrapping , the first step is to add two ciphertexts, with the most significant bit representing the result of $(x \wedge y)$. The most significant bit of the LWE ciphertext is then derived through blind rotation and sample extraction. Subsequently, the modulus and dimension are reduced using modulus switching and key switching, respectively. For the current XOR gate bootstrapping, two addition operations are required to put the result of $(x \vee y)$ in the most significant bit, and then blind rotation and sample extraction are used to obtain the most significant bit. Currently, the parameters used for gate bootstrapping are mainly designed for AND and NAND gates. However, when these parameters are applied to XOR gate bootstrapping, the decryption failure rate will increases. In addition, we observe that for ciphertexts in the single-bit plaintext space, addition directly corresponds to performing an XOR operation, which comes almost no additional cost. This property can be described as Free-XOR. Therefore, our goal is to design an efficient Free-XOR gate bootstrapping framework.

Finally, we apply our Free-XOR gate bootstrapping framework to homomorphic evaluation of symmetric ciphers, which can solve the size expansion problem of homomorphic ciphertexts in encryption frameworks. The central premise involves employing symmetric encryption for data transmission, while the server pre-computes the FHE ciphertext before evaluation. In symmetric cryptographic algorithms, the XOR operation serves as a fundamental component, extensively employed in key scheduling and data encryption processes. However, current gate bootstrapping techniques necessitate the refreshing of noise in the corresponding ciphertext for each computation of the gate operation. This requirement substantially elevates computational overhead, particularly when addressing intricate symmetric encryption schemes. A significant advantage of homomorphic computation lies in its capacity to perform operations directly on encrypted data without necessitating decryption. This feature facilitates the integration of homomorphic computation within symmetric cryptography. By executing XOR operations on ciphertexts, one can effectively enable the parallel processing of encrypted data while maintaining its security. Therefore, establishing a Free-XOR gate bootstrapping framework is critically important for the efficient homomorphic evaluation of symmetric cryptographic algorithms.

1.1 Contribution

Our primary contribution is the proposal of a novel Free-XOR gate bootstrapping framework, which leverages the near-zero cost of adding LWE ciphertexts to achieve XOR operations in the single-bit plaintext. Additionally, we present the construction of AND gate, along with the instantiation and implementation based on NTRU in single-bit plaintex space. Furthermore, our scheme is applied to the homomorphic evaluation of symmetric ciphers, including AES and ASCON. Specifically, our contributions are as follows.

- **A Novel Free-XOR Gate Bootstrapping Framework.** We propose a novel Free-XOR gate bootstrapping framework that efficiently leverages ciphertext in the single-bit plaintext space for nearly cost-free XOR evaluation. We present the workflow for constructing AND gates of our framework. Furthermore, the Free-XOR gate bootstrapping framework is categorized into two types-GGBS and IGBS-requiring 2 and 1 blind rotations, respectively, as shown in Figure 1. The Free-XOR gate bootstrapping framework is the first efficient method based on single-bit plaintext space, featuring specific efficiency parameters and practical implementation.

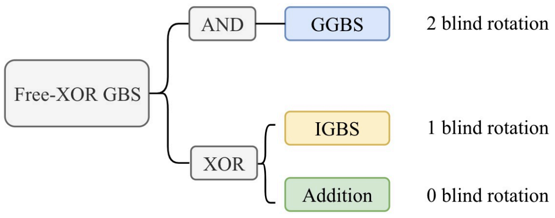


Figure 1: Classification of Free-XOR Gate Bootstrapping Framework.

- **More Compact and Faster NTRU-Based Instantiation of AND Gate.** We propose an NTRU-based instantiation of a single-bit plaintext-space AND gate and implement GGBS for AND gates with improved noise management and more compact parameters. This results in GGBS, which requires two blind rotations, being slightly faster than the state-of-the-art gate bootstrapping scheme that requires only one blind rotation [LLW+24]. Furthermoreour implementation demonstrates that our GGBS performance is 1.3× better than TFHE-rs and 1.7× faster than TFHE-pp, further validating the effectiveness of our optimization strategy in advancing homomorphic encryption. Additionally, our bootstrapped key reduces the key size by 3.3 times compared to the state-of-the-art scheme of LLW [LLW+24].

- **Efficient Homomorphic Evaluation in Symmetric Ciphers.** By adjusting parameters, our design enables an XOR-free computation with a depth of 7 using our Free-XOR gate bootstrapping. we apply our gate bootstrapping framework in single-bit plaintext space to homomorphic evaluation of symmetric ciphers, specifically AES and ASCON. Moreover, our algorithm has greater advantages when applied to lightweight symmetric ciphers. The homomorphic evaluation of AES takes 31 seconds, with a performance improvement of 1.5×, and the homomorphic evaluation of ASCON takes 28 seconds, with a performance improvement of 4.7×. Our Free-XOR gate bootstrapping framework provides a lightweight, symmetric cryptography-friendly solution for homomorphic computations. For symmetric cryptographic systems with constrained S-box computation depth, this framework significantly enhances performance.

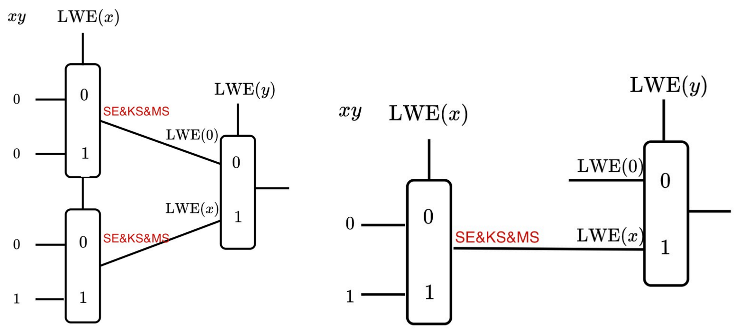
Table 1: Comparison of running time between our NTRU-Based instantiation and the latest NTRU-Based scheme. BR, Size, Opt, Class Refer to Blind Rotation, Key Size, Optimization and Classification, respectively.

Gate	NTRU-based [LLW+24]				Our Free-XOR GBS				
	BR	Time(ms)	Size(MB)	Opt(ms)	Class	BR	Time(ms)	Size(MB)	Opt(ms)
AND	1	40	7.1	3.8	GGBS	2	35	**3.2**(**3.3**×)	**3.7**
XOR	1	40	7.1	3.8	IGBS	1	17	**3.2**(**3.3**×)	**1.8**(**2.1**×)
					Addition	0	0	**0**	**0**

1.2 Technique Overview

Table 2: Truth Table for $(x \wedge y)$

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1



(a) Framework (b) Optimization Framework

Figure 2: Framework for Free-XOR gate bootstrapping, where KS, MS, BR and SE stands for key switching, modulus switching, blind rotation, sample extraction, respectively.

- **Construction of AND Gate.** In our Free-XOR gate bootstrapping, the XOR operation is almost free. For completeness, we show the construction of the AND gate. The construction of AND gates is inspired by the tree lookup table. Initially, we take the output of the truth table $(x \wedge y)$ in Table 2 as a new input to construct a two-layer structure, as illustrated in Figure 2. In the first layer, x serves as the control bit for blind rotation. When $x = 0$, the output is $\mathsf{LWE}(0)$ and $\mathsf{LWE}(0)$; when $x = 1$, the output is $\mathsf{LWE}(0)$ and $\mathsf{LWE}(1)$. Because the output of the first layer is always $\mathsf{LWE}(0)$ and $\mathsf{LWE}(x)$, we can simplify Figure 2a to two blind rotation in Figure 2b.

  Specifically, our AND algorithm first refreshes the noise of $\mathsf{LWE}^N_{\boldsymbol{s},Q}(\frac{Q}{2}x)$ by bootstrapping, while converting the LWE ciphertext encoding from $\frac{Q}{2}$ to $\frac{Q}{4}$. Then, $\mathsf{LWE}^N_{\boldsymbol{s},Q}(0)$ and $\mathsf{LWE}^N_{\boldsymbol{s},Q}(\frac{Q}{2}x)$ are packed into an RLWE or NTRU ciphertext, serving as the test polynomial for bootstrapping $\mathsf{LWE}^N_{\boldsymbol{s},Q}(\frac{Q}{2}y)$ and setting the accumulator. After completing the blind rotation, the result in the accumulator is added to $\mathsf{LWE}^N_{\boldsymbol{s},Q}(\frac{Q}{4}x)$, yielding the final computation of $\mathsf{LWE}^N_{\boldsymbol{s},Q}(\frac{Q}{2}xy)$, as illustrated in the Figure 3.

- **Efficient NTRU-based instantiation of the AND gate.** Existing NTRU-based gate bootstrapping suffers from low efficiency, primarily due to inadequate noise control. We implement an efficient NTRU-based instantiation of the AND gate operation, primarily enabled by effective noise management. (1) Original gate bootstrapping framework are typically based on 2-bit adders. In our work, we utilize a 1-bit plaintext space, which provides enhanced flexibility for noise accommodation, allowing for more compact parameter configurations. (2) we improve noise control by adjusting some operations of gate bootstrpping process. Instead of directly bootstrapping under the modulus q and dimension n, we first operate under the larger

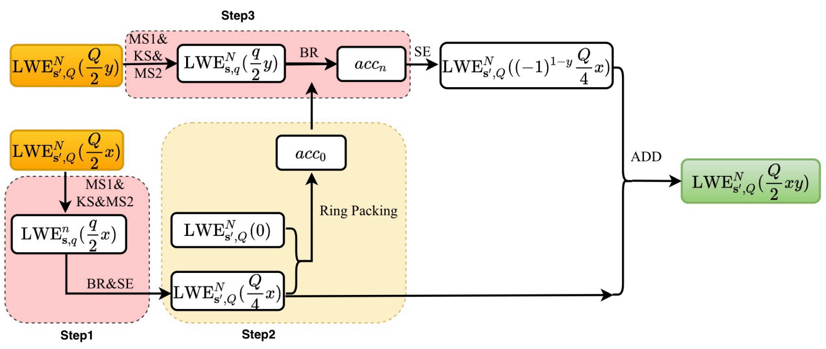

Step3
MS1& KS& MS2
BR
SE
$\mathrm{LWE}^N_{s',Q}(\frac{Q}{2}y)$
$\mathrm{LWE}^N_{s,q}(\frac{q}{2}y)$
acc_n
$\mathrm{LWE}^N_{s',Q}((-1)^{1-y}\frac{Q}{4}x)$
$\mathrm{LWE}^N_{s',Q}(\frac{Q}{2}x)$
MS1& KS&MS2
$\mathrm{LWE}^n_{s,q}(\frac{q}{2}x)$
BR&SE
acc_0
Ring Packing
$\mathrm{LWE}^N_{s',Q}(0)$
$\mathrm{LWE}^N_{s',Q}(\frac{Q}{4}x)$
ADD
$\mathrm{LWE}^N_{s',Q}(\frac{Q}{2}xy)$
Step1
Step2

Figure 3: AND gate construction workflow for Free-XOR gate bootstrapping, where KS, MS, BR and SE stands for key switching, modulus switching, blind rotation, sample extraction, respectively.

modulus Q and dimension N, switching to the smaller modulus q and dimension n only when bootstrapping is needed. (3) With the same security level, we increase the noise in the NTRU key and then increase the NTRU modulus to achieve more effective noise control. (4) Better noise management is employed in our Packing algorithm.

- **Packing.** Before the second blind rotation, the test polynomial needs to be set. Since both $\mathsf{LWE}(0)$ and $\mathsf{LWE}(x)$ are ciphertexts, they must be packed and set as the test polynomial $\mathsf{TestP}(x)$, and then the initial accumulator is set. For the AND gate, due to the special properties of $\mathsf{LWE}(0)$, only $\mathsf{LWE}(x)$ needs to be packed. Since we use NTRU-based bootstrapping, we need to convert the LWE ciphertext to NTRU ciphertext. When we perform key switching, we encrypt $\boldsymbol{s}[i] \cdot (1 + X + \cdots + X^{N-1})$ as the key instead of encrypting $\boldsymbol{s}[i]$, and we encrypt $f \cdot (1 + \cdots + X^{N-1})$ as another KSK to achieve packing into an NTRU ciphertext.

- **Efficient Implementation Utilizing Key Unrolling and AVX512 Acceleration.** We employ key unrolling techniques to enhance the performance of blind rotation within the FGBS algorithm. Additionally, we optimize the core components of blind rotation, including approximate decomposition, NTT, and Hadamard multiplication, leveraging AVX instructions to enable parallel computation. Notably, using AVX-512 instructions allows the NTT to achieve 16 parallel operations within a 32-bit word length.

1.3 Related Work

- **NTRU-based Gate Bootstrapping.** Bonte et al. introduced a bootstrapping scheme [BIP$^+$22b] where the NTRU ciphertext is structured as $g/f + m$. Although this ciphertext design facilitates outer product operations with GSN ciphertexts, it does not support efficient key-switching or ring automorphism operations, which presents a significant limitation. Addressing this inefficiency, Xiang et al. [XZD$^+$23] proposed a novel bootstrapping algorithm in 2023 that is grounded in automorphism techniques. They restructured the NTRU ciphertext to $(g + m)/f$, allowing the conversion of the NTRU ciphertext into an LWE ciphertext by sample extraction. This refinement effectively bypasses the complex and resource-intensive key switching method proposed by Bonte et al. Moreover, Xiang et al.'s ciphertext supports key operations such as NTRU key switching and automorphism, further improving computational efficiency.

- **FHEW-like Algorithms with Single-bit Plaintext Space.** Homomorphic computation based on single-bit plaintext space currently employs two primary methods. The first method is the LWE multiplication proposed by [CLOT21], which utilizes the LWE tensor of the BGV/ BFV/ CKKS to facilitate the multiplication of two ciphertexts. However, the noise resulting from this multiplication is $O(N^3)$, which is substantial. In addition, the paper does not provide specific parameter settings. Our exploration revealed that this method requires large parameters and exhibits low efficiency. The second method involves multiplying single-bit RLWE and RGSW ciphertexts; however, after a single computation, RGSW must be converted back into RLWE form to continue operations, which requires circuit bootstrapping [CGGI17, CGGI20, WWL$^+$24a]. Circuit bootstrapping is a hierarchical fully homomorphic encryption (FHE) approach that organizes the operations to be performed into a table, calculating them via table lookups. This method also requires large parameters and demonstrates low efficiency. In Eurocrypt2024, Wang et al. [WWL$^+$24b] proposed a more compact and faster circuit bootstrapping scheme, yet its efficiency remains significantly lower than that of gate bootstrapping.

1.4 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 introduce the Free-XOR gate bootstrapping framework, the GGBS construction process of AND, give examples based on NTRU, and propose a new packing method. In Section 4, we present our noise analysis and achieved performance. In Section 5, we demonstrate some optimization techniques for improving blind rotation efficiency. Finally, we present the application of our framework in the transciphering scenario in Section 6. Finally, in section 7, we summarize our work

2 Preliminaries

2.1 Notation

We denote as $\mathbb{Z}_q$ the set of integers modulo q. The $2N$-th cyclotomic ring, denoted as $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$, where N is a power of 2. The quotient ring is $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1)$, which represents $\mathcal{R}/q\mathcal{R}$. Elements in $\mathbb{Z}_q$ or $\mathcal{R}_q$ are denoted by regular letters, such as $a \in \mathbb{Z}_q$ or $\mathcal{R}_q$. Vectors in $\mathbb{Z}_q$ or $\mathcal{R}_q$ are represented by bold letters, such as $\boldsymbol{a}$, with $\boldsymbol{a}[i]$ indicating the i-th component. For $a \in \mathcal{R}_q$, a_i denotes the coefficient of the i-th term in a. The floor, ceiling, and rounding functions are denoted by $\lfloor\cdot\rfloor$, $\lceil\cdot\rceil$ and $\lfloor\cdot\rceil$ respectively. The l_2 and l_∞ norms are represented by $\|\cdot\|_2$ and $\|\cdot\|_\infty$, respectively. Sampling a from the discrete Gaussian distribution $\mathcal{D}_{\mu,\sigma}$ with mean μ and standard deviation σ is indicated by $a \leftarrow \mathcal{D}_{\mu,\sigma}$. Inner products and outer products are denoted by $\langle\cdot,\cdot\rangle$ and $\boxdot$, respectively. Additionally, $\odot$ can signify gadget product. We denote the error of a ciphertext ct by $\mathsf{Err}(ct)$, and the variance of the error by $\mathsf{Var}(\mathsf{Err}(ct))$.

2.2 Hard Problems

In this subsection, we revisit the foundational hard problems upon which FHE schemes are constructed: Learning with Errors (LWE) problem [Reg09], Ring Learning with Errors (RLWE) problem [LPR10], and the N-th Degree Truncated Polynomial Ring Units (NTRU) problem [KF17].

Decisional LWE Problem [Reg09]. For security parameter λ, let $n = n(\lambda)$ be the integer dimension, $q = q(\lambda) \geq 2$ be an integer, and let $\chi = \chi(\lambda)$ be a distribution over

$\mathbb{Z}$ (usually a discrete Gaussian distribution). The $\mathsf{LWE}_{(n,q,\chi)}$ problem is to distinguish between the following two distributions:

- Sample $(\boldsymbol{a}, b)$ uniformly from $\mathbb{Z}_q^{n+1}$.
- Sample $\boldsymbol{a}$ uniformly from $\mathbb{Z}_q^n$, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and sample e from χ. Set $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e$.

Decisional RLWE Problem [LPR10]**.** Given security parameter λ, $N = N(\lambda)$ and $q = q(\lambda)$ are positive integers. Let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$, and $\chi = \chi(\lambda)$ be a nosie distribution over $\mathcal{R}$. The $\mathsf{RLWE}_{(N,q,\chi)}$ problem is to distinguish between the following two distributions:

- Sample (a, b) uniformly from $\mathcal{R}_q^2$.
- Sample a uniformly from $\mathcal{R}_q$, $s \leftarrow \mathcal{R}_q$, and sample e from χ. Set $b = a \cdot s + e$.

Decisional NTRU Problem [KF17]**.** Given security parameter λ, $N = N(\lambda)$ and $q = q(\lambda)$ are positive integers. Let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$, and $\chi = \chi(\lambda)$ be a noise distribution over $\mathcal{R}$. The $\mathsf{NTRU}_{(d,q,\chi)}$ problem is to distinguish between the following two distributions:

- Sample polynomial c uniformly from $\mathcal{R}_q$.
- Sample $f \in \mathcal{R}_q$ satisfying f has an inverse in $\mathcal{R}_q$, $g \leftarrow \chi$ is a noise polynomial. Set $a = g/f \in \mathcal{R}_q$.

2.3 Message Encoding

In this paper, we utilize most significant bit (MSB) encoding, with the corresponding encoding and decoding functions defined as follows:

$$\text{Encode}: \ \gamma = \left\lfloor \frac{q}{t} \right\rfloor \cdot m + e, \qquad \text{Decode}: \ m = \left\lfloor \frac{t}{q} \cdot \gamma \right\rceil \bmod t.$$

In this context, t represents the plaintext modulus, and q denotes the ciphertext modulus.

Remark 1. *We use superscripts and subscripts to define the parameters of the ciphertext, such as* $\mathsf{LWE}^n_{\boldsymbol{s},q}(\frac{q}{t}m), \mathsf{RLWE}^n_{s,q}(\frac{q}{t}m)$*. In some cases, we may omit encoding for brevity like* $\mathsf{LWE}^n_{\boldsymbol{s},q}(m)$*.*

2.4 Approximate Gadget Decomposition

Gadget decomposition is a key technique to control noise growth in FHE. Approximate Gadget Decomposition can improve computational efficiency. Given a gadget vector $\boldsymbol{g} = (g_0, g_1, \cdots, g_{\ell-1})$, the Gadget decomposition for a ring element $t \in \mathcal{R}_t$ is to find a set of small elements $(t_0, t_1, \cdots, t_{\ell-1})$ such that $\sum_{i=0}^{\ell-1} t_i \cdot \boldsymbol{g}[i] = t$. The approximate Gadget vector for the approximate Gadget decomposition is $\boldsymbol{g} = (\lceil q/B^\ell \rceil, \lceil q/B^\ell \rceil \cdot B, \cdots, \lceil q/B^\ell \rceil \cdot B^{\ell-1})$, where $B^\ell < q$. For polynomial h, its approximate decomposition is a set of polynomials $h_0, h_1, \cdots, h_{\ell-1}$, where the coefficients are less than B. Define $\epsilon = \left\| \sum_{i=0}^{\ell-1} h_i \boldsymbol{g}[i] - h \right\|_\infty$, satisfying $\epsilon \leq \frac{1}{2} \left\lceil \frac{q}{B^\ell} \right\rceil$.

2.5 NTRU-based GSW-like Encryption

This section provides a brief overview of the NTRU encryption scheme and introduces several fundamental homomorphic building blocks.

Definition 1 (**NTRU Ciphertext**)**.** An NTRU ciphertext is expressed as

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

where g, f are the error ring polynomial sampled from a sub-Gaussian distribution and f has a standard deviation of $\sigma(f) = 3.5$.

The NTRU ciphertext structure inherently supports both homomorphic addition and homomorphic scalar multiplication. For two ciphertexts, ct_x and ct_y, encrypted with the same private key f, their homomorphic addition is given by:

$$\mathsf{ct}_x + \mathsf{ct}_y = \frac{g_x + g_y + (\mu_x + \mu_y)}{f} \in \mathcal{R}_Q.$$

Furthermore, scalar multiplication of a ciphertext by a scalar z can be directly performed through polynomial multiplication. For a ciphertext ct_x and a scalar z, the resulting scalar multiplication is given by:

$$z \cdot \mathrm{ct}_x = \frac{z \cdot (g_x + \mu_x)}{f} \in \mathcal{R}_Q.$$

Definition 2 (**NGS Ciphertext**)**.** For the gadget vector $\boldsymbol{g}$, we define $\mathsf{NTRU}_f^{'}$ and NGS_f as follows:

$$\mathsf{NTRU}'_f(m) := \left(\mathsf{NTRU}_f\left(\boldsymbol{g}[0]\cdot m\right), \mathsf{NTRU}_f\left(\boldsymbol{g}[1]\cdot m\right), \cdots, \mathsf{NTRU}_f\left(\boldsymbol{g}[\ell-1]\cdot m\right)\right) \in \mathcal{R}_Q^{\ell}.$$
$$\mathsf{NGS}_f(m) := \left(\mathsf{NTRU}'_f(f\cdot m)\right) \in \mathcal{R}_Q^{\ell}.$$

Definition 3 (**External Product**)**.** For a polynomial $t \in \mathcal{R}_Q$, the gadget multiplication of $\mathsf{RLWE}'(m)$ is given by:

$$\odot : \mathcal{R}_Q \times \mathsf{NTRU}' \rightarrow \mathsf{NTRU}$$

$$\begin{aligned} t \odot \mathsf{NTRU}'_f(m) &= \left\langle \left(t_0, \cdots, t_{\ell-1}\right), \left(\mathsf{NTRU}_f\left(\boldsymbol{g}[0]\cdot m\right), \cdots \mathsf{NTRU}_f\left(\boldsymbol{g}[\ell-1]\cdot m\right)\right)\right\rangle \\ &= \sum_{i=0}^{\ell-1} t_i \cdot \mathsf{NTRU}_f\left(\boldsymbol{g}[i]\cdot m\right) = \mathsf{NTRU}_f\left(\sum_{i=0}^{\ell-1} \boldsymbol{g}[i]\cdot t_i \cdot m\right) \\ &= \mathsf{NTRU}_f(t\cdot m) \in \mathcal{R}_Q \end{aligned}$$

The multiplication of the ciphertexts $\mathsf{NTRU}(m_1)$ and $\mathsf{NGS}(m_2)$ is given by:

$$\boxdot : \mathsf{NTRU} \times \mathsf{NGS} \rightarrow \mathsf{NTRU},$$

$$\begin{aligned} \mathsf{NTRU}_f\left(m_1\right) \boxdot \mathsf{NGS}_f\left(m_2\right) &= \left(\left(g+m_1\right)/f\right) \odot \left(\mathsf{NTRU}'_f\left(f\cdot m_2\right)\right) \\ &= \mathsf{NTRU}_f\left(\left(\left(g+m_1\right)/f\right)\cdot f m_2\right) \\ &= \mathsf{NTRU}_f\left(m_1\cdot m_2 + g\cdot m_2\right) \in \mathcal{R}_Q \end{aligned}$$

Remark 2. *Here, we use the approximate gadget decomposition to instantiate the external product as shown in scheme [LLW+24]. The approximate gadget vector* $\mathbf{g} = \left(B_0, \ldots, B_g^{\ell-1}\right)$. *The error introduced by* $\odot$ *is* $\frac{1}{12}N\ell B_g^2\sigma^2$, *where* σ^2 *is error variance. The error variance of* $t \odot \mathsf{NTRU}'_f(m)$ *is* $\frac{1}{12}N\ell B_g^2\sigma^2 + \frac{1}{6}N\epsilon^2$,*where* $\epsilon = \frac{1}{2}\left\lceil \frac{q}{B^\ell}\right\rceil$.

2.6 Gate Bootstrapping

Gate bootstrapping (GBS) reduces noise while homomorphically evaluating binary gate operations such as AND, NAND, OR and XOR. The current basic architecture of gate bootstrapping is proposed by Ducas and Mcciancio [DM15a], As shown in Figure 4. In the method described by [DM15a], the plaintext space consists of 2 bits, meaning $t = 4$. By performing homomorphic addition on ct_x and ct_y (that is, $ct = \mathrm{ct}_x + \mathrm{ct}_y$), we can add the plaintexts x and y. The resulting values after addition are in the set $\{0, 1, 2\}$. The MSB of the ciphertext ct is then extracted through blind rotation, resulting in the ciphertext for $(x \wedge y)$.

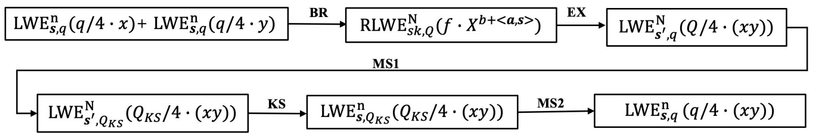

$\mathrm{LWE}^{n}_{s,q}(q/4 \cdot x)$+ $\mathrm{LWE}^{n}_{s,q}(q/4 \cdot y)$
BR
$\mathrm{RLWE}^{N}_{sk,Q}(f \cdot X^{b+<a,s>})$
EX
$\mathrm{LWE}^{N}_{s',q}(Q/4 \cdot (xy))$
MS1
$\mathrm{LWE}^{N}_{s',Q_{KS}}(Q_{KS}/4 \cdot (xy))$
KS
$\mathrm{LWE}^{n}_{s,Q_{KS}}(Q_{KS}/4 \cdot (xy))$
MS2
$\mathrm{LWE}^{n}_{s,q}(q/4 \cdot (xy))$

Figure 4: FHEW-like Gate Bootstrapping Procedure for AND gate [DM15a].

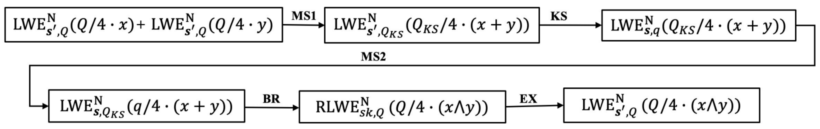

$\mathrm{LWE}^{N}_{s',Q}(Q/4 \cdot x)$+ $\mathrm{LWE}^{N}_{s',Q}(Q/4 \cdot y)$
MS1
$\mathrm{LWE}^{N}_{s',Q_{KS}}(Q_{KS}/4 \cdot (x+y))$
KS
$\mathrm{LWE}^{N}_{s,q}(Q_{KS}/4 \cdot (x+y))$
MS2
$\mathrm{LWE}^{N}_{s,Q_{KS}}(q/4 \cdot (x+y))$
BR
$\mathrm{RLWE}^{N}_{sk,Q}(Q/4 \cdot (x \wedge y))$
EX
$\mathrm{LWE}^{N}_{s',Q}(Q/4 \cdot (x \wedge y))$

Figure 5: AND Gate Bootstrapping procedure of FHEW scheme, starting from $\mathsf{LWE}_{Q,z}$ and switch to $\mathsf{LWE}_{q,s}$ before blind rotation[LMK+23].

Lee et al. [LMK+23] achieved a reduction in both noise and number of key switching by rearranging the sequence of operations within the gate bootstrapping procedure. Specifically, they begin with a ciphertext encrypted under a bigger modulus Q instead of the smaller modulus q, and perform modulus switching to q immediately before blind rotation (see Figure 5).

2.6.1 Sample Extraction

We show that the sample extraction technique [XZD+23] can extract the LWE ciphertext from NTRU ciphertext. They treat the NTRU ciphertext $\mathsf{ct} = (g + m)/f$ as an RLWE ciphertext $(-\mathsf{ct}, 0)$, and the decryption process with the secret key f is

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

Then the sample extraction is defined as

$$\mathsf{SampleExtraction}(\mathsf{ct}) = (-\mathsf{ct}_0, \mathsf{ct}_{N-1}, \mathsf{ct}_{N-2}, ..., \mathsf{ct}_1, 0) \in \mathsf{LWE}^{N}_{\mathsf{Coefs}(f),Q}(m_0).$$

2.6.2 LWE-to-LWE Key Switching

The LWE-to-LWE key-switching process [DM15a] can adjust the LWE dimension without altering the underlying message. The process is described in detail below. Let $\mathsf{ct} = \mathsf{LWE}^{N}_{\boldsymbol{z},Q_{\mathrm{KS}}}(m) \in \mathbb{Z}^{N}_{Q_{\mathrm{KS}}}$ be an LWE ciphertext under $\boldsymbol{z}$. LWE-to-LWE key-switching is to switch the secret from $\boldsymbol{z} \in \mathbb{Z}^N$ to $\boldsymbol{s} \in \mathbb{Z}^n$. B_{KS} is the base. The key-switching key consists of $\mathsf{LtLKSK}_{i,j,v} \in \mathsf{LWE}^{n}_{\boldsymbol{s},Q_{\mathrm{KS}}}(v\boldsymbol{z}[i]B^{j}_{\mathrm{KS}})$, where $v \in \{0, \ldots, B_{\mathrm{KS}} - 1\}$, for all $0 \le i \le N - 1$, $0 \le j \le \ell_{\mathrm{KS}} - 1$ and $\ell_{\mathrm{KS}} = \lceil \log_{B_{\mathrm{KS}}} Q_{\mathrm{KS}} \rceil$. For a ciphertext $\mathsf{ct} = (\boldsymbol{a}, b) \in \mathsf{LWE}^{N}_{\boldsymbol{z},Q_{\mathrm{KS}}}(m)$, the key switching procedure first decomposes each element under B_{KS}, i.e. $\boldsymbol{a}[i] = \sum_j a_{i,j} B_{\text{KS}}^j$, and outputs

$$\begin{aligned}\mathsf{ct}' &= \mathsf{LtLKeySwitch}(\mathsf{ct}) \\ &= (\mathbf{0}, b) + \sum_{i,j} \mathsf{LtLKSK}_{i,j,a_{i,j}} \bmod Q_{\text{KS}} \\ &= (\boldsymbol{a}', b') \in \mathsf{LWE}^n_{\boldsymbol{s},Q_{\text{KS}}}(m).\end{aligned}$$

The error variance of the result of the LWE-to-LWE key switching is bounded by $\sigma^2_{ct'} \leq \sigma^2_{ct} + N\ell_{\text{KS}}\sigma^2_{in}$, where σ^2_{in} is the error variance of key-switching key.

2.6.3 Modulus Switching

The modulus switching technique adjusts the modulus of a ciphertext [BGV14b, DM15b]. Given a ciphertext $\mathsf{ct} = (\boldsymbol{a}, b) \in \mathsf{LWE}^n_{\boldsymbol{s},Q}(m)$, the algorithm outputs a new ciphertext as

$$\mathsf{ct}' = \mathsf{ModSwitch}(\mathsf{ct}) = (\lfloor \frac{q}{Q} \cdot \boldsymbol{a} \rceil, \lfloor \frac{q}{Q} \cdot b \rceil) \in \mathsf{LWE}^n_{\boldsymbol{s},q}(m).$$

The variance of noise satisfies $\sigma^2_{ct'} \leq (\frac{q}{Q})^2 \cdot \sigma^2_{ct} + \frac{||\mathbf{s}||_2^2+1}{12}$.

Algorithm 1 NTRU Blind Rotation $\mathsf{BlindRotate}(\mathsf{acc}, \mathsf{acc} \cdot X^{\boldsymbol{a}[i]}, \mathsf{bsk})$

Input:
An LWE sample $\mathsf{ct} = (\boldsymbol{a}, b) \in \mathsf{LWE}^{n+1}_{\mathbf{s},q}(m)$, where $q|2N$.
A bootstrapping key $\mathsf{bsk}_i : \{\mathsf{GSN}_{f,Q}(\boldsymbol{s}[i])\}$, for $i = \in [0, n-1]$.
$\mathsf{acc} = X^{-b\cdot\frac{2N}{q}} \cdot \mathsf{testP}(X) \in \mathcal{R}_Q$.
Output:
An NTRU ciphertext $\mathsf{acc} \in \mathsf{NTRU}_{f,Q}(\mathsf{testP}(X) \cdot X^{(\frac{2N}{q})\cdot(-b+\langle \boldsymbol{a},\boldsymbol{s}\rangle)})$

```
1: for $i = 0$ to $n-1$ do
2:         $\mathsf{acc} = \mathsf{acc} + \big(\mathsf{acc} \cdot X^{\boldsymbol{a}[i]} - \mathsf{acc}\big) \boxdot \mathsf{bsk}_i$
3: end for
4: return $\mathsf{acc}$.
```

2.6.4 Blind Rotation

Blind rotation is the core operation in bootstrapping and also the most time-consuming part. Currently, blind rotation is primarily based on AP [ASP14, Per21, DM15a], GINX [CGGI16, CGGI17, CGGI20], the automorphisms by Lee et al. [LMK+23], and NTRU by Xiang et al. [XZD+23]. The blind rotation based on NTRU is detailed in Algorithm 2.

3 Framework of Free-XOR Gate Bootstrapping

We choose a single-bit plaintext space, where the XOR operation is performed by simply adding two ciphertexts, requiring almost no computational costhence the term Free-XOR. Building on this concept, we propose a Free-XOR gate bootstrapping framework, which supports various logical operations, including XOR, XNOR, OR, NAND, AND, and NOR gates. Since the AND and XOR gates together form the minimal complete set of logical operations, we will introduce the construction of the AND gate in this section. Furthermore, we present a specific instantiation of the AND gate operation based on the NTRU.

3.1 Construction of the AND Gate

Our Free-XOR gate bootstrap has two components, as shown in Figure 5: General Gate Bootstrap (GGBS) and Identity Gate Bootstrap (IGBS), with the main costs being 2 and 1 blind rotations, respectively.

3.1.1 Workflow of GBBS

We divide the GGBS of $x \wedge y$ into three steps. In this section, we outline the construction framework of GGBS as shown in Figure 3.

Bootstrapping of $\mathsf{LWE}(\frac{Q}{2}x)$. This process is an identity bootstrapping, primarily aimed at refreshing the noise in $\mathsf{LWE}(\frac{Q}{2}x)$ through bootstrapping, while remaining the result unchanged.

Packing. After refreshing the noise of $\mathsf{LWE}(\frac{Q}{2}x)$, $\mathsf{LWE}(0)$ and $\mathsf{LWE}\left(\frac{Q}{2}x\right)$ need to be packed into a single RLWE ciphertext or an NTRU ciphertext as the test polynomial. Considering that the test polynomial must satisfy negacyclicity, the encoding needs to be converted from $\frac{Q}{2}$ to $\frac{Q}{4}$. Then, the initial accumulator is set accordingly.

FHEW-like Bootstrapping the ciphertext of a Single-Bit y. The Bootstrapping the Ciphertext of a single-bit y is determined by evaluating the test polynomial. Specifically, if $y = 1$, the output yields $\mathsf{LWE}(\frac{Q}{2}x)$; conversely, if $y = 0$, the output results in $\mathsf{LWE}(0)$. This result represents the AND of $\mathsf{LWE}(\frac{Q}{2}x)$ and $\mathsf{LWE}(\frac{Q}{2}y)$.

3.2 Instantiation of NTRU-based Free-XOR Gate Bootstrapping

In this section, we present an NTRU-based instantiation of the Free-XOR gate bootstrapping, specifically demonstrating the GGBS gate bootstrapping, along with the corresponding parameter selections. We divide the NTRU-based Free-XOR gate bootstrapping process into four distinct steps. In this instantiation, we set $q = 2N$.

Step1: NTRU-based Bootstrapping of $\mathsf{LWE}^N_{\boldsymbol{s}',Q}(x)$. We first bootstrap a single-bit x ciphertext $\mathsf{ct}_x = \mathsf{LWE}^N_{\boldsymbol{s}',Q}(x)$ to reduce its noise. This process starts with modulus switching and key switching operations to convert the LWE ciphertext from modulus Q to modulus q, reducing the dimension from N to n.

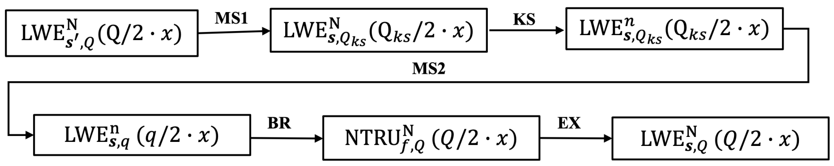

$\mathsf{LWE}^N_{s',Q}(Q/2 \cdot x)$
MS1
$\mathsf{LWE}^N_{s,Q_{ks}}(Q_{ks}/2 \cdot x)$
KS
$\mathsf{LWE}^n_{s,Q_{ks}}(Q_{ks}/2 \cdot x)$
MS2
$\mathsf{LWE}^n_{s,q}(q/2 \cdot x)$
BR
$\mathsf{NTRU}^N_{f,Q}(Q/2 \cdot x)$
EX
$\mathsf{LWE}^N_{s,Q}(Q/2 \cdot x)$

Figure 6: The Procedure for NTRU-based Identical Bootstrapping (IGBS) of $\mathsf{LWE}_{\boldsymbol{s}',Q}(x)$.

Specifically, the initial value of the accumulator is set to

$$\mathsf{acc} = X^{-b} \cdot \mathsf{testP}(X) \in \mathcal{R}_Q, \text{where}\, \mathsf{testP}(X) = \sum_{i=0}^{N-1} 2 \cdot \left\lfloor \frac{q}{2} \cdot i \right\rceil \cdot X^i \cdot X^{\frac{N}{2}}.$$

It should be noted that the test polynomial must satisfy the negacyclivity property, which means $\mathsf{testP}(X + N/2) = -\mathsf{testP}(X)$.

For a binary LWE secret key $\boldsymbol{s} \in \{0, 1\}^n$, the process for generating the bootstrapping key is described as follows [MP21, LLW+24]:

$$\mathsf{bsk} = \begin{cases} \mathsf{bsk}_i = \mathsf{NGS}_{f,Q}(0), & \text{if } (\boldsymbol{s}[i] = 0); \\ \mathsf{bsk}_i = \mathsf{NGS}_{f,Q}(1), & \text{if } (\boldsymbol{s}[i] = 1). \end{cases}$$

Next, followed by n iterative operations as

$$\mathsf{acc} = \mathsf{acc} + \Big(\mathsf{acc} \cdot X^{\boldsymbol{a}[i]} - \mathsf{acc}\Big) \boxdot \mathsf{bsk}_i, (0 \leq i \leq n-1). \tag{1}$$

Upon completion of n iterations, corresponding to the computation of the external product for n, the accumulator is obtained as follows:

$$\begin{aligned} \mathsf{acc} &= \mathsf{NTRU}_{f,Q}\left(\mathsf{testP(X)} \cdot X^{-b+\sum_{i=0}^{n-1} \boldsymbol{a}[i]\boldsymbol{s}[i]}\right) \\ &= \mathsf{NTRU}_{f,Q}\left(\mathsf{testP(X)} \cdot X^{-\left(\left\lfloor \frac{q}{2} \cdot x \right\rceil + e\right)}\right). \\ &= \mathsf{NTRU}_{f,Q}\left(\frac{Q}{4} x\right). \end{aligned}$$

Finally, after sample extraction, we extract the ciphertext $\mathsf{LWE}_Q(x)$ from acc, thereby refreshing the noise in the ciphertext x. And the encoding is converted from $\frac{Q}{2}$ to $\frac{Q}{4}$.

Algorithm 2 Constructing Efficient AND Gates Based-on NTRU.

```
Input:
    An LWE sample $\mathsf{ct}_x = (\boldsymbol{a}, b) \in \mathsf{LWE}^N_{\mathbf{s},Q}(x)$.
    An LWE sample $\mathsf{ct}_y = (\boldsymbol{a}', b') \in \mathsf{LWE}^N_{\mathbf{s},Q}(y)$.
    A bootstrapping key $\mathsf{bsk}_i : \{\mathsf{NGS}_{f,Q}(\boldsymbol{s}'[i])\}$, for $i \in [0, n-1]$.
Output:
    A refreshed LWE ciphertext $\mathsf{LWE}^N_{\boldsymbol{s},Q}(xy)$
 1: $\mathsf{ct} \leftarrow \mathsf{ModSwitch}_{Q \to Q_{\mathrm{KS}}(\mathsf{ct}_x)}$
 2: $\mathsf{ct} \leftarrow \mathsf{LtLKeySwtich}_{\boldsymbol{s} \to \boldsymbol{s}', N \to n}(\mathsf{ct})$
 3: $(\boldsymbol{a}, b) \leftarrow \mathsf{ModSwitch}_{Q_{\mathrm{KS}} \to q}(\mathsf{ct})$
 4: $\mathsf{testP}(X) \leftarrow \sum_{i=0}^{N-1} 2 \cdot \left\lfloor \frac{t}{q} \cdot i \right\rceil \cdot X^i \cdot X^{\frac{N}{2}}$
 5: $\mathsf{acc} \leftarrow \mathsf{testP}(X) \cdot X^{-b}$
 6: for $i = 0$ to $n-1$ do
 7:         $\mathsf{acc} = \mathsf{acc} + \big(\mathsf{acc} \cdot X^{\boldsymbol{a}[i]} - \mathsf{acc}\big) \boxdot \mathsf{bsk}_i$
 8: end for
 9: $\mathsf{ct} \leftarrow \mathsf{SampleExaction}(\mathsf{acc})$
10: $\mathsf{ct} \leftarrow \mathsf{Pack}(\mathsf{LWE}_Q(0), \mathsf{ct})$
11: $\mathsf{testP}'(X) \leftarrow \mathsf{ct} \cdot X^{N/2}$
12: $\mathsf{ct}' \leftarrow \mathsf{ModSwitch}_{Q \to Q_{\mathrm{KS}}}(\mathsf{ct}_y)$
13: $\mathsf{ct}' \leftarrow \mathsf{LtLKeySwtich}_{\boldsymbol{s} \to \boldsymbol{s}', N \to n}(\mathsf{ct})$
14: $(\boldsymbol{a}', b') \leftarrow \mathsf{ModSwitch}_{Q_{\mathrm{KS}} \to q}(\mathsf{ct})$
15: $\mathsf{acc}' \leftarrow \mathsf{testP}'(X) \cdot X^{-b'}$
16: for $i = 0$ to $n-1$ do
17:         $\mathsf{acc}' = \mathsf{acc}' + \Big(\mathsf{acc}' \cdot X^{\boldsymbol{a}'[i]} - \mathsf{acc}'\Big) \boxdot \mathsf{bsk}_i$
18: end for
19: $\mathsf{ct}'' \leftarrow \mathsf{SampleExaction}_Q(\mathsf{acc}')$
20: $\mathsf{ct}_{xy} \leftarrow \mathsf{ct}'' + \mathsf{ct}$
21: return $\mathsf{ct}_{xy}$
```

Step2: Setting $\mathsf{testP}'(\mathsf{X})$ by packing. The core challenge in designing an AND gate for a single-bit plaintext space is configuring the test polynomial for bootstrapping the ciphertext y. The primary step in this process involves packing both $\mathsf{LWE}(0)$ and $\mathsf{LWE}(x)$ into an NTRU ciphertext to construct the test polynomial as a lookup table. Packing technique is as follows:

$$\tilde{b} \leftarrow \sum_{i=0}^{N-1}(-\boldsymbol{a}[i]) \odot \mathsf{NTRU}_f(\boldsymbol{s}[i]\cdot(1+X+\cdots+X^{N-1})) + b \odot \mathsf{NTRU}'_f(f\cdot(1+\cdots+X^{N-1})).$$

At this time, $\tilde{b}$ is the desired NTRU ciphertext, $\mathsf{TestP}'(X) = \tilde{b}\cdot X^{N/2}$. For packing details, see Section 3.3.

Step3: Bootstrapping the $\mathsf{ct}_y = \mathsf{LWE}_{\boldsymbol{s},Q}(\frac{Q}{2}y)$. First, we apply key switching and modulus switching, as described in lines 1214 of Algorithm 2, to reduce the dimensionality, modulus, and noise of the LWE ciphertex. We obtain $(\boldsymbol{a}', b') = \mathsf{LWE}^n_{\boldsymbol{s}',q}(\frac{q}{2}y)$.

Setting the bootstrapping key bsk as:

$$\mathsf{bsk} = \begin{cases} \mathsf{bsk}_i = \mathsf{NGS}_{f,Q}(0), & \text{if } (\boldsymbol{s}[i]=0); \\ \mathsf{bsk}_i = \mathsf{NGS}_{f,Q}(1), & \text{if } (\boldsymbol{s}[i]=1). \end{cases}$$

Initializing the accumulator as: $\mathsf{acc} = \mathsf{testP}'(X)\cdot X^{-b}$

$$\mathsf{acc} = \mathsf{acc} + \left(\mathsf{acc}\cdot X^{\boldsymbol{a}[i]} - \mathsf{acc}\right) \boxdot \mathsf{bsk}_i, (0 \le i \le n-1). \tag{2}$$

Perform the external product n times:

$$\begin{aligned} \mathsf{acc} &= \mathsf{NTRU}_{f,Q}\left(\mathsf{testP}'(X)\cdot X^{-b+\sum_{i=0}^{n-1}\mathbf{a}[i]\boldsymbol{s}[i]}\right) \\ &= \mathsf{NTRU}_{f,Q}\left(\mathsf{testP}'(X)\cdot X^{-(\lfloor \frac{q}{t}\cdot y \rceil + e)}\right) \\ &= \mathsf{NTRU}_{f,Q}((-1)^{1-y}\frac{Q}{4}x) \end{aligned}$$

Extract the LWE ciphertext from the NTRU ciphertext to obtain $\mathsf{LWE}_Q\left((-1)^{1-y}\frac{Q}{4}x\right)$. Subsequently, add $\mathsf{LWE}_Q\left(\frac{Q}{4}x\right)$ to achieve the final result $\mathsf{LWE}_Q(\frac{Q}{2}xy)$.

3.3 Packing

Packing is to pack one or more LWE ciphertexts into an RLWE ciphertext or an NTRU ciphertext. The packing method of [CGGI17] Algorithm 1 is that the number of packed LWE ciphertexts must be N. If we need to pack is two ciphertexts, then we repeat each ciphertext $N/2$ times, and then pack N LWE ciphertexts into one RLWE ciphertext. But the variance of the noise introduced by ring packing is

$$\sigma^2 \le \frac{1}{12}\ell N^2 B^2 \sigma^2_{\text{LtRKSK}} + \frac{1}{3}n\epsilon^2, \quad \epsilon = \frac{1}{2}\lceil \frac{q}{B^\ell} \rceil. \tag{3}$$

Where N denote the dimension of the LWE ciphertexts, B and ℓ denote the base and the length of the gadget decomposition, respectively. The noise introduced by this method is significant, as the LWE dimension is N, resulting in greater noise than that introduced during the blind rotation in the bootstrapping process.

We propose a new method for efficiently packing LWE ciphertexts into an NTRU ciphertext, facilitating this transformation. Since the AND gate requires setting a test polynomial before the second blind rotation, the LWE ciphertexts of 0 and $\mathsf{LWE}_{\boldsymbol{s},Q}(\frac{Q}{4}x)$ must be packed into an NTRU cipertext. The packing method we propose introduces minimal noise, ensuring more efficient and secure ciphertext transformations.

Correctness and Error Analysis.

Algorithm 3 Packing $\mathsf{Pack}(\mathsf{LWE}^N_{\boldsymbol{s},Q}(0), \mathsf{ct})$

```
Input: Two LWE ciphertext $\mathsf{LWE}^N_{\boldsymbol{s},Q}(0)$ and $\boldsymbol{ct} = (\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{s}\rangle + \frac{Q}{4}x + e) \in \mathsf{LWE}^N_{\boldsymbol{s},Q}(\frac{Q}{4}x)$.
Input: $\mathsf{LTNKSK}_i = \mathsf{NTRU}_f(\boldsymbol{s}[i] \cdot (1 + X + ... + X^{N-1}))$, for $i \in [0, N-1]$
    $\mathsf{KSK} = \mathsf{NTRU}'_f(f \cdot (1 + \cdots X^{N-1}))$.
Output: $\mathsf{NTRU}_{f,Q}(\frac{Q}{4}x(1 + X + \cdots + X^{N-1}))$
    $\mathsf{ct} = b \odot \mathsf{KSK}$
    for $i = 0$ to $N-1$ do
        $\mathsf{ct} \leftarrow \mathsf{ct} + (-\boldsymbol{a}[i]) \odot \mathsf{LTNKSK}_i$
    end for
    return An NTRU ciphertext $\mathsf{ct}$
```

Proof. Basing the correctness of the gadget multiplication, we have

$$\begin{aligned}
&\sum_{i=0}^{N-1} (-\boldsymbol{a}[i]) \odot \mathsf{NTRU}_f(\boldsymbol{s}[i] \cdot (1 + X + ... + X^{N-1})) + b \odot \mathsf{KSK} \\
&= \mathsf{NTRU}_{f,Q}((\sum_{i=0}^{N-1} (-\boldsymbol{a}[i]\boldsymbol{s}[i] + b)(1 + X + \cdots + X^{N-1})) \\
&= \mathsf{NTRU}_{f,Q}((\frac{Q}{4}x)(1 + X + \cdots + X^{N-1})).
\end{aligned}$$

□

Then, the error variance introduced by packing is

$$\sigma^2 \le \frac{1}{12}\ell N B^2 \sigma^2_{\text{LtRKSK}} + \frac{1}{3}n\epsilon^2, \quad \epsilon = \frac{1}{2}\lceil \frac{q}{B^\ell} \rceil. \tag{4}$$

where N denote the dimension of the LWE ciphertexts, B and ℓ denote the base and the length of the gadget decomposition, respectively.

4 Error Analysis and Implementation

4.1 Error Analysis and Parameter Selection.

This section mainly analyzes the noise of our NTRU-based instantiation. In this context, $\sigma^2_{\text{Packing}}$, σ^2_{BR}, σ^2_{LtLKS} represent the noise levels introduced by the packing, blind rotation and LWE-to-LWE key switching, respectively.

$$\begin{aligned}
\sigma^2_{\text{Packing}} &\le \frac{1}{12}n\ell B^2 \sigma^2_{\text{LtRKSK}} + \frac{1}{3}n\epsilon^2, \ell = \lfloor \log_B Q \rfloor, \epsilon = \frac{1}{2}\left\lceil \frac{Q}{B^\ell} \right\rceil. \\
\sigma^2_{\text{BR}} &\le \frac{1}{12}nN\ell B^2 \sigma^2_{\text{BRKSK}} + \frac{1}{3}nN\epsilon^2, \ell = \lfloor \log_B Q \rfloor, \epsilon = \frac{1}{2}\left\lceil \frac{Q}{B^\ell} \right\rceil.
\end{aligned}$$

$$\sigma^2_{\text{LtLKS}} \le N\ell\sigma^2_{\text{LtLKSK}}, \ell = \lfloor \log_B Q \rfloor.$$

The formula for evaluating the decryption rate is $1 - \text{erf}\left(\frac{q/t}{2\sqrt{2}\sigma}\right)$, where in our framework, the plaintext modulus $t = 2$, so the decryption failure rate formula is: $1 - \text{erf}\left(\frac{q/2}{2\sqrt{2}\sigma}\right)$. σ is the standard deviation of the noise in the entire bootstrap process.

For the GGBS, $\sigma = \sqrt{\frac{q^2}{Q^2}\left(\sigma^2_{\text{Packing}} + 3\sigma^2_{\text{BR}}\right) + \frac{Q^2_{\text{KS}}}{Q^2}\left(\sigma^2_{\text{KS}} + \sigma^2_{\text{MS1}}\right) + \sigma^2_{\text{MS2}}}$.

For the IGBS, $\sigma = \sqrt{\frac{q^2}{Q^2}(\sigma_{\mathrm{BR}}^2) + \frac{Q_{\mathrm{KS}}}{Q^2}(\sigma_{\mathrm{KS}}^2 + \sigma_{\mathrm{MS1}}^2) + \sigma_{\mathrm{MS2}}^2}$.

we give the NTRU/NGS and LWE parameters required for the NTRU-based single-bit plaintext space gate bootstrapping scheme at a 128-bit security level, as shown in Tables 3 and 4. The security level of HE schemes is determined by several factors, including the secret key distribution, the dimensions and modulus of the (R)LWE sample, and the standard deviations of the error according to the HE standard [ACC+18]. Then, we use the LWE estimator [APS15] to estimate the security level.

Table 3: Bootstrapping parameters for NTRU/NGS ciphertext. Among themλ ,N,σ,B_{BR} denotes the security level, the degree of the ring polynomial, the standard deviation of the NTRU key f, the basis of gadget decomposition of blind rotations, respectively. while ℓ_{BR} and ℓ'_{BR} refer to the lengths of the gadget decomposition and the approximate gadget decomposition, respectively. .

Parameters	Key distrib.	λ	N	σ	Q	B_{BR}	ℓ_{BR}	ℓ'_{BR}	FR.
STD128B2	Gaussian	128	2^{10}	3.5	$\approx 2^{25}$	2^8	4	2	2^{-115}

Table 4: Bootstrapping parameters for LWE ciphertext. In this context, λ ,n ,σ is the security level, the dimension of the LWE ciphertext, the standard deviation of the noise in the fresh ciphertext, Q_{KS} refers to the base used for key switching, and ℓ_{KS} specifies the number of blocks utilized during the key switching process.

Parameters	Key distrib.	λ	n	σ	q	Q_{KS}	B_{KS}	ℓ_{KS}
STD128B2	Binary	128	541	3.19	2^{11}	2^{13}	2^7	2

Table 5: Gate Bootstrapping parameters for other schemes.

Scheme	Key distrib.	λ	n	N	σ	Q	B	ℓ_{BR}	ℓ'_{BR}
[MP21]	Ternary	128	512	1024	3.19	$\approx 2^{25}$	2^7	4	–
[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	–
[LLW+24]	Bianry	128	571	1024	3.19	$\approx 2^{19.9}$	2^3	6	5

It is important to note that the LWE estimator has been recently updated, and our NTRU parameters have been evaluated using the latest version. Our parameters STD128B2 are highly compact. Not only has the number of blocks in the gadget decomposition been reduced from 5 to 2, but the LWE dimension has also decreased from 571 to 541, resulting in a significant improvement in efficiency. The techniques for noise management are as follows.

Enhanced Noise Control. We have revised the parameter settings of the scheme presented in [LLW+24] and selected more optimal parameters. Specifically, we can reduce n from 571 to 541 and decrease the decomposition length from 5 to 2. This improvement is primarily attributed to the following reasons.

- We use single-bit message encoding, whereas the existing scheme uses two-bit message encoding. A bit of a gap allows more freedom for noise growth.
- Existing gate bootstrapping methods utilize a 2-bit adder, requiring an addition operation prior to bootstrapping. In contrast, our Free-XOR gate bootstrapping

eliminates the need for this addition, thereby saving 0.5 bits of space and allowing for the accommodation of more noise.

- To maintain the desired security level, we first increased the noise in the NTRU key and subsequently raised the NTRU modulus to achieve more effective noise control.

- We swapped the gate bootstrapping process in [DM15a] to align with the procedure used in LMK+ [LMK+23]. The gate bootstrapping begins with the modulus Q. This approach allows the noise from both XOR and AND operations to accumulate under modulus Q, thereby enabling more effective noise control.

Smaller Key Size. Using NTRU-based blind rotations and reducing the number of blocks in gadget decomposition helps minimize the key size. In Table 6, we compare the key sizes across different schemes. Our approach employs NTRU-based blind rotations, leading to a reduction in the number of n and gadget decomposition blocks. When the LWE key is binary, the size of NTRU is given by $\ell_{BR} n N \log_2(Q)$. Our key size of 2.1MB, which is 3.3 times smaller than [LLW+24].

4.2 Implementation and Performance

We implemented our algorithm based on our own library. The experiments were conducted on a machine equipped 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, and compiled with Clang version 11.3.0.

We implemented an NTRU-based instantiation of the AND gate in our library. Under the STD128B2 parameters, our GGBS achieves speeds that are 3 times faster than FHEW-like gate bootstrapping and 1.15 times faster than [LLW+24], with key sizes reduced by a factor of 6 and 3.3, respectively. Our IGBS achieves even faster performance, being 6 times faster than FHEW-like gate bootstrapping and twice as fast as Lis method, with key sizes reduced by a factor of 6 and 2.35, respectively.

Table 6: Comparison of running times with other shcems.

Schemes	Assumption	Key Size (MB)	Times (ms)
FHEW [MP21]	RLWE	13.5	102.5
LMK+[LMK+23]	RLWE	12.7	113
XZD+ [XZD+23]	NTRU	17.9	61.2
LLW+ [LLW+24]	NTRU	7.1	40
GGBS STD128B2	NTRU	**2.1(3.3×)**	**35(1.15×)**
IGBS STD128B2	NTRU	**2.1(3.3×)**	**17(2.35×)**

5 Optimization techniques

In this section, we primarily discuss the optimization techniques employed in our approach, including key unrolling and AVX512 acceleration instructions.

5.1 Blind Rotation Using Key Unrolling Technique

Using the technique from [BMMP18], we enhance the efficiency of blind rotation in bootstrapping by reducing the number of external products. Specifically, instead of computing $\boldsymbol{a}_i \cdot \boldsymbol{s}_i$ bit by bit, we now group pairs and compute them together. By observing the following formula:

$$X^{as+a's'} = ss'X^{a+a'} + s\left(1-s'\right)X^{a} + (1-s)s'X^{a'} + (1-s)\left(1-s'\right)$$

To compute this new function, they modify the bootstrapping key to include encryptions of the values $ss', s(1-s'), (1-s)s'$, and $(1-s)(1-s')$. As a result, this idea reduces the number of iterations in blind rotation by half, effectively halving the number of external product computations. At the cost of increasing the size of the bootstrapping key to four times its original size. Specifically, the accumulation process of the CMux gate blind rotation changes from $X^{b+\sum_{i=0}^{n-1}\boldsymbol{a}[i]\boldsymbol{s}[i]}$ to:

$$X^{\sum_{i=0}^{n-1}\boldsymbol{a}[i]\boldsymbol{s}[i]} = X^{\sum_{i=0}^{(n-1)/2}\boldsymbol{a}[2i]\boldsymbol{s}[2i]+\boldsymbol{a}[2i+1]\boldsymbol{s}[2i+1]}.$$

Specifically, as in Algorithm 4. Table 5 presents a comparative analysis of the computation time, demonstrating the improvements achieved by employing approximate gadget decomposition and key unrolling techniques relative to other existing schemes.

Algorithm 4 Blind Rotate Using Key Unrolling.

```
Input:
    An LWE sample $\mathsf{ct} = (\mathbf{a}, b) \in \mathsf{LWE}^n_{\mathbf{s},q}(m)$, where $q|2N$.
    For all $i$ in $[0, n/2-1]$, $\mathrm{bk}_{i,0}, \mathrm{bk}_{i,1}$, and $\mathrm{bk}_{i,2}$ are respectively NGS encryptions of
    $\boldsymbol{s}[2i]\boldsymbol{s}[2i+1], \boldsymbol{s}[2i]\left(1-\boldsymbol{s}[2i+1]\right)$, and $\boldsymbol{s}[2i+1]\left(1-\boldsymbol{s}[2i]\right)$
    A test polinomial $\mathsf{testP}(X)$
Output:
    An RLWE ciphertext $\mathsf{acc} \in \mathsf{RLWE}_{s',Q}(X^{-b+\sum_{i=0}^{n-1}\boldsymbol{a}[i]\boldsymbol{s}[i]})$ .
1: Set $\mathsf{acc} = X^{-b} \cdot \mathsf{testP}(X) \in \mathcal{R}_Q^2$
2: for $i = 0$ to $n/2-1$ do
3:     $\mathsf{acc} \leftarrow \left(\left(X^{\boldsymbol{a}[2i]+\boldsymbol{a}[2i+1]}-1\right)\mathbf{brk}_{i,0}+\left(X^{\boldsymbol{a}[2i]}-1\right)\mathbf{brk}_{i,1}+\left(X^{\boldsymbol{a}[2i+1]}-1\right)\mathbf{brk}_{i,2}\right)\boxdot\mathsf{acc}$
4: end for
5: return acc.
```

Performance. Our GGBS is accelerated by key unrolling to 23ms. Compared with the original algorithm, it is about 1.5 speed up.
Error Analysis and Decryption Failure Rate. Note that the key unrolling can increase the growth compared to the original blind rotation process as shown in scheme [LLW+24]. Further, the error variance of blind rotation is 1.5 times larger than the original σ^2_{BR}. Based on the error analysis shown in Section 4.1, we can get the current decryption failure rate is now 2^{-58}.

5.2 AVX-512 Instructions

The Intel AVX (Advanced Vector Extensions) instruction set, developed by Intel, is an extension designed to enhance CPU performance during vectorized operations. First introduced in 2011, AVX has been progressively expanded and optimized in subsequent processor generations. By increasing the width of registers (from 128-bit to 256-bit, and further to 512-bit with AVX-512) and incorporating additional floating-point and integer arithmetic instructions, the AVX instruction set significantly boosts parallel computing capabilities.

In FHE, polynomial operations are among the core computational tasks. The AVX instruction set, leveraging vectorization techniques, enables parallel processing of polynomial coefficients as vector elements. For instance, within the NTT/FFT algorithms, AVX can utilize its 256-bit or 512-bit data instructions to handle multiple coefficients simultaneously, thereby accelerating the computation of a single polynomial multiplication. In homomorphic encryption schemes, AVX instructions can be employed to expedite the encryption and decryption of ciphertexts. By exploiting the vectorization and parallelism capabilities of the AVX instruction set, the computational complexity of fully homomorphic encryption schemes can be significantly reduced, enhancing their practical feasibility.

After applying key unrolling techniques and optimizing with AVX512 instructions, the runtime of our gate bootstrapping scheme is presented in Table 7.

Table 7: Comparison of bootstrapping runtimes with AVX instructions.

Scheme	Instruction	FFT/NTT	Key Size (MB)	Time (ms)
TFHEpp [Mat20]	AVX-512	FFT	13.5	9
TFHE-rs [Zam22]	AVX-512	FFT	13.5	6.8
LLW+[LLW+24]	AVX-512	NTT	10.7	3.8
Our GGBS	AVX-512	NTT	**3.2(3.3×)**	**3.7**
Our IGBS	AVX-512	NTT	**3.2(3.3×)**	**1.8(2.1×)**

6 Application

Transciphering. Although current FHE schemes are approaching practical applicability, the size of homomorphic ciphertexts remains several orders of magnitude larger than that of plaintexts, primarily due to the security of these schemes relying on hard problems such as LWE, RLWE, and NTRU. This necessitates the use of larger modulus and higher-degree polynomials to ensure sufficient security. To address these challenges, Naehrig et al. [NLV11] proposed a transciphering framework. Specifically, the client encrypts the data using a symmetric encryption scheme, producing a ciphertext $\mathcal{E}_k(m)$. Then, the client generates a homomorphic encryption of the symmetric key, $\mathsf{Enc}(k)$, and transmits both the encrypted key and the ciphertext to the server. Upon receiving the encrypted data, the server homomorphically evaluates the decryption circuit of the symmetric encryption. Using $\mathsf{Enc}(k)$ and $\mathcal{E}_k(m)$, the server computes the homomorphic decryption of the ciphertext, resulting in the homomorphically encrypted original data: Once the data is decrypted homomorphically, the server can proceed with further homomorphic evaluations on the encrypted data.

Given that our single-bit plaintext bootstrapping method is inherently XOR-free, it provides substantial advantages for the homomorphic evaluation of symmetric ciphers, particularly those with a high proportion of XOR operations and lightweight symmetric ciphers. To demonstrate its effectiveness, we present its application in the homomorphic computation of AES and Ascon.

6.1 Parameter Selection and Implementation

We have chosen the parameters **STD128B3** for the LWE and NTRU as detailed in Tables 8 and 9, respectively. To facilitate a higher number of XOR-free operations, we expand the basis B_{BR} of the blind rotation decomposition and increase the number of blocks used in the calculations.

Table 8: Bootstrapping parameters for NTRU/NGS ciphertext.In this context, λ denotes the security parameter, N is the degree of the ring polynomial, σ represents the standard deviation of the NTRU key f, B_{BR} indicates the basis used for blind rotations, while ℓ_{BR} and ℓ'_{BR} refer to the lengths of the gadget decomposition and the approximate gadget decomposition, respectively.

Parameters	Key distrib.	λ	N	σ	Q	B_{BR}	ℓ_{BR}	ℓ'_{BR}	FR.
STD128B3	Gaussian	128	2^{10}	2.45	$\approx 2^{25}$	2^6	4	3	2^{-748}

Table 9: Bootstrapping parameters for LWE ciphertext. In this context, λ represents the security parameter, n denotes the dimension of the LWE ciphertext, σ indicates the standard deviation of the noise in the fresh ciphertext, B_{KS} refers to the base used for key switching, and ℓ_{KS} specifies the number of blocks utilized during the key switching process.

Parameters	Key distrib.	λ	n	σ	q	Q_{KS}	B_{KS}	ℓ_{KS}
STD128B3	Binary	128	541	3.19	2^{11}	2^{13}	2^7	2

The formula we employ to calculate the decryption failure rate is given by: $1 - \mathsf{erf}\left(\frac{q/4}{\sqrt{2}\sigma}\right)$.

We employ our Free-XOR gate bootstrapping framework in the single-bit plaintext space to homomorphically compute AES and ASCON in two modes.

- GGBS is primarily used for computing the AND gate, and the noise after each AND gate computation is approximately $\sigma^2_{3\mathrm{BR}}$.
- IGBS is primarily used to reduce noise following the homomorphic evaluation of XOR gates and to refresh the ciphertext of each bit after a single round of homomorphic AES and ASCON computation.

Table 10 presents the execution times of GGBS and IGBS under the parameters of STD128B3, which are implemented in C and optimized using AVX512 instructions.

Table 10: Execution Time (in milliseconds) for C and AVX512 Implementations of General Gate Bootstrapping (GGBS) and Identity Gate Bootstrapping Scheme (IGBS). The reported times represent the average execution duration for each GGBS and IGBS implementation, based on 100,000 runs.

Implementation	**STD128B3**	
	C (ms)	AVX512 (ms)
GGBS	39.2	5.05
IGBS	18.66	2.51

6.2 Homomorphic evaluation of AES

AES-128 is a variant of the AES (Advanced Encryption Standard) encryption algorithm, indicating that this version uses a 128-bit key length for encryption and decryption. AES-128 uses a 128-bit (16-byte) key and a fixed 128-bit (16-byte) block size, processing each data block at this length and performing 10 rounds of encryption. Each round primarily includes SubBytes, ShiftRows, MixColumns and AddRoundKey, with the 10-th round only performing three steps: ShiftRows, MixColumns and AddRoundKey.

AddRoundKey. The round keys are derived from the original key during the key expansion process and are provided alongside the message prior to the commencement of encryption. The first round key is applied at the start of the computation, with the remaining round keys introduced at the conclusion of each subsequent round, for a total of 11 applications. Since these round keys are freshly generated, this process is XOR-free.

ShiftRows. The shifting of rows is a simple operation that only requires swapping of indices trivially handled in the code. This operation has no effect on the noise.

MixColumns. The MixColumns operation involves a 4×4 matrix multiplication with constant terms $\{x+1, x, 1\}$ modulo $(x^8+x^4+x^3+x+1)$. In fact, this modular multiplication can be efficiently computed using simple XOR and shift operations, as demonstrated in [DHS16, WWL+23].

$$b_0b_1b_2b_3b_4b_5b_6b_7 \xrightarrow{\times 1} b_0b_1b_2b_3b_4b_5b_6b_7$$
$$b_0b_1b_2b_3b_4b_5b_6b_7 \xrightarrow{\times x} b_7b_0b_1b_2b_3b_4b_5b_6 \oplus 0b_70b_7b_7000$$
$$b_0b_1b_2b_3b_4b_5b_6b_7 \xrightarrow{\times (x+1)} b_0b_1b_2b_3b_4b_5b_6b_7 \oplus b_7b_0b_1b_2b_3b_4b_5b_6 \oplus 0b_70b_7b_7000$$

Given the minimal number of XOR operations required, this step can be considered XOR-free.

SubBytes. Homomorphic evaluation S-Box is the most computationally expensive part of homomorphic encryption in AES. We adopt S-box3 from [Lab], which consists of 94 XOR gates, 4 NOT gates, and 34 AND gates. The complete S-box logic is given in Appendix A. We will illustrate our approach to homomorphic computation by using the first two steps of the S-Box as examples.

Table 11: XOR-1(27)

T1 = X0 + X3	T2 = X0 + X5	T3 = X0 + X6
T4 = X3 + X5	T5 = X4 + X6	T6 = T1 + T5
T7 = X1 + X2	T8 = X7 + T6	T9 = X7 + T7
T10 = T6 + T7	T11 = X1 + X5	T12 = X2 + X5
T13 = T3 + T4	T14 = T6 + T11	T15 = T5 + T11
T16 = T5 + T12	T17 = T9 + T16	T18 = X3 + X7
T19 = T7 + T18	T20 = T1 + T19	T21 = X6 + X7
T22 = T7 + T21	T23 = T2 + T22	T24 = T2 + T10
T25 = T20 + T17	T26 = T3 + T16	T27 = T1 + T12

Table 12: Round $1-10$ XOR-1(27), the unit is σ_{BR}^2.

Cipertext	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
Noise	2	2	2	2	2	4	2	3	3	6	2	2	4	6
Cipertext	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24	T25	T26	T27	
Noise	4	4	7	2	4	6	2	3	5	8	13	6	4	

Step1: Evaluate 27 XOR gates homomorphically.

- The noise in the ciphertexts $X1, \ldots, X7$ is the noise refreshed by IGBS, denoted as σ^2_{BR}. This value will be used as the unit of noise for further calculations. After evaluating 27 XOR gates, the noise list from $T1$ to $T27$ is shown in Table 12. No bootstrapping is necessary at this stage.

Step2: Homomorphic evaluation of 9 AND gates.

- For each multiplication, that is, when performing an AND gate, the noise accumulation for each AND gate is approximately σ^2_{BR}.

Table 13: AND-1 (9)

M1 = T13 ∗ T6	M2 = T23 ∗ T8	M4 = T19 ∗ X7
M6 = T3 ∗ T16	M7 = T22 ∗ T9	M9 = T20 ∗ T17
M11 = T1 ∗ T15	M12 = T4 ∗ T27	M14 = T2 ∗ T10

Table 14: The ciphertext noise after AND-1(9) of round 1-10, the unit is σ^2_{BR}.

Ciphertext	M1	M2	M4	M6	M7	M9	M11	M12	M14
Noise	3	3	3	3	3	3	3	3	3

Table 15: Counts of GGBS and IGBS per Round and Total Counts from Rounds 1 to 10.

	Round 1	Round 2	...	Round 10	Total (1∼10)
GGBS	$34 \cdot 16$	$34 \cdot 16$	...	$34 \cdot 16$	5400
IGBS	$8 \cdot 16$	$8 \cdot 16$	...	$8 \cdot 16$	1280

In the subsequent homomorphic evaluation, the operations are as follows: XOR-2(16), AND-2(3), XOR-3(2), AND-3(4), XOR-4(11), AND-4(18), and XOR-5(42). Homomorphic calculation of AES requires 5400 GGBS and 1280 IGBS, as shown in 15.

6.3 Homomorphic Evaluation of ASCON

Ascon is a lightweight cryptographic algorithm initially proposed by Graz University of Technology [DEMS21]. After a series of developments and optimizations, it distinguished itself in both the NIST and CAESAR competitions, eventually being selected as a NIST standard algorithm. Ascon is both a lightweight authenticated encryption with associated data (AEAD) algorithm and a lightweight hash (HASH) function.

The most computationally expensive component of ASCON is the 320-bit permutation, which consists of Round constan addition, the S-box layer, and the linear diffusion layer. The 5bit S-box of ASCON is shown in Figure 7. The logical operations corresponding to the S-box are shown in the Table 16. In hashing mode, ASCON requires 64 S-boxes per round, with a recommended total of 12 rounds.

Round constant addition. Since these round keys are freshly generated, this process is XOR-free.

Homomorphic computation of S-box. Each S-box has 5 bits, and the noise after homomorphically computing an S-box is shown in the table 17.

Linear Diffusion Layer. The linear diffusion layer introduces diffusion within each 64-bit register word x_i ,which is shown as:

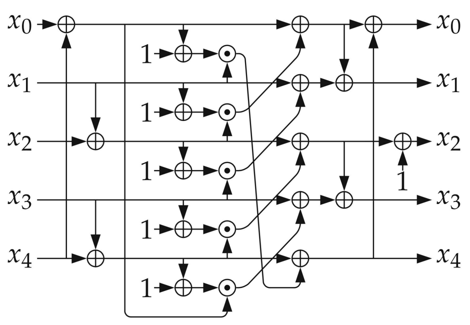


Figure 7: S-box of ASCON [DEMS21].

Table 16: S-box of ASCON.

$x0 = x0 + x4$	$x4 = x4 + x3$	$x2 = x2 + x1$		
$t0 = x0$	$t1 = x1$	$t2 = x2$	$t3 = x3$	$t4 = x4$
$t0 = \neg t0$	$t1 = \neg t1$	$t2 = \neg t2$	$t3 = \neg t3$	$t4 = \neg t4$
$t0 = t0 * x1$	$t1 = t1 * x2$	$t2 = t2 * x3$	$t3 = t3 * x4$	$t4 = t4 * x0$
$x0 = x0 + t1$	$x1 = x1 + t2$	$x2 = x2 + t3$	$x3 = x3 + t4$	$x4 = x4 + t0$
$x1 = x1 + x0$	$x0 = x0 + x4$	$x3 = x3 + x2$	$x2 = \neg x2$	

Table 17: Noise1 refers to the noise after computing the ASOCN S-box, while Noise2 represents the noise after performing a permutation, the unit is σ_{BR}^2.

Ciphertext	x_0	x_1	x_2	x_3	x_4
Noise1	7	9	5	11	4
Noise2	22	21	27	29	20

$$\begin{aligned}
x_0 &\leftarrow x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)\\
x_1 &\leftarrow x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)\\
x_2 &\leftarrow x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)\\
x_3 &\leftarrow x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)\\
x_4 &\leftarrow x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)
\end{aligned}$$

After the Linear Diffusion Layer is processed homomorphically, the accumulation of noise is presented in Table 17.

Table 18: Round-by-Round Counts and Cumulative Totals of GGBS and IGBS Operations in ASCON (Rounds 1 to 12).

	Round 1	...	Round 12	Total (1∼12)
GGBS	$5 \cdot 64$	...	$5 \cdot 64$	3840
IGBS	$5 \cdot 64$	...	$5 \cdot 64$	3840

6.4 Performances

AES. Based on the execution times of the GGBS and IGBS operations measured in Table 10, and considering the number of gates required for the homomorphic AES computation outlined in Table 15, the total computation time can be calculated as follows: 5400

GGBS gates, each taking 5.05 seconds, and 1600 IGBS gates, each taking 2.51 seconds. Consequently, the total computation time is 30.5 seconds, while the homomorphic computation of the entire AES takes 31 seconds. In Table 19, we compare the performance of various AES homomorphic evaluation schemes. Our method completes the AES computation in only 31 seconds, making it 1.5 times faster than the state-of-the-art Thunderbird [WLW$^+$24].

ASCON. Based on the previously measured execution times for GGBS and IGBS operations in Table 10, and considering the number of gates required for homomorphic AES computation in Table 18. The total computation time can be calculated as follows: 3840 GGBS operations are required, with each GGBS taking 5.05 milliseconds. This results in a total computation time of 27.5 seconds. For the homomorphic evaluation of the complete ASCON, the total time is 28 seconds. In Table 20, we compare the performance of various schemes for AES homomorphic evaluation. Our approach is 4.8 times faster than the state-of-the-art method presented in [BPR23].

Table 19: Comparison of Homomorphic AES Computation, FBS stands for Functional Bootstrapping, and CBS denotes Circuit Bootstrapping.

Scheme	Evaluation mode	Latency
BGV	Leveled [GHS12]	4mins
	Bootstrapped [GHS12]	18mins
CKKS	Bootstrapped [ADE$^+$23]	31mins
FHEW-like	FBS [RSK22]	4.2mins
	FBS [BPR23]	212s
	CBS [WWL$^+$23]	86s
	CBS [WLW$^+$24]	46s
NTRU-Based	**Free-XOR GBS**	**31s(1.5x)**

Table 20: Comparison of Homomorphic ASCON Computation. FBS, GBS stands for Functional Bootstrapping resbectively.

Scheme	Evaluation mode	Latency
CKKS	Bootstrapped [ADE$^+$23]	1260s
FHEW-like	TFHE GBS [MG21]	200s
	FBS [BPR23]	131s
NTRU-Based	**Free-XOR GBS**	**28s(4.7x)**

7 Conclusion

In this paper, we propose a Free-XOR gate bootstrapping framework based on a single-bit plaintext space. To ensure completeness, we present a construction workflow for AND gates along with a specific instantiation based on NTRU. Our improved noise management allows us to select very compact parameters, resulting in our GGBS algorithm being faster than existing state-of-the-art schemes, despite requiring two blind rotations. Additionally, we achieve a reduction in key size by a factor of 3.3. Furthermore, we introduce a novel method for packing LWE into NTRU, which maintains low noise levels. Finally, we apply the Free-XOR gate bootstrapping framework to transciphering scenarios, demonstrating efficient homomorphic computation of AES and ASCON, with improvements of 1.5 times and 4.7 times, respectively, compared with the state-of-the-art work.

A The S-box Circuit of AES

The S-box of AES is shown in the Table 21.

Table 21: The S-box of AES

#XOR-1(27)	M10 = M9 + M6	M47 = M40 * T8	L25 = L6 + L10
T1 = X0 + X3	M13 = M12 + M11	M48 = M39 * X7	L26 = L7 + L9
T2 = X0 + X5	M15 = M14 + M11	M49 = M43 * T16	L27 = L8 + L10
T3 = X0 + X6	M16 = M3 + M2	M50 = M38 * T9	L28 = L11 + L14
T4 = X3 + X5	M17 = M5 + T24	M51 = M37 * T17	L29 = L11 + L17
T5 = X4 + X6	M18 = M8 + M7	M52 = M42 * T15	Y0 = L6 + L24
T6 = T1 + T5	M19 = M10 + M15	M53 = M45 * T27	Y1 = L16 + L26
T7 = X1 + X2	M20 = M16 + M13	M54 = M41 * T10	Y1 = Y1 + 1
T8 = X7 + T6	M21 = M17 + M15	M55 = M44 * T13	Y2 = L19 + L28
T9 = X7 + T7	M22 = M18 + M13	M56 = M40 * T23	Y2 = Y2 + 1
T10 = T6 + T7	M23 = M19 + T25	M57 = M39 * T19	Y3 = L6 + L21
T11 = X1 + X5	M24 = M22 + M23	M58 = M43 * T3	Y4 = L20 + L22
T12 = X2 + X5	M27 = M20 + M21	M59 = M38 * T22	Y5 = L25 + L29
T13 = T3 + T4		M60 = M37 * T20	Y6 = L13 + L27
T14 = T6 + T11	#AND-2(3)	M61 = M42 * T1	Y6 = Y6 + 1
T15 = T5 + T11	M25 = M22 * M20	M62 = M45 * T4	Y7 = L6 + L23
T16 = T5 + T12	M31 = M20 * M23	M63 = M41 * T2	Y7 = Y7 + 1
T17 = T9 + T16	M34 = M21 * M22		
T18 = X3 + X7		#XOR-5(42)	
T19 = T7 + T18	#XOR-3(2)	L0 = M61 + M62	
T20 = T1 + T19	M28 = M23 + M25	L1 = M50 + M56	
T21 = X6 + X7	M26 = M21 + M25	L2 = M46 + M48	
T22 = T7 + T21		L3 = M47 + M55	
T23 = T2 + T22	#AND-3(4)	L4 = M54 + M58	
T24 = T2 + T10	M29 = M28 * M27	L5 = M49 + M61	
T25 = T20 + T17	M30 = M26 * M24	L6 = M62 + L5	
T26 = T3 + T16	M32 = M27 * M31	L7 = M46 + L3	
T27 = T1 + T12	M35 = M24 * M34	L8 = M51 + M59	
		L9 = M52 + M53	
#AND-1(9)	#XOR-3(11)	L10 = M53 + L4	
M1 = T13 * T6	M33 = M27 + M25	L11 = M60 + L2	
M2 = T23 * T8	M36 = M24 + M25	L12 = M48 + M51	
M4 = T19 * X7	M37 = M21 + M29	L13 = M50 + L0	
M6 = T3 * T16	M38 = M32 + M33	L14 = M52 + M61	
M7 = T22 * T9	M39 = M23 + M30	L15 = M55 + L1	
M9 = T20 * T17	M40 = M35 + M36	L16 = M56 + L0	
M11 = T15 * T1	M41 = M38 + M40	L17 = M57 + L1	
M12 = T27 * T4	M42 = M37 + M39	L18 = M58 + L8	
M14 = T10 * T2	M43 = M37 + M38	L19 = M63 + L4	
	M44 = M39 + M40	L20 = L0 + L1	
#XOR-2(16)	M45 = M42 + M41	L21 = L1 + L7	
M3 = T14 + M1		L22 = L3 + L12	
M5 = M4 + M1	#AND-4(18)	L23 = L18 + L2	
M8 = T26 + M6	M46 = M44 * T6	L24 = L15 + L9	

References

[ACC$^+$18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. Homomorphic encryption security standard. *HomomorphicEncryption. org, Toronto, Canada, Tech. Rep*, 11, 2018.

[ADE$^+$23] Ehud Aharoni, Nir Drucker, Gilad Ezov, Eyal Kushnir, Hayim Shaul, and Omri Soceanu. E2e near-standard and practical authenticated transciphering. *Cryptology ePrint Archive*, 2023.

[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.

[ASP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*, pages 297–314. Springer, 2014.

[BGV14a] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BGV14b] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BIP$^+$22a] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder VL Pereira, and Nigel P Smart. Final: faster fhe instantiated with ntru and lwe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 188–215. Springer, 2022.

[BIP$^+$22b] 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.

[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.

[BPR23] Nicolas Bon, David Pointcheval, and Matthieu Rivain. Optimized homomorphic evaluation of boolean functions. *Cryptology ePrint Archive*, 2023.

[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. *ADVANCES IN CRYPTOLOGY-ASIACRYPT 2016, PT I*, 10031:3–33, 2016.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 377–408. Springer, 2017.

[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.

[CHK$^+$18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.

[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.

[CLOT21] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 670–699. Springer, 2021.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. 2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34:1–42, 2021.

[DHS16] Yarkın Doröz, Yin Hu, and Berk Sunar. Homomorphic aes evaluation using the modified ltv scheme. *Designs, Codes and Cryptography*, 80:333–358, 2016.

[DM15a] Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*, pages 617–640. Springer, 2015.

[DM15b] 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.

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

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. In *Annual Cryptology Conference*, pages 850–867. Springer, 2012.

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched ntru parameters. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–26. Springer, 2017.

[Lab] Information Technology Laboratory. https://csrc.nist.gov/projects/circuit-complexity/list-of-circuits. In *NIST*.

[LLW+24] Zhihao Li, Xianhui Lu, Zhiwei Wang, Ruida Wang, Ying Liu, Yinhang Zheng, Lutan Zhao, Kunpeng Wang, and Rui Hou. Faster ntru-based bootstrapping in less than 4 ms. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):418–451, 2024.

[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.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23, Heidelberg, 2010. Springer.

[Mat20] Kotaro Matsuoka. TFHEpp: pure C++ implementation of TFHE cryptosystem. https://github.com/virtualsecureplatform/TFHEpp, 2020.

[MG21] Kalikinkar Mandal and Guang Gong. Homomorphic evaluation of lightweight cipher boolean circuits. In *International Symposium on Foundations and Practice of Security*, pages 63–74. Springer, 2021.

[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.

[NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124, 2011.

[Per21] Hilder Vitor Lima Pereira. Bootstrapping fully homomorphic encryption over the integers in less than one second. In *Public-Key Cryptography–PKC 2021: 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10–13, 2021, Proceedings, Part I 24*, pages 331–359. Springer, 2021.

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

[RSK22] Rasoul Akhavan Mahdavi Roy Stracovsky and Florian Kerschbaum. Faster evaluation of aes using tfhee. In *Poster Session, FHE.Org - 2022, 2022. https: //rasoulam.github.io/data/poster-aes-tfhe.pdf*, 2022.

[WLW+24] Benqiang Wei, Xianhui Lu, Ruida Wang, Kun Liu, Zhihao Li, and Kunpeng Wang. Thunderbird: Efficient homomorphic evaluation of symmetric ciphers in 3gpp by combining two modes of tfhe. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):530–573, 2024.

[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.

[WWL$^+$24a] Ruida Wang, Benqiang Wei, Zhihao Li, Xianhui Lu, and Kunpeng Wang. Tfhe bootstrapping: Faster, smaller and time-space trade-offs. In *Australasian Conference on Information Security and Privacy*, pages 196–216. Springer, 2024.

[WWL$^+$24b] Ruida Wang, Yundi Wen, Zhihao Li, Xianhui Lu, Benqiang Wei, Kun Liu, and Kunpeng Wang. Circuit bootstrapping: faster and smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 342–372. Springer, 2024.

[XZD$^+$23] Binwu Xiang, Jiang Zhang, Yi Deng, Yiran Dai, and Dengguo Feng. Fast blind rotation for bootstrapping fhes. In *Annual International Cryptology Conference*, pages 3–36. Springer, 2023.

[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.