Article

Not peer-reviewed version

Multi-Group Fully Homomorphic Encryption Scheme Based on LWE and NTRU

Yongheng Li , Jing Wen , Shaoling Liang , Fanqi Kong , Baohua Huang *

Posted Date: 8 January 2026

doi: 10.20944/preprints202601.0638.v1

Keywords: multi-group homomorphic encryption; parallelism; secure multi-party computation; secret sharing; bootstrapping

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

Multi-Group Fully Homomorphic Encryption Scheme Based on LWE and NTRU

Yongheng Li [1], Jing Wen [2], Shaoling Liang [2], Fanqi Kong [1] and Baohua Huang [1,*]

[1] School of Computer and Electronic Information, Guangxi University, Nanning 530004, China
[2] Guangxi Zhuang Autonomous Region Information Center, Nanning 530000, China
[*] Correspondence: bhhuang66@gxu.edu.cn

Abstract

Multi-Group Homomorphic Encryption (MGHE) is a pivotal advance in secure multi-party computation, integrating merits of Multi-Party Homomorphic Encryption (MPHE) and Multi-Key Homomorphic Encryption (MKHE) to eliminate MPHE's fixed-party limitation and mitigate MKHE's ciphertext expansion from dynamic enrollment. However, the efficient single-key FINAL scheme cannot extend to multi-party scenarios, due to the challenge of defining valid multiplication for vector NTRU ciphertexts, which hinders its use in multi-group bootstrapping and curbs efficiency. To address this, additive secret sharing is adopted to convert vector NTRU ciphertext multiplication into secret share multiplication, enabling shared bootstrapping key generation within groups. For the first time, a multi-group ciphertext bootstrapping algorithm based on LWE and NTRU is proposed. Bootstrapping tasks are decomposed for parallel processing, and a hybrid product algorithm is designed to aggregate subtask outputs, boosting multi-group bootstrapping speed to match that of single-key ciphertexts. Noise accumulation is analyzed, with 100-bit and 128-bit security parameter sets selected for validation. Experiments show that 30/50-party multi-group bootstrapping takes only 1.87/2.58 seconds respectively.

Keywords: multi-group homomorphic encryption; parallelism; secure multi-party computation; secret sharing; bootstrapping

---

1. Introduction

Homomorphic encryption (HE) is a cryptographic technology that enables computations to be performed directly on encrypted data. Dubbed the "holy grail" of cryptography, HE holds enormous application potential in the field of information security. Based on the scope of homomorphic computations supported, HE can be categorized into Partial Homomorphic Encryption (PHE), Leveled Homomorphic Encryption (LHE), and Fully Homomorphic Encryption (FHE). Early research on HE focused on PHE, with representative schemes including the RSA scheme proposed by Rivest et al. [2] and the Paillier encryption scheme [3] developed later. The RSA scheme exhibits multiplicative homomorphism—decrypting the product of two ciphertexts yields the product of the corresponding plaintexts. In contrast, the Paillier scheme possesses additive homomorphism—decrypting the product of two ciphertexts results in the sum of the corresponding plaintexts. However, PHE schemes are highly restrictive as they only support either additive or multiplicative homomorphism. Decades after the concept of "homomorphic encryption" was proposed, Gentry [4] introduced a framework for constructing FHE, shifting academic focus from PHE to LHE and FHE. LHE supports both additive and multiplicative operations on ciphertexts, and the resulting new ciphertexts are encryptions of the corresponding plaintext sums or products. Nevertheless, LHE does not allow unlimited homomorphic computations—if the noise accumulated after each computation exceeds the scheme's tolerance threshold, the ciphertext will fail to decrypt. Gentry's FHE framework consists of LHE and bootstrapping. The core idea of bootstrapping is to execute a homomorphic decryption circuit on the ciphertext before

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

the noise reaches the threshold. As long as the noise of the bootstrapped ciphertext remains sufficiently low to support one additional homomorphic operation, LHE can be transformed into FHE.

Existing bootstrapping methods primarily fall into three categories. The first category is tailored for BGV/BFV schemes [5–8], leveraging the Chinese Remainder Theorem (CRT) over polynomial rings. Its core idea involves re-encrypting the original ciphertext and then decrypting the inner-layer ciphertext of the new ciphertext. The second category is designed for the CKKS scheme [9]. Since CKKS ciphertexts treat noise as part of the plaintext, the decryption function is replaced by an approximate modulus function, with the remaining steps closely resembling those of the first category. The third category is the bootstrapping based on blind rotation in TFHE schemes [10,11]. Unlike the previous two, blind rotation utilizes the negative cyclic property of the polynomial ring $\mathbb{Z}_Q[X]/(X^N+1)$. By iteratively multiplying the rotation polynomial with key-related monomials, specific coefficients of the rotation polynomial are rotated to the constant term in the encrypted state, thereby achieving homomorphic decryption. In TFHE, blind rotation consists of two types of ciphertexts (RLWE and RGSW) and their "external product". Given the large size of these two ciphertext types, Bonte et al. [12] proposed the NTRU-based FINAL scheme to reduce the size of bootstrapping keys. The FINAL scheme replaces RLWE and RGSW ciphertexts with scalar NTRU and vector NTRU ciphertexts, respectively. Since scalar NTRU ciphertexts consist of only one polynomial, and the size of vector NTRU ciphertexts is half that of RGSW ciphertexts, the NTRU-based bootstrapping algorithm not only offers superior space complexity but also requires fewer polynomial multiplications for executing the "external product" compared to TFHE.

The primary application of FHE is outsourcing computation, where data owners can encrypt their data and entrust it to untrusted computing service providers without revealing sensitive information. However, in secure multi-party computation (SMPC) scenarios, there is a need to compute data from multiple sources, rendering single-key FHE inadequate. To meet this demand, researchers have extended FHE to multi-party settings, leading to two main technical routes: static Multi-Party Homomorphic Encryption (MPHE) [13–15] and dynamic Multi-Key Homomorphic Encryption (MKHE) [16–18]. MPHE requires parties to be fixed prior to computation, with various keys generated collectively by all parties. Since encryption, decryption, and bootstrapping are performed under the same key, its computational efficiency is independent of the number of parties, ensuring high efficiency but limited flexibility. In contrast, MKHE does not require pre-computation negotiation among parties—each party generates its own key and participates in encryption, decryption, and bootstrapping without disclosing any key-related information. However, the size of multi-key ciphertexts is positively correlated with the number of parties, resulting in high flexibility but low computational efficiency. To balance flexibility and efficiency, Kwak et al. [19] proposed a new HE primitive called Multi-Group Homomorphic Encryption (MGHE). MGHE treats ciphertexts of MPHE as single-key ciphertexts under a joint public key, while ciphertexts encrypted under joint public keys of different groups are computed jointly in a multi-key manner. In the initial MGHE construction, bootstrapping adheres to the first, second, and third methods based on RLWE and RGSW, lacking support for NTRU-based blind rotation bootstrapping. To harness the advantages of the FINAL scheme and enhance the practicality of MGHE, this study designs an NTRU-based bootstrapping algorithm for multi-group LWE ciphertexts. Specifically, the main contributions of this paper are as follows:

1. A secret sharing-based bootstrapping key generation algorithm is proposed. The FINAL scheme lacks an asymmetric encryption variant, and existing NTRU-based blind rotation algorithms [20,21] employ symmetrically encrypted bootstrapping keys. To ensure compatibility between the proposed bootstrapping algorithm and NTRU-based blind rotation while maintaining low noise levels in bootstrapping keys, additive secret sharing is introduced in the offline phase before bootstrapping. parties within the same group collectively negotiate and generate bootstrapping keys through operations such as addition and multiplication of secret shares.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

2. A multi-group hybrid product algorithm is proposed. To parallelize bootstrapping, the core steps of bootstrapping are divided into multiple subtasks by group. Each subtask independently performs blind rotation using the corresponding group bootstrapping key, and the outputs of the subtasks are aggregated using the multi-group hybrid product algorithm.
3. Secure parameter sets are determined and the effectiveness is verified through experiments. Since the FINAL scheme is based on NTRU, "overstretched" parameters render the scheme vulnerable to sublattice attacks. By analyzing the upper bound of noise, the parameters of the proposed scheme are ensured to remain within a secure range using LWE [22] and NTRU estimators [23]. Experiments are conducted under two different parameter sets with the number of groups ranging from 2 to 8, validating the effectiveness of the proposed scheme.

2. Related Work

In MGHE, plaintexts within the same group are encrypted using a joint public key in the form of MPHE, while ciphertexts encrypted under joint public keys of different groups are computed in a multi-key manner. Thus, MGHE can be regarded as an extension of MPHE to multi-key scenarios. Current mainstream single-key HE schemes include BGV [24], BFV [25], CKKS [26], and FHEW [27]/TFHE [10,11]. Both MPHE and MKHE schemes are constructed on top of these single-key schemes.

2.1. Multi-Party Homomorphic Encryption Schemes

The earliest MPHE scheme was constructed by Asharov et al. [28] based on the LWE-based BGV scheme. In this scheme, parties independently generate public keys, broadcast them, and compute the joint public key locally by summing the received public keys from other parties. Similarly, Mouchet et al. [14] constructed an RLWE-based MPHE on the BFV scheme, which can be easily extended to BGV and CKKS schemes. FHEW/TFHE-type schemes employ two layers of ciphertexts: the first layer is basic LWE ciphertexts, and the second layer is RGSW ciphertexts for homomorphic decryption. Unlike BGV, BFV, and CKKS schemes, TFHE encodes the LWE secret key into the monomial X and encrypts it into the second-layer ciphertext as the bootstrapping key, achieving homomorphic decryption through the blind rotation algorithm. Thus, constructing a multi-party TFHE scheme requires generating multi-party bootstrapping keys. Lee et al. [29] realized the generation of multi-party blind rotation keys by leveraging homomorphic multiplication between RGSW ciphertexts, constructing a feasible multi-party bootstrapping algorithm. Subsequently, Park et al. [30] built the first multi-party TFHE scheme based on this bootstrapping algorithm.

2.2. Multi-Key Homomorphic Encryption Schemes Based on Blind Rotation

As the fastest method for bootstrapping a single ciphertext, blind rotation is not only applicable to TFHE schemes but also compatible with all (R)LWE-based encryption schemes. Thus, blind rotation is a key technology for both MPHE and MKHE schemes. Chen et al. [18] first extended TFHE to multi-key scenarios, proposing the CCS scheme. This scheme uses the hybrid product between multi-key RLWE ciphertexts and single-key Uni-Enc ciphertexts to replace the external product between multi-key RLWE and multi-key RGSW ciphertexts, achieving lower time and space overhead. Based on the hybrid product algorithm proposed by Chen et al., Kwak et al. [31] constructed a generalized external product algorithm and designed a parallelizable multi-key LWE ciphertext bootstrapping algorithm (KMS scheme). The initial blind rotation algorithm, which used RLWE and RGSW ciphertexts as inputs, was later improved in the FINAL scheme by Bonte et al. [12]. Based on the NTRU problem, their key modification was to replace these ciphertexts with the more compact scalar NTRU and vector ciphertexts, respectively. Since NTRU scalar and vector ciphertexts are smaller in size than RLWE and RGSW ciphertexts, and fewer polynomial multiplications are required for their homomorphic multiplication, the overhead of blind rotation is reduced. Building on the FINAL scheme, Xu et al. [32] proposed an MKHE scheme that achieves approximately 5x speedup compared to the CCS scheme for two parties. However, this scheme controls noise accumulation during homomorphic computations

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

by fixing the Hamming weight of NTRU keys, which Kim et al. [33] pointed out compromises the scheme's security, making it unsuitable for practical applications. Park et al. [34] applied the hybrid product algorithm to the key switching key generation protocol, avoiding the invocation of the hybrid product during blind rotation, resulting in significant performance improvements over the CCS and KMS schemes. Nevertheless, it still does not support parallel execution.

3. Preliminaries

3.1. Notions

Boldface letters denote vectors (e.g., $\boldsymbol{a}$), and the inner product between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is denoted as$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$. $\mathbb{Z}$ denotes the ring of integers. For a given positive integer q, $\mathbb{Z}_q$ denotes the quotient ring $\mathbb{Z}/q\mathbb{Z}$, whose elements are integers in the interval $[-\frac{q}{2}, \frac{q}{2})$. Similarly, given positive integers Q and N (where N is a power of 2), $R = \mathbb{Z}[X]/(X^N + 1)$ denotes the 2N-th cyclotomic ring, and the quotient ring $R_Q = R/QR = \mathbb{Z}_Q[X]/(X^N + 1)$ consists of integer polynomials with coefficients in $\mathbb{Z}_Q$. $e \leftarrow \chi$ indicates that the random variable e is sampled from the distribution χ. Similarly, $e = (e_1, \ldots, e_n) \leftarrow \chi^n$ denotes that the vector e is obtained by sampling n times independently from χ. For clarity, if e is a polynomial (even if its coefficients are sampled multiple times from a distribution), the former notation is still used. σ and σ^2 denote the standard deviation and variance of a distribution, respectively. For a random variable a, $\mathrm{Var}(a)$ denotes its variance if $a \in \mathbb{Z}$, or the variance of its coefficients if $a \in R$. For a ciphertext c, $\mathrm{err}(c)$ denotes the ciphertext noise, and $\mathrm{Var}(\mathrm{err}(c))$ denotes the variance of the ciphertext noise. $\lfloor \cdot \rceil$, $\lceil \cdot \rceil$, and $\lfloor \cdot \rfloor$ denote the rounding function, ceiling function, and floor function, respectively. In secret sharing schemes, for a secret s, $[s]_i$ denotes the secret share belonging to the i-th party, and $[s] = \{[s]_i\}$ denotes the set of secret shares.

3.2. Hard Problems

Definition 1. *Decisional LWE Problem [35].*

Given positive integers q, n and a distribution χ over $\mathbb{Z}$, the decisional LWE problem is to distinguish between:

1. Randomly sampled pairs $(x, \boldsymbol{y}) \in \mathbb{Z}_q^{n+1}$;
2. Pairs $(b = -\sum_{i=1}^{n} a_i s_i + e, \boldsymbol{a}) \in \mathbb{Z}_q^{n+1}$, where $\boldsymbol{a}$ is uniformly sampled from $\mathbb{Z}_q^n$, a_i is the i-th element of $\boldsymbol{a}$, $\boldsymbol{s}$ is uniformly sampled from $\mathbb{Z}^n$, s_i is the i-th element of $\boldsymbol{s}$, and e is sampled from χ.

Definition 2. *Decisional RLWE Problem [36].*

Given positive integers Q, N, the polynomial ring $R = \mathbb{Z}[X]/(X^N + 1)$, $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$, and a distribution χ over R, the decisional RLWE problem is to distinguish between:

1. Randomly sampled pairs $(x, y) \in R_Q^2$;
2. Pairs $(b = -as + e, a) \in R_Q^2$, where a, s are uniformly sampled from R_Q, and e is sampled from χ.

Definition 3. *Decisional NTRU Problem [12].*

Given positive integers Q, N, the polynomial ring $R = \mathbb{Z}[X]/(X^N + 1)$, $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$, and a distribution χ over R, let $f \leftarrow \chi$, $e \leftarrow \chi$ with f invertible in R_Q. The decisional NTRU problem is to distinguish between:

1. A randomly sampled element $a \in R_Q$;
2. The element $\frac{e}{f} \in R_Q$.

Definition 4. *Decisional Vector NTRU Problem [37].*

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

Given positive integers Q, N, d, the polynomial ring $R = \mathbb{Z}[X]/(X^N + 1)$, $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$, and a distribution χ over R, let $f, e_1, \ldots, e_d \leftarrow \chi$ with f invertible in R_Q. The decisional vector NTRU problem is to distinguish between:

1. A randomly sampled vector $\boldsymbol{a} = (a_1, \ldots, a_d) \in R_Q^d$;
2. The vector $\left(\frac{e_1}{f}, \ldots, \frac{e_d}{f}\right) \in R_Q^d$.

3.3. Gadget Decomposition

Gadget decomposition is a technique for decomposing an integer in $(\mathbb{Z}_Q$ into multiple small integers in $\mathbb{Z}$. Given positive integers B, $d = \lceil \log_B Q \rceil$, and $a \in \mathbb{Z}_Q$, the gadget decomposition vector with base B is defined as $\mathbf{g}_B = (B^0, B^1, \ldots, B^{d-1}) \in \mathbb{Z}_Q^d$. The decomposition function of the integer a with base B is given by:

$$\mathfrak{g}_B^{-1}(a) = (a_1, a_2, \ldots, a_d) \in \mathbb{Z}^d, \tag{1}$$

where $\sum_{i=1}^d a_i \cdot B^{i-1} = a \mod Q$. For brevity, $\langle \mathfrak{g}_B^{-1}(a), \mathbf{g}_B \rangle$ is denoted as $\mathfrak{g}_B^{-1}(a) \cdot \mathbf{g}_B$. Gadget decomposition is not limited to $\mathbb{Z}_Q$ but can also be extended to the polynomial ring R_Q. For a polynomial $a(X) \in R_Q$, it is treated as a coefficient vector, and the above conclusion holds by decomposing each element of the vector individually.

3.4. FINAL Scheme

The NTRU problem serves as a crucial hard problem for constructing post-quantum FHE schemes, and its parameter selection directly impacts the security of the cryptographic scheme. Schemes built on NTRU are vulnerable to sublattice attacks if "overstretched" parameters are used [38]. To avoid overstretched parameters while ensuring the correctness of homomorphic operations, Bonte et al. proposed the FINAL scheme [12]. Subsequently, Xiang et al. [20] modified the ciphertext form of the scheme to construct their ring isomorphism-based blind rotation algorithm. The modified FINAL scheme is described as follows:

- FINAL.Setup(1^λ): Input a security parameter λ, generate security-compliant parameters including the NTRU polynomial modulus Q, polynomial degree N, Gadget decomposition base B, key distribution χ_{nk} over R, and noise distribution χ_{ne} over R. Output $Params = (Q, N, B, \chi_{nk}, \chi_{ne})$.
- FINAL.KeyGen(): Input the parameters generated by the Setup algorithm, select an invertible polynomial $f \in R$ from χ_{nk}, and output the NTRU key f.
- FINAL.ScalarEnc(m, f): Input a plaintext $m \in R_Q$ and the NTRU key f, randomly select noise $e \leftarrow \chi_{ne}$, set $\Delta = \frac{Q}{4}$ as the scaling factor, and output the NTRU scalar ciphertext

$$c = \frac{e + \Delta m}{f} \in R_Q. \tag{2}$$

- FINAL.VectorEnc(m, f): Input a plaintext $m \in R_Q$ and the NTRU key f, randomly select noise $e_1, \ldots, e_d \leftarrow \chi_{ne}$ where $d = \lceil \log_B Q \rceil$, and output the NTRU vector ciphertext

$$\boldsymbol{c} = \left(\frac{e_1}{f} + B^0 \cdot m, \ldots, \frac{e_d}{f} + B^{d-1} \cdot m\right) \in R_Q^d. \tag{3}$$

To implement the CMux gate, the TFHE scheme defines the external product between TRLWE and TRGSW ciphertexts. Similarly, the FINAL scheme defines the external product between scalar NTRU and vector NTRU ciphertexts, and proves the upper bound of the noise variance for this external product operation.

Definition 5. *NTRU External Product*

Let $c = NTRU_{Q,f,\Delta}(m)$ denote a scalar NTRU ciphertext encrypting the plaintext m with modulus Q, scaling factor Δ, and NTRU key f. Let $\boldsymbol{c} = NTRU'_{Q,f,B}(m)$ denote a vector NTRU ciphertext

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026

doi:10.20944/preprints202601.0638.v1

encrypting the plaintext m with modulus Q, Gadget decomposition base B, and NTRU key f. The external product is defined as

$$\boxdot : NTRU_{Q,f,\Delta} \times NTRU'_{Q,f,B} \rightarrow NTRU_{Q,f,\Delta}, \tag{4}$$

where

$$c \boxdot \boldsymbol{c} = \langle \mathfrak{g}_B^{-1}(c), \boldsymbol{c} \rangle. \tag{5}$$

Lemma 1. *Noise of NTRU External Product*

Given positive integers Q, N, B, d, a scalar NTRU ciphertext $c = \frac{e+\Delta m}{f} \in R_Q$, and a vector NTRU ciphertext $\boldsymbol{c} = \left(\frac{e_1}{f} + B^0 \cdot m', \ldots, \frac{e_d}{f} + B^{d-1} \cdot m' \right) \in R_Q^d$, let $err(c)$ and $err(\boldsymbol{c})$ denote the noise of c and $\boldsymbol{c}$, respectively, and $Var(err(c))$ and $Var(err(\boldsymbol{c}))$ denote their noise variances. Then, the noise variance of the ciphertext $c' = c \boxdot \boldsymbol{c}$ satisfies

$$Var(err(c')) \leq dN \cdot \frac{B^2}{12} \cdot Var(err(c)) + Var(err(\boldsymbol{c})). \tag{6}$$

3.5. XZD Blind Rotation

The first automorphism-based blind rotation algorithm was proposed by Lee et al. [29] (LMKC scheme). Unlike the AP scheme [39], which achieves exponent accumulation through multiplication operations, the LMKC scheme leverages the automorphism map $X \rightarrow X^t$ to implement scalar multiplication on exponents, thereby significantly reducing the number of multiplication operations in the accumulator. Subsequently, the XZD scheme proposed by Xiang et al. [20] further shortens the time required for key switching after multiplication in the accumulator by adopting scalar and vector NTRU ciphertexts, effectively reducing the overall computational overhead of blind rotation.

The order of the monomial X over the 2N-th cyclotomic polynomial ring $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$ is 2N, i.e., $X^{2N} = 1 \mod (X^N + 1)$. Thus, for a positive integer t, multiplying any element $r(X) \in R$ by the monomial X^t results in a regular cyclic left or right shift of the polynomial's coefficients. Blind rotation leverages this property to rotate specific coefficients to the leading term. Formally, given an element $r(X) = \sum_{i=0}^{q-1} i \cdot X^{-i} \in R$, compute the rotation polynomial

$$r\left(X^{\frac{2N}{q}}\right) = \sum_{i=0}^{q-1} i \cdot X^{-\frac{2N}{q} i}. \tag{7}$$

For a ciphertext $c = (b, a_1, \ldots, a_n) \in \mathbb{Z}_q^{n+1}$ satisfying $\Delta \cdot m + e = b + \sum_{i=1}^{n} a_i s_i$ (where Δ is the LWE scaling factor, q is the LWE ciphertext modulus, and q divides 2N), the goal of blind rotation is to compute the following in the encrypted state

$$r\left(X^{\frac{2N}{q}}\right) \cdot X^{\frac{2N}{q} \cdot (b + \sum_{i=1}^{n} a_i s_i)} = (\Delta \cdot m + e) + \sum_{i \neq \Delta m + e} i \cdot X^{\frac{2N}{q}(\Delta \cdot m + e - i)}. \tag{8}$$

Before performing blind rotation, the XZD scheme preprocesses the ciphertext to be bootstrapped $c = (b, a_1, \ldots, a_n)$ by computing

$$c' = \left(\frac{2N}{q} b + 1, \frac{2N}{q} a_1 + 1, \ldots, \frac{2N}{q} a_n + 1 \right) \in \mathbb{Z}_q^{n+1}, \tag{9}$$

ensuring the existence of the automorphism. Subsequently, the rotation polynomial is encrypted into a scalar NTRU ciphertext, and the monomial X^{s_i} (related to the i-th component of the LWE secret key) is encrypted into a vector NTRU ciphertext. During blind rotation, X^{s_i} is iteratively multiplied into the rotation polynomial via the external product between the two types of ciphertexts, followed

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

by key switching based on automorphism to multiply the ciphertext component$a_i$into the rotation polynomial.

The XZD blind rotation consists of two algorithms: the Blind Rotation Key Generation (BRKGen) algorithm and the Blind Rotation Evaluation (BREval) algorithm. Let $Params = (Q, N, B_{br}, \chi_{nk}, \chi_{ne})$ be the public parameters output by $FINAL.Setup$, and $d_{br} = \lceil \log_{B_{br}} Q \rceil$. Define the set $S = \left\{ \frac{2N}{q} \cdot i + 1 \mid 1 \leq i \leq q-1 \right\}$, and let $LWE_{q,\mathbf{s}}(m) = (b, a_1, \ldots, a_n) \in \mathbb{Z}_q^{n+1}$ denote the LWE ciphertext encrypting the plaintext m with modulus q and secret key $\mathbf{s}$. The two algorithms are described as follows.

- BRKGen(): Given an LWE secret key $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n$ and an NTRU key $f \in R_Q$, the algorithm outputs $EVK = \{evk_j \mid 1 \leq j \leq n+1\} \cup \{nksvk_j \mid j \in S\}$, where
  1. $evk_1 = NTRU'_{Q,f,B_{br}}\left(\frac{X^{s_1}}{f}\right)$, $evk_i = NTRU'_{Q,f,B_{br}}(X^{s_i})$ for $2 \leq i \leq n$, $evk_{n+1} = NTRU'_{Q,f,B_{br}}\left(X^{-\sum_{i=1}^{n} s_i}\right)$,
  2. $nksvk_j = NTRU'_{Q,f,B_{br}}\left(\frac{f(X^j)}{f(X)}\right)$.
- BREval($(b, \mathbf{a}), r, EVK, \Delta$): Input an LWE ciphertext $LWE_{q,\mathbf{s}}(m) = (b, a_1, \ldots, a_n) \in \mathbb{Z}_q^{n+1}$, a rotation polynomial r, the evaluation key EVK generated by BRKGen, and the NTRU scaling factor Δ. The algorithm computes and outputs

$$ACC = NTRU_{Q,f,\Delta}\left(r\left(X^{\frac{2N}{q}}\right) \cdot X^{\frac{2N}{q} \cdot (b + \sum_{i=1}^{n} a_i s_i)}\right). \tag{10}$$

The detailed steps are presented in Algorithm 1.

Algorithm 1 BREval($(b, \mathbf{a}), r, EVK, \Delta$) (adapted from [20]).

Require:
An LWE ciphertext $LWE_{q,\mathbf{s}}(m) = (b, a_1, ..., a_n) \in \mathbb{Z}_q^{n+1}$
A Rotation Polynomial $r(X)$
Evaluation Key $EVK = (\{evk_j\}_{1 \leq j \leq n+1}, \{nksk_j\}_{j \in S})$
Scaling factor Δ
Ensure:
A scalar NTRU ciphertext $NTRU_{Q,f,\Delta}(r(X^{\frac{2N}{q}}) \cdot X^{\frac{2N}{q} \cdot (b + \sum_{i=1}^{n} a_i s_i)})$
1: **for** $i = 1$ to n **do**
2: $w_i \leftarrow \frac{2N}{q} a_i + 1$
3: $w'_i \leftarrow w_i^{-1} \mod 2N$
4: **end for**
5: $w'_{n+1} \leftarrow 1$
6: $ACC \leftarrow \Delta \cdot r(X^{\frac{2N}{q} w'_1}) \cdot X^{\frac{2N}{q} b w'_1}$
7: **for** $i = 1$ to n **do**
8: $ACC \leftarrow ACC \boxdot evk_i$
9: **if** $w_i w'_i \neq 1$ **then**
10: $c(X) \leftarrow ACC$
11: $ACC \leftarrow c(X^{w_i w'_{i+1}}) \boxdot nksk_{w_i w'_{i+1}}$
12: **end if**
13: **end for**
14: $ACC \leftarrow ACC \boxdot evk_{n+1}$
15: return ACC

3.6. Additive-Only Multi-Party RLWE-Based Homomorphic Encryption

The core algorithms of the multi-party BFV scheme proposed by Mouchet et al. [40] can be combined to form an additive-only (supporting scalar multiplication) multi-party RLWE homomorphic encryption scheme (AHE).

- AHE.Setup(1^λ): Input a security parameter λ, output security-compliant public parameters including the RLWE ciphertext modulus Q, plaintext modulus t, polynomial degree N, secret key

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

distribution χ_{rk} over R, noise distribution χ_{re} over R, and a common reference polynomial $p \in R$. Output $Params = (Q, t, N, \chi_{rk}, \chi_{re}, p)$.

- AHE.SKeyGen(): Each party $\mathcal{P}_i \in \mathcal{G}$ selects an RLWE secret key $z_i \leftarrow \chi_{rk}$.
- AHE.PKeyGen(z_i): Each party $\mathcal{P}_i \in \mathcal{G}$ selects noise $e_i \leftarrow \chi_{re}$, computes $r = -p \cdot z_i + e_i$, and sets $pk = (r, p) \in R_Q^2$.
- AHE.JointPKeyGen($\{z_i\}_{i \in \mathcal{G}}$): Input the public keys $pk = (r_i, p)$ of all parties, compute $\widetilde{r} = \sum_{i \in \mathcal{G}} r_i$, and output the joint RLWE public key $rjpk = (\widetilde{r}, p) \in R_Q^2$.
- AHE.Enc(m): Input a plaintext $m \in R_t$, select noise $e1, e_2 \leftarrow \chi_{re}$, randomly select a temporary secret polynomial $g \leftarrow \chi_{rk}$, compute $b = \widetilde{r} \cdot g + \frac{Q}{t} \cdot m + e_1$ and $a = p \cdot g + e_2$, and output the ciphertext $c = (b, a) \in R_Q^2$.
- AHE.Dec($c, \{z_i\}_{i \in G}$): Input a ciphertext $c = (b, a)$ encrypted under the joint RLWE public key $rjpk$ and the RLWE secret keys $\{z_i\}_{i \in G}$ of all parties, compute $m = \lfloor \frac{t \cdot (b + \sum_{i \in \mathcal{G}} a \cdot z_i)}{Q} \rceil \in R_t$.
- AHE.Add(c, c'): Input two multi-party RLWE ciphertexts $c = (b, a)$ and $c' = (b', a')$ encrypting plaintexts $m, m' \in R_t$ under the same public key, compute and output $\bar{c} = (b + b', a + a') \in R_Q^2$.
- AHE.ScalarMult(β, c): Input a scalar polynomial $\beta \in R_t$ and a multi-party RLWE ciphertext $c = (b, a)$, compute and output $\bar{c} = (\beta \cdot b, \beta \cdot a) \in R_Q^2$.

4. Basic Multi-Group LWE-Based Homomorphic Encryption without Bootstrapping

This section first introduces the basic LWE-based multi-group homomorphic encryption scheme, whose network architecture is shown in Figure 1.

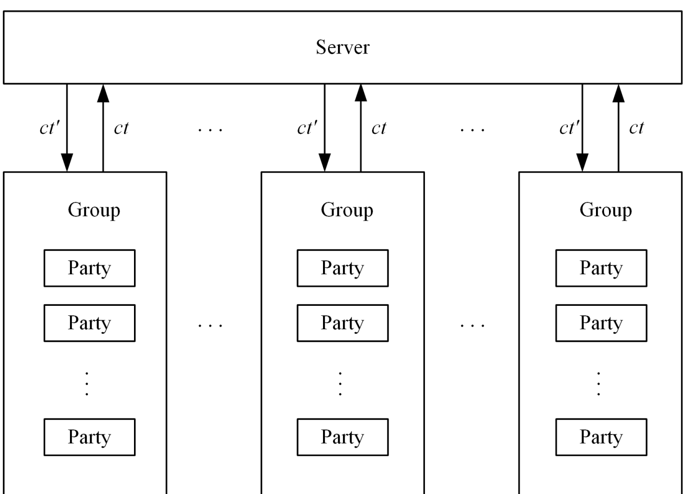

Figure 1. Party structure of multi-group homomorphic encryption scheme.

Parties in each group encrypt plaintexts using the group joint public key to obtain ciphertexts ct, which are then sent to the server. After receiving the ciphertexts, the server performs homomorphic computation and bootstrapping, and returns the new ciphertext ct' to be decrypted. Under the MGHE framework, our basic scheme includes two sets of algorithms: SKeyGen, PKeyGen, JointPKeyGen and Enc, which are dedicated to intra-group interactive computation, as well as Setup, NAND and Dec, which are for inter-group computation. In addition, to support asymmetric encryption while maintaining a small ciphertext modulus, we distinguish between the encryption modulus q' and the bootstrapping modulus q. The encryption modulus q' is much larger than q, and ciphertexts under the encryption modulus q' can tolerate greater noise. After encrypting a plaintext, a party needs to switch the ciphertext modulus to the bootstrapping modulus q through the ModulusSwitch algorithm to perform subsequent NAND and bootstrapping algorithms.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

Let $\mathcal{G}$ be a set of parties, $\mathcal{G}_i$ denote the i-th group, j be the unique index of the party in the i-th group within $\mathcal{G}_i$, and $\mathcal{P}_{i,j}$ denote the j-th party in the i-th group. Our MGHE includes the following algorithms.

- MGHE.Setup(1^λ): Takes a security parameter λ as input, generates global parameters meeting security requirements, including LWE encryption modulus q', LWE bootstrapping modulus q, LWE dimension n, LWE secret key distribution χ_{lk} over $\mathbb{Z}$, noise distribution χ_{le} over $\mathbb{Z}$, and a common reference matrix $M \in \mathbb{Z}^{n\times n}$. Outputs $Params = (q', q, n, \chi_{lk}, \chi_{le}, M)$.
- MGHE.SKeyGen(): Parties $\mathcal{P}_{i,j}$ in each group independently generate a private LWE secert key $\boldsymbol{s} \leftarrow \chi^n_{lk}$, and output the LWE secret key $\boldsymbol{s} \in \mathbb{Z}^n$.
- MGHE.PKeyGen($\boldsymbol{s}$): Party $\mathcal{P}_{i,j}$ selects noise $\boldsymbol{e} \leftarrow \chi^n_{le}$, computes $\boldsymbol{r} = -M\cdot \boldsymbol{s} + \boldsymbol{e} \in \mathbb{Z}^n_{q'}$, and sets $pk = (\boldsymbol{r}, M) \in \mathbb{Z}^n_{q'} \times \mathbb{Z}^{n\times n}_{q'}$.
- MGHE.JointPKeyGen($\{pk_j\}_{j\in\mathcal{G}}$): Takes the LWE public keys $pk_j = (\boldsymbol{r}_j, M)$ of each party in a group $\mathcal{G}$, computes $\widetilde{\boldsymbol{r}} = \sum_{j\in G} \boldsymbol{r}_j$, and outputs the LWE joint public key $jpk = (\widetilde{\boldsymbol{r}}, M) \in \mathbb{Z}^n_{q'} \times \mathbb{Z}^{n\times n}_{q'}$.
- MGHE.Enc(jpk, m): Takes a plaintext bit $m \in \{0,1\}$ and the LWE joint public key jpk of a group as input, randomly selects a temporary secret vector $\boldsymbol{v} \leftarrow \chi^n_{lk}$, noise $e_1 \leftarrow \chi_{le}$, $\boldsymbol{e}_2 \leftarrow \chi^n_{le}$, computes $b = \langle \widetilde{\boldsymbol{r}}, \boldsymbol{v}\rangle + \frac{q'}{4} m + e_1 \in \mathbb{Z}_{q'}$, $\boldsymbol{a} = \boldsymbol{v}^T \cdot M + \boldsymbol{e}_2 \in \mathbb{Z}^n_{q'}$, and outputs the LWE ciphertext $c = (b, \boldsymbol{a}) \in \mathbb{Z}^{n+1}_{q'}$ encrypting the plaintext m under the joint LWE public key jpk.
- MGHE.ModulusSwitch($\boldsymbol{c}, q$): Takes the LWE ciphertext $\boldsymbol{c} = (b, a_1, \ldots, a_n) \in \mathbb{Z}^{n+1}_{q'}$ generated by the MGHE.Enc algorithm and the LWE bootstrapping modulus q for bootstrapping, computes and outputs $\boldsymbol{c}' = \left(\lceil \frac{q}{q'} \cdot b \rfloor, \lceil \frac{q}{q'} \cdot a_1 \rfloor, \ldots, \lceil \frac{q}{q'} \cdot a_n \rfloor\right) \in \mathbb{Z}^{n+1}_q$.
- MGHE.Dec($\boldsymbol{c}, \{\boldsymbol{s}_{i,j}\}_{1\le i\le k, j\in\mathcal{G}_i}$): Takes the multi-group LWE ciphertext $c = (b, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k)$ and the LWE secret keys $\boldsymbol{s}_{i,j}$ of each party in each group as input, computes $u_{i,j} = \langle \boldsymbol{a}_i, \boldsymbol{s}_{i,j}\rangle$ and outputs $m = \lfloor \frac{2(b+\sum_{i=1}^k \sum_{j\in\mathcal{G}_i} u_{i,j})}{q} \rceil \in \mathbb{Z}_2$.
- MGHE.NAND($\boldsymbol{c}, \boldsymbol{c}'$): Takes two multi-group LWE ciphertexts $\boldsymbol{c} = (b, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k) \in \mathbb{Z}^{kn+1}_q$ and $\boldsymbol{c}' = (b', \boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_k) \in \mathbb{Z}^{kn+1}_q$ encrypted under the same set of public keys $\{jpk_i\}_{1\le i\le k}$, which encrypt plaintext bits m, m' respectively. For $1 \le i \le k$, let $\boldsymbol{a}_i = (a_{i,1}, \ldots, a_{i,n})$ and $\boldsymbol{a}'_i = (a'_{i,1}, \ldots, a'_{i,n})$, computes $\widetilde{\boldsymbol{a}}_i = (a_{i,1} - a'_{i,1}, \ldots, a_{i,n} - a'_{i,n}) \in \mathbb{Z}^n_q$, and outputs the ciphertext $c = (\frac{5q}{8} - b - b', \widetilde{\boldsymbol{a}}_1, \ldots, \widetilde{\boldsymbol{a}}_k) \in \mathbb{Z}^{kn+1}_q$ encrypting the plaintext m NAND m'.

RLWE is a ring variant of LWE. The above multi-group homomorphic encryption scheme can be constructed based on RLWE. Since the RLWE-based multi-group scheme operates on the ring R_Q, the main difference between the RLWE-based multi-group homomorphic encryption scheme and the above LWE-based scheme lies in the encryption and decryption algorithms. Let the RLWE joint public key be $jpk = (r, p) \in R^2_Q$. In the encryption algorithm, we compute $b = r \cdot v + \frac{Q}{4} \cdot m + e_1$ and $a = p \cdot v + e_2$, where $v \in R_Q$ is a secret polynomial, and e_1, e_2 are selected from a distribution over a certain polynomial ring. We denote $\boldsymbol{c} = (b, a_1, \ldots, a_k) \in R^{k+1}_Q$ as the multi-group RLWE ciphertext encrypted by the RLWE joint public keys of k groups, with the corresponding secret keys $\{z_{i,j}\}_{1\le i\le k, j\in\mathcal{G}_i}$. The ciphertext satisfies $b + \sum_{i=1}^k \sum_{j\in\mathcal{G}_i} a_i \cdot z_{i,j} \approx \Delta \cdot m$, where Δ is a scaling factor.

The encryption scheme in this paper is based on Boolean circuits. Since the NAND gate has functional completeness, we can use the NAND gate as the basic unit to construct any Boolean function. Since homomorphic computation is performed under the circuit model, the bootstrapping scheme introduced in this paper belongs to gate bootstrapping. That is, homomorphic decryption is performed immediately after computing a NAND gate, and the scaling factor is converted from $\frac{q}{2}$ to $\frac{q}{4}$ to support the next homomorphic NAND computation. After multiple gate bootstrapping operations, the decryption algorithm MGHE.Dec requires each party in each group to execute an interactive protocol called "distributed decryption" to ensure the security of the joint key. The current mainstream implementation is based on the smudging lemma in the AJL scheme [28]. Each party samples an super-polynomially large smudging noise $e^{(i,j)}_{smg}$ from a large noise distribution, computes $u_{i,j} + e^{(i,j)}_{smg}$ and broadcasts it to mask the intermediate results during the decryption process. To ensure

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

the correctness of decryption, the ciphertext modulus also needs to be set very large. However, in our multi-group ciphertext bootstrapping algorithm based on NTRU and LWE, since the NTRU ciphertext modulus Q cannot be super-polynomially large, this method is not applicable.

For such schemes with limited ciphertext modulus, the KLO scheme proposed by Kraitsberg et al. [41] constructs a secure three-party distributed decryption protocol using garbled circuits, and the scheme can be extended to more parties. Although this decryption protocol requires additional communication overhead, no additional noise needs to be introduced, so the parameters of the encryption scheme do not need to be adjusted.

5. Bootstrapping for Multi-Group LWE Ciphertext

This section introduces how to bootstrap multi-group LWE ciphertexts. Bootstrapping can be divided into two parts: bootstrapping key generation and bootstrapping execution. The generation of the bootstrapping key uses additive secret sharing technology to compute the bootstrapping key related to the joint NTRU key $F = f_1 f_2 \dots f_k$. In addition, in additive secret sharing, the secret can be recovered by simply summing the secret shares held by each party. Therefore, we can design a multi-group hybrid product algorithm suitable for multi-group ciphertexts to realize the multiplication between different types of ciphertexts, and further construct a parallelizable bootstrapping algorithm.

5.1. Generating Polynomial Multiplication Triples

The multiplication triple $(a, b, c = a \cdot b)$ was proposed by Beaver [42] to reduce the number of communication rounds in secret sharing multiplication, thereby converting communication overhead into precomputation overhead in the offline phase. Existing multiplication triple generation schemes mainly include the MASCOT scheme [43] based on Oblivious Transfer (OT) and the Low/High Gear scheme [44] based on Additive-only Homomorphic Encryption (AHE). However, the triple elements generated by these schemes are all values over the integer ring (field), which cannot be used for secret multiplication over the polynomial ring R_Q. Therefore, we design a multi-party multiplication triple generation algorithm TripleGen that supports polynomial secret multiplication according to the framework proposed in [45]. The algorithm comprises two sub-algorithms: the Setup sub-algorithm is used to initialize public parameters, and the EvalGen is used to generate triples.

- TripleGen.Setup(): Successively executes AHE.Setup, AHE.SKeyGen, AHE.PKeyGen, and AHE.JointPKeyGen, outputs public parameters $Params = (Q, t, N, \chi_{rk}, \chi_{re}, p)$, secret keys $\{z_i\}_{1 \le i \le k}$ of each party, and joint public key $rjpk = (r, p)$.
- TripleGen.EvalGen($Params, \{z_i\}_{1 \le i \le k}, rjpk$): Takes the public parameters $Params$ output by TripleGen.Setup, the secret keys $\{z_i\}_{1 \le i \le k}$ of each party, and the joint public key $rjpk$ as input, and outputs a random polynomial multiplication triple. The detailed specifications referred to Algorithm 2.

When performing decryption in the fifth step, we also need to use a distributed decryption protocol. The difference is that party $\mathcal{P}_1$ does not need to broadcast its partial decryption result, while other parties need to send their partial decryption results to $\mathcal{P}_1$. This asymmetric decryption method ensures that no party other than $\mathcal{P}_1$ knows the decryption result.

Let the secrets corresponding to the triple shares output by the algorithm be a, b, c. Since we need $c = a \cdot b$ to hold over the ring R_Q, the plaintext modulus t of AHE is required to be larger than the coefficients of c to ensure that the c output by the algorithm is not implicitly modulo t.

In the second step of the algorithm, party $\mathcal{P}_1$ broadcasts the ciphertext C, which encrypts $\sum_{i=1}^{k} a_i$, and the upper bound of its plaintext coefficients is k. In the third step, each party $\mathcal{P}_i$ computes the ciphertext C_i', which encrypts the plaintext $b_i \cdot \sum_{i=1}^{k} a_i$, with the upper bound of coefficients being kN. Similarly, the ciphertext C' in the fifth step encrypts $(\sum_{i=1}^{k} a_i) \cdot (\sum_{i=1}^{k} b_i)$, with the upper bound of plaintext coefficients being $k^2 N$. The ciphertext C'' encrypts $c_1 = (\sum_{i=1}^{k} a_i) \cdot (\sum_{i=1}^{k} b_i) - \sum_{i=2}^{k} c_i$, with the upper bound of its coefficients being $k^2 N + k - 1$. Therefore, to ensure correctness, it is required that $t > 2kN + k - 1$.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Algorithm 2 TripleGen($Params, \{z_i\}_{1\le i\le k}, rjpk$).

Require:
AHE public parameters $Params$.
Set of RLWE secret keys $\{z_i\}_{1\le i\le k}$ has each z_i belonging to $\mathcal{P}_i$.
Joint RLWE public key $rjpk$.
Ensure: Polynomial multiplication triple $([a],[b],[c])$.
1: For $1 \le i \le k$, party $\mathcal{P}_i$ randomly selects $a_i, b_i \in R_3$ from a symmetric three-point distribution (probability $\frac{1}{4}$ for taking values -1 or 1, and $\frac{1}{2}$ for taking value 0), computes $C_i = AHE.Enc(a_i)$ and sends it to $\mathcal{P}_1$.
2: $\mathcal{P}_1$ computes C=AHE.Add(AHE.Add($\dots$ AHE.Add(C_1, C_2), C_3) $\dots, C_k$) and broadcasts it to all parties.
3: For $1 \le i \le k$, party $\mathcal{P}_i$ computes
E_i=AHE.Enc(0),
C_i'=AHE.Add(AHE.ScalarMult(b_i, C), E_i),
then sends C_i' to $\mathcal{P}_1$.
4: For $2 \le i \le k$, party $\mathcal{P}_i$ randomly selects a polynomial $c_i \in R_3$ from the symmetric three-point distribution, computes C_i''=AHE.Enc($-c_i$) and sends it to $\mathcal{P}_1$.
5: $\mathcal{P}_1$ computes
C'=AHE.Add($\dots$AHE.Add(C_1', C_2') $\dots, C_k'$),
C''=AHE.Add(AHE.Add($\dots$AHE.Add(C_2'', C_3'') $\dots, C_k''$), C'),
c_1=AHE.Dec($C'', \{z_i\}_{1\le i\le k}$).
6: let $[a]_i = a_i, [b]_i = b_i, [c]_i = c_i (1 \le i \le k)$.
7: return $([a],[b],[c])$

5.2. Additive Secret Sharing over Polynomial Ring

In additive secret sharing, a secret dealer splits a secret into k secret shares and distributes them to k parties $\mathcal{P}_i (1 \le i \le k)$. To reconstruct the secret, it is only necessary to simply sum the secret shares of each party. It should be noted that the secret cannot be reconstructed with fewer than k secret shares. The classic additive secret sharing is defined over the integer ring (field) [46], and we extend it to the polynomial ring $R_Q = \mathbb{Z}_Q[X]/(X^N+1)$. Since the secret shares are elements over the polynomial ring, we add an automorphism algorithm. Therefore, additive secret sharing over polynomial rings has a total of 6 algorithms (all operations are implicitly performed over the polynomial ring R_Q), as follows.

- ASS.Split(s): Takes a secret $s \in R_Q$ belonging to $\mathcal{P}_i$ as input. For $1 \le i \le k$, party $\mathcal{P}_i$ randomly selects $k-1$ elements $[s]_1, \dots, [s]_{i-1}, [s]_{i+1}, \dots, [s]_k$ in R_Q, computes $[s]_i = s - \sum_{j=1}^{i-1}[s]_j - \sum_{j=i+1}^{k}[s]_j$, and distributes the secret share $[s]_i$ of the secret s to party $\mathcal{P}_i$.
- ASS.Recover($\{[s]_i\}_{1\le i\le k}$): Each party $\mathcal{P}_i$ broadcasts the share $[s]_i$ of the secret s. After collecting $\{[s]_j\}_{1\le j\le k, j\ne i}$ sent by the other $k-1$ parties, computes $s = \sum_{i=1}^{k}[s]_i$.
- ASS.Add($\{[x]_i\}_{1\le i\le k}, \{[y]_i\}_{1\le i\le k}$): Takes the secret shares $[x]_i, [y]_i$ of secrets x and y as input. For $1 \le i \le k$, party $\mathcal{P}_i$ locally computes $[z]_i = [x]_i + [y]_i$, and outputs the secret shares $\{[z]_i\}_{1\le i\le k}$ of the secret $z = x + y$.
- ASS.ScalarMult($\alpha, \{[x]_i\}_{1\le i\le k}$): Takes the secret shares $[x]_i$ of the secret x and a polynomial $\alpha \in R_Q$ as input. For $1 \le i \le k$, party $\mathcal{P}_i$ locally computes $[z]_i = \alpha \cdot [x]_i$, and obtains the secret shares $\{[z]_i\}_{1\le i\le k}$ of the secret $z = \alpha \cdot x$.
- ASS.Mult($\{[x]_i\}_{1\le i\le k}, \{[y]_i\}_{1\le i\le k}, \{[a]_i, [b]_i, [c]_i\}_{1\le i\le k}$): Takes the secret shares $[x]_i, [y]_i$ of secrets x and y, and the secret shares $[a]_i, [b]_i, [c]_i$ of the multiplication triple $(a, b, c = a \cdot b)$ as input. Each party $\mathcal{P}_i$ locally computes $[u]_i = [x]_i - [a]_i$, $[v]_i = [y]_i - [b]_i$ and broadcasts them. After receiving the broadcasts from other parties, party $\mathcal{P}_i$ locally recovers the secrets $u = \sum_{i=1}^{k}[u]_i$ and $v = \sum_{i=1}^{k}[v]_i$. For $\mathcal{P}_1$, locally computes $[z]_1 = u \cdot v + [c]_1 + v \cdot [a]_1 + u \cdot [b]_1$; for other parties $\mathcal{P}_i (2 \le i \le k)$, locally computes $[z]_i = [c]_i + v \cdot [a]_i + u \cdot [b]_i$.
- ASS.Auto($\{[x]_i\}_{1\le i\le k}, \tau_t$): Takes the secret shares $[x]_i$ of the secret x and an automorphism $\tau_t : X \to X^t$ as input. For $1 \le i \le k$, party $\mathcal{P}_i$ locally computes $[z]_i = \tau_t([x]_i)$, and obtains the secret shares $\{[z]_i\}_{1\le i\le k}$ of the secret $z = \tau_t(x)$.

In the same computation task, the multiplication triple input to the ASS.Mult algorithm cannot be reused; otherwise, other parties can infer the corresponding secret. For brevity, the parameter passed to ASS.Mult in the following is a set of triple slices, and the used triple is discarded after each call to

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

secret multiplication. Meanwhile, if the number of parties k is already clear in the following, we will use $[x]$ instead of $\{[x]_i\}_{1\le i\le k}$ to denote the secret shares of the secret x.

5.3. Multi-Group Hybrid Product

The hybrid product algorithm was proposed by Chen et al. [18] in the MK-TFHE scheme in 2019 to support the product between multi-key RLWE ciphertexts and single-key Uni-Enc ciphertexts for constructing CMux gates in the multi-key scenario. Subsequently, Kwak [31] generalized it to support the multiplication of single-key RLEV ciphertexts and multi-key RLWE ciphertexts, thereby realizing the parallelization of blind rotation. In our parallelized bootstrapping scheme, since the intermediate ciphertext of bootstrapping is a single-group NTRU ciphertext rather than a single-key RLEV ciphertext, we propose a multi-group hybrid product algorithm to support the parallelization of bootstrapping, which includes the hybrid product key generation(MGHPKeyGen) , as well as the hybrid product(MGHybridProd) that takes scalar NTRU ciphertexts and multiple sets of ciphertexts as inputs.

- MGHPKeyGen($\{z_j\}_{j\in\mathcal{G}}, [F]$): Takes the RLWE secret key z_j of the j-th party $\mathcal{P}_j$ in group G and the secret share $[F]$ of the joint NTRU key F as input, randomly selects a common reference polynomial vector $\boldsymbol{p} \in R^{d_{br}}$, each party selects a secret polynomial $r_j \leftarrow \chi_{nk}$ from the noise distribution over the ring R and noise polynomial vectors $\boldsymbol{e}_j^{(0)}, \boldsymbol{e}_j^{(1)}, \boldsymbol{e}_j^{(2)} \leftarrow \chi_{ne}^{d_{br}}$. Denote B_{br} and d_{br} as the decomposition base and decomposition degree of the ciphertext modulus Q respectively, and $\mathbf{g}_{B_{br}}$ as the corresponding decomposition vector. For $j \in \mathcal{G}$, compute

$$\boldsymbol{H}_j = -\boldsymbol{p}\cdot z_j + \boldsymbol{e}_j^{(0)}, \widetilde{\boldsymbol{H}} = \sum_{j\in\mathcal{G}} \boldsymbol{H}_j \in R_Q^{d_{br}}, \tag{11}$$

$$\boldsymbol{I}_{j,0} \leftarrow \chi_{ne}^{d_{br}}, \widetilde{\boldsymbol{I}}_0 = \sum_{j\in\mathcal{G}} \boldsymbol{I}_{j,0} \in R_Q^{d_{br}}, \tag{12}$$

$$\boldsymbol{I}_{j,1} = \boldsymbol{p}\cdot r_j + [F]_j \cdot \mathbf{g}_{B_{br}} + \boldsymbol{e}_j^{(1)},\ \widetilde{\boldsymbol{I}}_1 = \sum_{j\in\mathcal{G}} \boldsymbol{I}_{j,1} \in R_Q^{d_{br}}, \tag{13}$$

$$\boldsymbol{I}_{j,2} = -\widetilde{\boldsymbol{I}}_0 \cdot z_j + r_j \cdot \mathbf{g}_{B_{br}} + \boldsymbol{e}_j^{(2)}, \widetilde{\boldsymbol{I}}_2 = \sum_{j\in\mathcal{G}} \boldsymbol{I}_{j,2} \in R_Q^{d_{br}}, \tag{14}$$

  Then output $\widetilde{\boldsymbol{H}}, HPK = \widetilde{\boldsymbol{I}}_0, \widetilde{\boldsymbol{I}}_1, \widetilde{\boldsymbol{I}}_2$.
- MGHybridProd($\{\widetilde{\boldsymbol{H}}_l\}_{1\le l\le k}, HPK_i, ct, \widetilde{\boldsymbol{c}}_i$): Takes the MGRLWE public keys $\{\widetilde{\boldsymbol{H}}_l\}_{1\le l\le k}$, the HPK_i output by the i-th group after executing MGHPKeyGen, the multi-group RLWE ciphertext $ct = (c_0, \ldots, c_k) \in R_Q^{k+1}$ encrypting the plaintext m, and the vector NTRU ciphertext $\widetilde{\boldsymbol{c}}_i = NTRU'_{Q,Fi,B_{br}}(\frac{m}{F_i})$ encrypting the plaintext μ as input. For $0 \le l \le k$, let $\widetilde{\boldsymbol{H}}_0 = -\boldsymbol{p}$, where $\boldsymbol{p}$ is the polynomial vector shared by multiple groups in MGHPKeyGen, computes

$$y_l = c_l \boxdot \widetilde{\boldsymbol{c}}_i,\ u_l = y_l \boxdot \widetilde{\boldsymbol{I}}_{i,1},\ v = \sum_{l=0}^{k} y_l \boxdot \widetilde{\boldsymbol{H}}_l \tag{15}$$

  then update

$$c'_l = \begin{cases} u_0 + v \boxdot \widetilde{\boldsymbol{I}}_{i,2}, & l = 0 \\ u_l + v \boxdot \widetilde{\boldsymbol{I}}_{i,0}, & l = i \\ u_l, & otherwise \end{cases} \tag{16}$$

  Outputs the multi-group RLWE ciphertext $ct' = (c'_0, \ldots, c'_k) \in R_Q^{k+1}$ encrypting the plaintext $\mu \cdot m$.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

5.4. Other Core algorithms for Bootstrapping

This subsection introduces other core algorithms for multi-group ciphertext bootstrapping, including the group registration algorithm Enroll for generating auxiliary material of the bootstrapping key, the JBRKGen algorithm for generating the joint bootstrapping key, the sample extraction algorithm SampleExt for extracting multi-group LWE ciphertexts from multi-group RLWE ciphertexts, the ModSwitch algorithm for switching the modulus of multi-group LWE ciphertexts, the algorithm LKSKeyGen for generating LWE key switching keys, and the LKeySwitch algorithm for switching LWE ciphertext keys.

- Enroll($\{\boldsymbol{s}_j, f_j\}_{j\in\mathcal{G}}$): Takes the LWE secret key $\boldsymbol{s}_j$ and its corresponding NTRU key f_j of each party $\mathcal{P}_j$ in a group $\mathcal{G}$ as input, and outputs $SF, SIF, SSK, SNSSK$, which respectively denote the secret share of the joint NTRU key F, the secret share of F^{-1}(multiplicative inverse of F), the secret share of the secret key, and the secret share of the negative sum secret keys. The algorithm is presented as shown in Algorithm 3.

Algorithm 3 Enroll($\{\boldsymbol{s}_j, f_j\}_{j\in\mathcal{G}}, Triples$).

Require:
LWE secret key $\boldsymbol{s}_j$ of all parties $\mathcal{P}_j$ in the group $\mathcal{G}$
NTRU key f_j of all parties $\mathcal{P}_j$ in the group $\mathcal{G}$
Set of multiplication triples $Triples$

Ensure: Secret shares $[F], [\frac{1}{F}], \{[X^{\sum_{j\in\mathcal{G}} s_{j,l}}]\}_{1\le l\le n}, [X^{-\sum_{j\in\mathcal{G}}\sum_{l=1}^{n} s_{j,l}}]$

1: For $j\in\mathcal{G}, 1\le l\le n$, each party $\mathcal{P}_j$ computes
sf_j=ASS.Split(f_j), sif_j=ASS.Split($\frac{1}{f_j}$), $ssk_{j,l}$= ASS.Split($X^{s_{j,l}}$), $snsk_{j,l}$=ASS.Split($X^{-s_{j,l}}$),
and broadcasts the corresponding secret shares to other parties in the network.
2: For $1\le l\le n$, $\mathcal{P}_1$ computes
SF=ASS.Split(1), SIF=ASS.Split(1), SSK=ASS.Split(1), $SNSSK$=ASS.Split(1), $sask_l$= ASS.Split(1).
3: **for** j in $\mathcal{G}$ **do**
4: SF=ASS.Mult($SF, sf_j, Triples$) ▷ corresponding to $[F]$
5: SIF=ASS.Mult($SIF, sif_j, Triples$) ▷ corresponding to $[\frac{1}{F}]$
6: **for** $l = 1$ to n **do**
7: $sask_l$=ASS.Mult($sask_l, ssk_{j,l}, Triples$) ▷ corresponding to $\{[X^{\sum_{j\in\mathcal{G}} s_{j,l}}]\}_{1\le l\le n}$
8: $SNSSK$=ASS.Mult($SNSSK, snsk_{j,l}, Triples$) ▷ corresponding to $[X^{-\sum_{j\in\mathcal{G}}\sum_{l=1}^{n} s_{j,l}}]$
9: **end for**
10: **end for**
11: $SSK = \{sask_l\}_{1\le l\le n}$
12: return $SF, SIF, SSK, SNSSK$

The secret shares SF, SIF ,SSK, $SNSSK$ are key information for generating the evaluation key EVK in the XZD scheme. After obtaining the share sets of these four components, we can easily generate the joint bootstrapping key(multi-group version of EVK) in the multi-group scenario using additive secret sharing.

- JBRKGen($SF, SIF, SSK, SNSSK, Triples$): Takes SF, SIF, SSK, $SNSSK$ output by the Enroll algorithm and a set of multiplication triples $Triples$ as input, outputs the joint bootstrapping key $JEVK$. The details are as shown in Algorithm 4.

The above algorithm generates multi-group version of the EVK in the XZD scheme via additive secret sharing; specifically, it implements the encryption algorithm in the FINAL scheme using various algorithms under the framework of additive secret sharing, and completes the encryption of the relevant keys in accordance with the flow of BRKGen algorithm in the XZD scheme.

- SampleExt($\boldsymbol{c}$): Takes the MGRLWE ciphertext $\boldsymbol{c} = (c_0, ..., c_k) \in R_Q^{k+1}$ as input, denotes $c_{i,j}$ as the j-th coefficient of the polynomial c_i, sets $c'_0 = c_{0,0}$ and $\boldsymbol{c}'_i = (c_{i,0}, -c_{i,N-1}, ..., -c_{i,1})$ for $1\le i\le k$, and outputs $\widetilde{\boldsymbol{c}} = (c'_0, \boldsymbol{c}'_1, ..., \boldsymbol{c}'_k) \in \mathbb{Z}_Q^{kn+1}$.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026

doi:10.20944/preprints202601.0638.v1

Algorithm 4 JBRKGen($SF, SIF, SSK, SNSSK, Triples$).

Require:
Secret shares $SF, SIF, SNSK, SSSK$
Set of multiplication triples $Triples$
Ensure: joint bootstrapping key $JEVK = (\{jevk_l\}_{1\le l\le n+1}, \{jnksk_r\}_{r\in S})$

1: For $j \in \mathcal{G}, u \in S, 1 \le l \le n+1, 0 \le r \le d_{br} - 1$, each party $\mathcal{P}_j$ randomly selects noise $e_{j,l,r} \leftarrow \chi_{ne}$, $e'_{j,u,r} \leftarrow \chi_{ne}$ and computes $[e_{j,l,r}]$= ASS.Split($e_{j,l,r}$), $[e_{j,u,r}]$= ASS.Split($e_{j,u,r}$), $\mathcal{P}_1$ computes $[E_{l,r}]$=ASS.Split(0), $[E_{u,r}]$=ASS.Split(0), and broadcasts the corresponding secret shares to other parties in the network.
2: **for** j in $\mathcal{G}$ **do**
3: **for** $r = 0$ to $d_{br} - 1$ **do**
4: **for** $l = 1$ to $n+1$ **do**
5: $[E_{l,r}]$ = ASS.Add($[E_{l,r}]$, $[e_{j,l,r}]$)
6: **end for**
7: **for** u in S **do**
8: $[E'_{u,r}]$ = ASS.Add($[E'_{u,r}]$, $[e'_{j,u,r}]$)
9: **end for**
10: **end for**
11: **end for**
12: **for** $l = 1$ to $n+1$ **do**
13: **for** $r = 0$ to $d_{br} - 1$ **do**
14: **if** $l == 1$ **then**
15: $[jevk_{l,r}] = \text{ASS.Mult}(\text{SIF}, \text{ASS.Add}([E_{l,r}], \text{ASS.ScalarMult}(B^r_{br}, sask_1)), Triples)$
16: **else if** $l == n+1$ **then**
17: $[jevk_{l,r}] = \text{ASS.Add}(\text{ASS.ScalarMult}(B^r_{br}, SNSSK), \text{ASS.Mult}(SIF, [E_{l,r}], Triples))$
18: **else**
19: $[jevk_{l,r}] = \text{ASS.Add}(\text{ASS.ScalarMult}(B^r_{br}, sask_l), \text{ASS.Mult}(SIF, [E_{l,r}], Triples))$
20: **end if**
21: **end for**
22: **end for**
23: **for** u in S **do**
24: **for** $r = 0$ to $d_{br} - 1$ **do**
25: $[jnksk_{u,r}] = \text{ASS.Mult}(SIF, \text{ASS.Add}([E'_{u,r}], \text{ASS.ScalarMult}(B^r_{br}, \text{ASS.Auto}(SF, \tau_u))), Triples)$
26: **end for**
27: **end for**
28: **for** $l = 1$ to $n+1$ **do**
29: $jevk_l = (\text{ASS.Recover}([jevk_{l,0}]), ..., \text{ASS.Recover}([jevk_{l,d_{br}-1}]))$
30: **end for**
31: **for** u in S **do**
32: $jnksk_u = (\text{ASS.Recover}([jnksk_{u,0}]), ..., \text{ASS.Recover}([jnksk_{u,d-1}]))$
33: **end for**
34: $JEVK = (\{jevk_l\}_{1\le l\le n+1}, \{jnksk_u\}_{u\in S})$.
35: return $JEVK$

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

doi:10.20944/preprints202601.0638.v1

- ModSwitch($\widetilde{c}, Q, q$): Takes the MGLWE ciphertext $\widetilde{\boldsymbol{c}} = (c_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k)$, the original modulus Q and the new modulus q as input, computes $c_0' = \left\lfloor \frac{q}{Q} \cdot c_0 \right\rceil$ and $\boldsymbol{c}_i' = \left\lfloor \frac{q}{Q} \cdot \boldsymbol{c}_i \right\rceil$ for $1 \leq i \leq k$, and outputs the new ciphertext $\widetilde{c}' = (c_0', \boldsymbol{c}_1', \ldots, \boldsymbol{c}_k')$.
- LKSKeyGen($\boldsymbol{z}_j, \boldsymbol{s}_j$): Takes the LWE secret keys $\boldsymbol{z}_j = (z_{j,1}, \ldots, z_{j,N}) \in \mathbb{Z}^N$ and $\boldsymbol{s}_j = (s_{j,1}, \ldots, s_{j,n}) \in \mathbb{Z}^n$ of party $\mathcal{P}_j$ in the group $\mathcal{G}$ as input, randomly selects a matrix $A_l \in \mathbb{Z}_{q_{ks}}^{d \times n}$ within the group, randomly samples the noise $\boldsymbol{e}_{j,l} \leftarrow \chi_{le}^{d_{ks}}$ for $1 \leq l \leq N$, computes $\boldsymbol{b}_{j,l} = -A_l \cdot \boldsymbol{s}_j + \boldsymbol{e}_{j,l} + z_{j,l} \cdot \mathbf{g}_{B_{ks}}$, denotes $\widetilde{\boldsymbol{b}}_l = \sum_{j \in \mathcal{G}} \boldsymbol{b}_{j,l} \in \mathbb{Z}_{q_{ks}}^{d_{ks}}$, and outputs $LKSK_l = \left\{ \left(\widetilde{\boldsymbol{b}}_l, A_l \right) \right\}_{1 \leq l \leq N}$.
- LKeySwitch($\boldsymbol{c}, \{LKSK_i\}_{1 \leq i \leq k}$): Takes the MGLWE ciphertext $\boldsymbol{c} = (c_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k) \in \mathbb{Z}^{kN+1}$ as well as the key switching key $LKSK_i = \left\{ \left(\widetilde{\boldsymbol{b}}_{i,l}, A_{i,l} \right) \right\}_{1 \leq l \leq N}$ for the i-th group as input, denotes $\boldsymbol{c}_i = (c_{i,1}, \ldots, c_{i,N})$ for $1 \leq i \leq k, 1 \leq l \leq N$, computes $c_0' = c_0 + \sum_{i=1}^{k} \sum_{l=1}^{N} \left(\mathfrak{g}_{B_{ks}}^{-1}(c_{i,l}) \right)^T \cdot \widetilde{\boldsymbol{b}}_{i,l} \in \mathbb{Z}_{q_{ks}}$, $\boldsymbol{c}_i' = \sum_{l=1}^{N} \left(\mathfrak{g}_{B_{ks}}^{-1}(c_{i,l}) \right)^T \cdot A_{i,l} \in \mathbb{Z}_{q_{ks}}^n$, and outputs $\widetilde{c}' = (c_0', \boldsymbol{c}_1', \ldots, \boldsymbol{c}_k') \in \mathbb{Z}_{q_{ks}}^{kn+1}$.

5.5. Parallelizable Multi-Group Ciphertext Bootstrapping Algorithm

In this subsection, we formally describe the multi-group LWE ciphertext bootstrapping algorithm based on LWE and NTRU proposed in this paper. The algorithm takes a multi-group LWE ciphertext to be bootstrapped and some precomputed bootstrapping materials as inputs, and outputs a multi-group LWE ciphertext with a lower noise that encrypts the same plaintext.

In the proposed multi-group homomorphic encryption scheme, bootstrapping involves three types of keys: LWE secret keys, RLWE secert keys, and NTRU keys. For $1 \leq i \leq k$ and $j \in \mathcal{G}_i$, let $\boldsymbol{s}_{i,j} \leftarrow \chi_{lk}^n$, $z_{i,j}, f_{i,j} \leftarrow \chi_{nk}$ denote the LWE secret key, RLWE secret key, and NTRU key of $\mathcal{P}_{i,j}$, respectively. Let Q, q_{ks}, and q denote the ciphertext modulus of NTRU/MGRLWE, the LWE key switching modulus, and the LWE bootstrapping modulus, respectively, with $d_{br} = \lceil \log_{B_{br}} Q \rceil$. We define the following precomputed materials.

1. Set of multiplication triples: $Triples \leftarrow$ TripleGen.
2. Secret shares generated by Enroll algorithm of parties $\mathcal{P}_{i,j}$ in group $\mathcal{G}_i$: $SF_i, SIF_i, SSK_i, SNSSK_i \leftarrow$ Enroll.
3. Joint bootstrapping key of group $\mathcal{G}_i$: $JEVK_i \leftarrow$ JBRKGen.
4. Multi-group hybrid product key of group $\mathcal{G}_i$: $\widetilde{\boldsymbol{H}}_i, HPK_i \leftarrow$ MGHPKeyGen.
5. LWE key switching key of group $\mathcal{G}_i$: $LKSK_i \leftarrow$ LKSKeyGen.

We define the multi-group ciphertext bootstrapping algorithm BootMGCT in Algorithm 5.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

Algorithm 5 BootMGCT$(\boldsymbol{c}, r(X), \{JEVK_i\}_{1\le i\le k}, \{HPK_i\}_{1\le i\le k}, \{LKSK_i\}_{1\le i\le k})$.

Require:
Noisy MGLWE ciphertext $\boldsymbol{c} = (b, \boldsymbol{a}_1, ..., \boldsymbol{a}_k) \in \mathbb{Z}_q^{kn+1}$.
Rotation Polynomial $r(X)$.
Joint Blind Rotation Keys $\{JEVK_i\}_{1\le i\le k}$.
Hybrid Product Key $\{HPK_i\}_{1\le i\le k}$.
LWE Key Switching Key $\{LKSK_i\}_{1\le i\le k}$.
Ensure:
Refreshed MGLWE ciphertext $\boldsymbol{c}' = (b', \boldsymbol{a}'_1, ..., \boldsymbol{a}'_k) \in \mathbb{Z}_q^{kn+1}$
1: $ACC \leftarrow (\lfloor \frac{Q}{8} \rceil \cdot r(X^{\frac{2N}{q}}) \cdot X^{\frac{2N}{q}\cdot b}, 0, ..., 0) \in R_Q^{k+1}$.
2: **for** $i = 1$ to k **do**
3: **for** $r = 0$ to $d_{br} - 1$ **do**
4: $ACC_{i,r} \leftarrow$ BREval$((0, \boldsymbol{a}_i), 1, JEVK_i, B_{br}^r)$
5: **end for**
6: **end for**
7: Parse $ACC'_i = (ACC_{i,0}, ..., ACC_{i,d_{br}-1})$.
8: **for** $i = 1$ to k **do**
9: $ACC \leftarrow$ MGHybridProd$(\{\widetilde{\boldsymbol{H}}_l\}_{0\le l\le k}, HPK_i, ACC, ACC'_i)$.
10: **end for**
11: $ACC \leftarrow$ SampleExt(ACC)
12: $ACC \leftarrow ACC + (\frac{Q}{8}, \boldsymbol{0}, ..., \boldsymbol{0})$
13: $ACC \leftarrow$ ModSwitch(ACC, Q, q_{ks})
14: $ACC \leftarrow$ LKeySwitch$(ACC, \{LKSK_i\}_{1\le i\le k})$
15: $ACC \leftarrow$ ModSwitch(ACC, q_{ks}, q)
16: return ACC

Line 1 of the algorithm stores the rotation polynomial into the accumulator (ACC). Lines 3 to 5 compute the ciphertext $\mathrm{NTRU}'_{Q,F_i,B_{br}}\left(\frac{X^{\sum_{j\in\mathcal{G}_i}\langle \boldsymbol{a}_i, \boldsymbol{s}_{i,j}\rangle}}{F_i}\right)$. These two values are taken as inputs to perform k rounds of iteration(MGHybridProd), outputting the multi-group RLWE (MGRLWE) ciphertext encrypted under the joint RLWE public key of each group, wherein the ciphertext encrypt the value of $L(X) = \lfloor \frac{Q}{8} \rceil \cdot r(X^{\frac{2N}{q}}) \cdot X^{\frac{2N}{q}(b+\sum_{i=1}^{k}\sum_{j\in\mathcal{G}_i}\langle \boldsymbol{a}_i, \boldsymbol{s}_{i,j}\rangle)}$. The exponent part of the monomial $r(X^{\frac{2N}{q}}) \cdot X^{\frac{2N}{q}(b+\sum_{i=1}^{k}\sum_{j\in\mathcal{G}_i}\langle \boldsymbol{a}_i, \boldsymbol{s}_{i,j}\rangle)}$ corresponds to the decryption formula of MGLWE ciphertexts. The SampleExt algorithm in Line 11 outputs the LWE ciphertext that encrypts the first coefficient of $L(X)$ under the encryption of the RLWE secret key coefficient vector. Since its first coefficient is either -1 or 1, Step 12 accumulates $(\frac{Q}{8}, \boldsymbol{0}, \ldots, \boldsymbol{0})$ into ACC, mapping its value to either 0 or 1. After the subsequent steps of the algorithm, a low-noise MGLWE ciphertext with $\frac{q}{4}$ as the scaling factor is output.

6. Noise Analysis

This section analyzes the noise generated by the proposed bootstrapping algorithm. For an input multi-group LWE ciphertext $\boldsymbol{c}$, the noise of the output ciphertext $\boldsymbol{c}'$ originates from BREval, MGHybridProd, ModSwitch, and LKeySwitch. We analyze the noise output by each of these algorithms separately, and finally derive the total noise output by the bootstrapping algorithm.

The LWE scheme has parameters q', q, q_{ks}, B_{ks}, d_{ks}, n, which correspond to the LWE ciphertext modulus of MGHE.Enc, the LWE ciphertext modulus for bootstrapping, the ciphertext modulus for key switching, the gadget decomposition base, the gadget decomposition degree, and the LWE dimension, respectively. The RLWE/NTRU scheme has parameters Q, B_{br}, d_{br}, N, which correspond to the RLWE/NTRU ciphertext modulus, the corresponding gadget decomposition base, the gadget decomposition degree, and the degree of the polynomial in the RLWE/NTRU ciphertext, respectively. Two bootstrapping schemes are involved in the bootstrapping process: LWE and RLWE/NTRU, with noise distributions χ_{le} and χ_{ne} (corresponding variances $\sigma_{le}^2 2$ and σ_{ne}^2) and key distributions χ_{lk} and χ_{nk} (corresponding variances σ_{lk}^2 and σ_{nk}^2), respectively. For the convenience of analysis, we assume that there are k groups participating in the computation, and each group has K parties.

Lemma 2. *Noise Variance of BREval* [20]

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Let σ_{brk}^2 denote the noise variance of the input *jevk*, *jnksk*. For the ciphertext c output by the BREval algorithm, the noise variance of c satisfies

$$Var(err(c)) \leq (2n+1) \cdot Nd_{br} \frac{B_{br}^2}{12} \cdot \sigma_{ne}^2. \tag{17}$$

Proof. The BREval algorithm performs at most $2n+1$ external products between scalar NTRU ciphertexts and vector NTRU ciphertexts. The rotation polynomial can be regarded as a vector NTRU ciphertext with zero noise. Hence, we complete the proof via Lemma 1.

Lemma 3. *Noise Variance of MGHybridProd*

Given a multi-group RLWE ciphertext c with noise variance $Var(err(c))$, the noise $err(c')$ of the ciphertext c' output by the MGHybridProd algorithm satisfies

$$\begin{aligned} Var(err(c')) = Var(err(c)) &+ \frac{Kkd_{br}N^2B_{br}^2\sigma_{ne}^2 Var(err(\widetilde{\boldsymbol{c}}_i))}{12} \\ &+ \frac{K^2kd_{br}B_{br}^2\sigma_{nk}^2\sigma_{ne}^2}{6} \\ &+ \frac{(2k\sigma_{ne}^2 + Var(err(\widetilde{\boldsymbol{c}}_i))d_{br}B_{br}^2}{12}. \end{aligned} \tag{18}$$

Proof. Assume that the noise variance of the multi-group RLWE ciphertext c encrypting the plaintext m before MGHybridProd is $Var(err(c))$, and the noise variance of the ciphertext c' encrypting the plaintext μm after MGHybridProd is $Var(err(c'))$. Let Z denote the group joint RLWE secret key, and R, E denote the sums of r and e defined in the algorithm, respectively. According to the MGHybridProd algorithm, the noise $err(c')$ of the new ciphertext satisfies

$$\begin{aligned} err(c') &= \left(\sum_{l=0}^{k} c'_l \cdot Z_l\right) - \frac{Q}{4}\mu m \\ &= \mu \cdot err(c) + \left(\sum_{l=0}^{k} Z_l \cdot err(y_l)\right) + R_i \left(\sum_{l=1}^{k} y_l \boxdot \boldsymbol{E}_l^{(0)}\right) \\ &+ \left(\sum_{l=0}^{k} y_l \boxdot \boldsymbol{E}_i^{(0)} \cdot Z_l\right) + \left(v \boxdot \boldsymbol{E}_i^{(2)}\right), \end{aligned} \tag{19}$$

Given that μ is a monomial with an absolute coefficient value of 1, the variance of the first term is $Var(err(c))$. Since $err(y_l) = err(c_l \boxdot \widetilde{\boldsymbol{c}}_i)$, it follows from Lemma 1 that $err(y_l) = d_{br}N\frac{B_{br}^2}{12} \cdot err(\widetilde{\boldsymbol{c}}_i)$. As $Z_0 = 1$ and Z_l is the sum of the RLWE secret keys held by K parties, we can derive that $Var(Z_l) = K\sigma_{nk}^2$. Accordingly, the variance corresponding to the second term in the noise expression is $(1 + Kk\sigma_{nk}^2 N)d_{br}N\frac{B_{br}^2}{12}\text{Var}(err(\widetilde{\boldsymbol{c}}_i))$. On the other hand, R_i and $\boldsymbol{E}_l^{(0)}$ are both accumulations of random variables independently selected by K parties, which means their variances are $K\sigma_{nk}^2$ and $K\sigma_{ne}^2$, respectively. Based on this, the variance of the third term in the expression can be expressed as $Kk\sigma_{nk}^2 d_{br}N\frac{B_{br}^2}{12} \cdot (K\sigma_{ne}^2)$. Similarly, the variance of the fourth term is given by $\frac{d_{br}N(K\sigma_{ne}^2)B_{br}^2}{12} + \frac{Kk\sigma_{nk}^2 d_{br}N(K\sigma_{ne}^2)B_{br}^2}{12}$, and the variance of the last term is $\frac{d_{br}N\cdot(K\sigma_{ne}^2)B_{br}^2}{12}$. By integrating and rearranging all the aforementioned variance terms, the lemma is thus proved.

Lemma 4. *Noise Variance of ModSwitch*

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

Given positive integers Q, q, a multi-group LWE ciphertext $\boldsymbol{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k)$ with noise variance $Var(err(\boldsymbol{c}))$, and the corresponding joint LWE secret key $\boldsymbol{s}$, a new ciphertext $\boldsymbol{c}'$ is obtained after modulus switching. The noise $err(\boldsymbol{c}')$ of the new ciphertext satisfies

$$Var(err(\boldsymbol{c}')) = \frac{q^2}{Q^2} \cdot Var(err(\boldsymbol{c})) + \frac{1 + Kkn \cdot \sigma_{lk}^2}{12} \tag{20}$$

Proof. Consider an original ciphertext $\boldsymbol{c} = (b, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k)$ with modulus q, LWE dimension n, and secret keys $\boldsymbol{s}_{i,j}$ of each party in each group, which needs to be switched to modulus Q. After modulus switching, $c' = (b', \boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_k)$, where $b' = \left\lfloor \frac{q}{Q} \cdot b \right\rceil = \frac{q}{Q} \cdot b + \epsilon, \boldsymbol{a}'_i = \left(\left\lfloor \frac{q}{Q} \cdot a_{i,1} \right\rceil, \ldots, \left\lfloor \frac{q}{Q} \cdot a_{i,n} \right\rceil \right) = \left(\frac{q}{Q} \cdot a_{i,1} + \epsilon, \ldots, \frac{q}{Q} \cdot a_{i,n} + \epsilon \right)$. Here, ϵ denotes the rounding error, which follows a uniform distribution over $\left(-\frac{1}{2}, \frac{1}{2}\right)$. The noise satisfies $e' = b' - \sum_{i=1}^{k} \sum_{j=1}^{K} \langle \boldsymbol{a}'_i, \boldsymbol{s}_{i,j} \rangle - \frac{q}{4} m = \frac{q}{Q} e + \epsilon - \sum_{i=1}^{k} \sum_{j=1}^{K} \langle \boldsymbol{\epsilon}, \boldsymbol{s}_{i,j} \rangle$, thus $Var(err(\boldsymbol{c}')) = \frac{q^2}{Q^2} err(\boldsymbol{c}) + \frac{1+Kkn \cdot \sigma_{lk}^2}{12}$.

Lemma 5. *Noise Variance of LKeySwitch*

Given a multi-group LWE ciphertext $\boldsymbol{c} = (b, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k)$ with LWE dimension N encrypted under k joint RLWE secret key coefficient vectors $\boldsymbol{z}_i, (1 \le i \le k)$, the LWE key switching algorithm outputs a multi-group LWE ciphertext $\boldsymbol{c}' = (b', \boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_k)$ with LWE dimension n encrypted under k LWE joint secret keys $\boldsymbol{s}_i$. The noise variance of the new ciphertext $\boldsymbol{c}'$ satisfies

$$Var(err(c')) = Var(err(c) + \frac{KkNd_{ks}B_{ks}^2\sigma_{le}^2}{12}. \tag{21}$$

Proof. The components of the joint LWE secret key corresponding to the ciphertext $\boldsymbol{c}' = (b', \boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_k)$ output by the LKeySwitch algorithm are denoted by $\boldsymbol{s}_i$. The corresponding noise $\boldsymbol{e}'$ satisfies

$$\begin{aligned}
e' &= b' + \sum_{i=1}^{K} \langle \boldsymbol{a}'_i, \boldsymbol{s}_i \rangle - \frac{q}{4} m \\
&= b + \sum_{i=1}^{K} \sum_{l=0}^{N-1} \mathfrak{g}_{B_{ks}}^{-1}(a_{i,l})^T \cdot (\mathbf{g}_{B_{ks}} \cdot z_{i,l} + e_{i,l}) - \frac{q}{4} m \\
&= b + \sum_{i=1}^{K} \sum_{l=0}^{N-1} a_{i,l} \cdot z_{i,l} + \sum_{i=1}^{K} \sum_{l=0}^{N-1} \mathfrak{g}_{B_{ks}}^{-1}(a_{i,l})^T \cdot e_{i,l} - \frac{q}{4} m \\
&= e + \sum_{i=1}^{K} \sum_{l=0}^{N-1} \mathfrak{g}_{B_{ks}}^{-1}(a_{i,l})^T \cdot e_{i,l}.
\end{aligned} \tag{22}$$

Since the variance of $e_{i,l}$ is $K\sigma_{le}^2$, it follows that $Var(err(\boldsymbol{c}')) = Var(err(\boldsymbol{c}) + \frac{KkNd_{ks}B_{ks}^2\sigma_{le}^2}{12}$.

Theorem 1. *Noise Variance of the Multi-Key Bootstrapping Algorithm*

Given a ciphertext $\boldsymbol{c}$ to be bootstrapped with noise variance $Var(err(\boldsymbol{c}))$, a rotation polynomial r, joint bootstrapping keys $\{JEVK_i\}_{1 \le i \le k}$, hybrid product keys $\{HPK_i\}_{1 \le i \le k}$, $\{\widetilde{\boldsymbol{H}}_i\}_{0 \le i \le k}$, and LWE

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

key switching keys $\{LKSK_i\}_{1\le i\le k}$, the noise variance of the new ciphertext output by the BootMGCT algorithm, denoted as $Var(err(\boldsymbol{c}'))$, satisfies

$$\begin{aligned} Var(err(\boldsymbol{c}')) = & \frac{q^2Kk(k+1)d_{br}^2N^3(2n+1)B_{br}^4\sigma_{ne}^4}{288Q^2} \\ & + \frac{q^2K^2k(k+1)d_{br}B_{br}^2\sigma_{nk}^2\sigma_{ne}^2}{12Q^2} \\ & + \frac{q^2kd_{br}B_{br}^2}{144Q^2}\left(24K + d_{br}N(2n+1)B_{br}^2\right)\sigma_{ne}^2 \\ & + \frac{q^2(1+KkN\sigma_{lk}^2) + q_{ks}^2(1+Kkn\sigma_{lk}^2)}{12q_{ks}^2} \\ & + \frac{q^2KkNd_{ks}B_{ks}^2\sigma_{le}^2}{12q_{ks}^2}. \end{aligned} \tag{23}$$

Proof. According to Lemma 2, the noise variance of $\widetilde{c}_i$ input to MGHybridProd is $(2n+1)\cdot Nd_{br}\frac{B_{br}^2}{12}\cdot\sigma_{ne}^2$, denoted as σ_{br}^2. Before entering the i-th loop in Line 9 of the algorithm, the last $k-i$ terms of the multi-group RLWE ciphertext stored in ACC are all zero, so no noise accumulates. According to Lemma 3, after executing the MGHybridProd loop in Line 9, the noise variance of the ciphertext is $\sigma_{hp}^2 = \frac{Kk(k+1)d_{br}N^2B_{br}^2\sigma_{ne}^2\sigma_{br}^2}{24} + \frac{K^2k(k+1)d_{br}B_{br}^2\sigma_{nk}^2\sigma_{ne}^2}{12} + \frac{k(2K\sigma_{ne}^2+\sigma_{br}^2)d_{br}B_{br}^2}{12}$. According to Lemma 4, after the modulus switching in Line 12, the noise variance of the ciphertext is $\sigma_{ms_1}^2 = \frac{q_{ks}^2}{Q^2}\sigma_{hp}^2 + \frac{1+KkN\sigma_{lk}^2}{12}$. According to Lemma 5, after the LWE key switching in Line 13, the noise variance of the ciphertext is $\sigma_{ks}^2 = \sigma_{ms_1}^2 + \frac{KkNd_{ks}B_{ks}^2\sigma_{le}^2}{12}$. Similar to the first modulus switching, the noise after the second modulus switching in Line 14 of the algorithm is $\sigma_{ms_2}^2 = \frac{q^2}{q_{ks}^2}\sigma_{ks}^2 + \frac{1+Kkn\sigma_{lk}^2}{12}$. The theorem is proved by combining the above results.

7. Parameters and Implementation

7.1. Security Model and Security Analysis

The proposed scheme is divided into an offline phase and an online phase. The offline phase refers to the key generation phase, which is distinguished from the online phase of computing the target function. Since the security of the online phase has been proven in [20,31], we focus on the security proof of the offline phase.

Under the semi-honest adversary model, the additive secret sharing (ASS) over the polynomial ring $R_Q = \mathbb{Z}_Q[X]/(X^N+1)$ ensures secret confidentiality through strict share randomness and operational security. In the secret splitting phase (ASS.Split), a secret $s \in R_Q$ is split into k shares, where $k-1$ shares are uniformly randomly selected from R_Q, and only the remaining 1 share is determined by the difference between the secret and other shares. This ensures that any subset of shares from fewer than k parties contains no valid information about the secret, and the adversary cannot infer the secret from the subset of shares. Secret addition (ASS.Add), scalar multiplication (ASS.ScalarMult), and automorphism (ASS.Auto) are all completed through local computations by parties, and new secret shares can be generated without interaction, avoiding the leakage of intermediate information. Secret multiplication (ASS.Mult) relies on Beaver multiplication triple $([a],[b],[c])$, as long as the same triple is not reused, the adversary cannot obtain the original secret through intermediate differences [42]. The security of the triple generation protocol directly depends on the RLWE-based additive homomorphic encryption (AHE) scheme, and the security of AHE is based on the computational hardness of the decisional RLWE problem. Thus, semi-honest adversaries cannot obtain secrets from the triple generation process.

Additive secret sharing is used in the Enroll algorithm, JBRKGen algorithm, and MGHPKeyGen algorithm. The Enroll algorithm does not involve secret reconstruction, so the security of the Enroll

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

algorithm is equivalent to that of additive secret sharing. The JBRKGen and MGHPKeyGen algorithms need to obtain vector NTRU ciphertexts and RLWE ciphertexts through secret reconstruction, without directly leaking key information.

In summary, the offline phase of the proposed scheme can guarantee privacy under the semi-honest adversary model. The semantic security (IND-CPA) of the online phase can be directly derived based on the decisional LWE/RLWE problem, decisional NTRU problem, and decisional vectorized NTRU problem through conventional security reduction methods.

7.2. Parameter Selection

We consider the parameter selection of the multi-group homomorphic encryption scheme from two dimensions: usability and security. Usability means that the set parameters must ensure that the noise size of the ciphertext is always within a specified range to maintain the correctness of the decryption process. Security means that the parameter configuration should be able to resist common attack algorithms, making it impossible for attackers to successfully crack the scheme within the scope of computational feasibility.

According to Lemma 7 of the FHEW scheme [27], for two LWE ciphertexts c_1, c_2 encrypting plaintexts $m_1,\ m_2 \in \mathbb{Z}_2$ with noises $e_1,\ e_2$ respectively, after performing the homomorphic NAND operation, the noise e' of the new ciphertext c' satisfies $e' = e_1 + e_2 \pm \frac{q}{8}$. At this time, the scaling factor corresponding to the ciphertext is $\frac{q}{2}$, so the absolute value of the noise e' needs to satisfy $|\frac{2e'}{q}| < \frac{1}{2}$, i.e., $|e'| < \frac{q}{4}$. Therefore, the sum of the noises of the two ciphertexts undergoing the homomorphic NAND gate needs to satisfy $|e_1 + e_2| < \frac{q}{8}$. In gate bootstrapping, we need to control the upper noise bound of the two input ciphertexts, i.e., the upper noise bound of each ciphertext after bootstrapping needs to be lower than $\frac{q}{16}$. To ensure the correctness of decryption, the upper noise bound of each step in the bootstrapping algorithm is also required not to exceed the threshold $\frac{Q_i}{16}$, where Q_i is the ciphertext modulus of the current ciphertext. We use 6 times the standard deviation as the high-probability upper bound of the noise. Based on the noise variances of ciphertexts at different stages listed in the noise analysis, we obtain the constraints

$$6\sigma_{br} < \frac{Q}{16}, 6\sigma_{hp} < \frac{Q}{16}, 6\sigma_{ms_1} < \frac{q_{ks}}{16}, 6\sigma_{ks} < \frac{q_{ks}}{16}, 6\sigma_{ms_2} < \frac{q}{16}, \tag{24}$$

which serve as one of the bases for parameter selection. The LWE secret key distribution is a symmetric three-point distribution over $\{-1, 0, 1\}$, where the probability of taking values -1 and 1 is $\frac{1}{4}$, and the probability of taking value 0 is $\frac{1}{2}$. The noise of the LWE scheme is sampled from a discrete Gaussian distribution with the mean of 0 and the standard deviation of 3. For the keys and noises of the NTRU and RLWE schemes, we also use the above two distributions respectively. Sampling a polynomial means sequentially sampling coefficients from the distribution, so $\sigma_{lk}^2 = \sigma_{nk}^2 = \frac{1}{2}, \sigma_{le}^2 = \sigma_{ne}^2 = 9$.

From the noise analysis, the noise level of NTRU ciphertexts reaches its peak after executing the MGHybridProd operation. At this time, the NTRU modulus Q satisfies $Q = O(N^{1.5}) < O(N^{2.484})$. From the perspective of the asymptotic complexity of noise, the parameter selection of this scheme can theoretically ensure that it is within a safe range. Therefore, we can use the NTRU estimator [23] to evaluate the security strength of the NTRU parameters. Similarly, we use the LWE estimator [22] for parameter selection of the (R)LWE scheme. Tables 1 and 2 show the parameter sets for 100-bit security and 128-bit security, respectively.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

Table 1. Parameters set of 100 bits security

Set	(k_m, K_m)	q'	q	q_{ks}	B_{ks}	d_{ks}	n	Q	B_{br}	d_{br}	N	Estimate Security
I	(2, 15)	2^{14}	4096	2^{30}	1024	3	600	2^{50}	32	10	4096	100
II	(4, 7)	2^{14}	4096	2^{30}	1024	3	600	2^{50}	32	10	4096	100
III	(8, 7)	2^{14}	4096	2^{30}	1024	3	600	2^{50}	16	13	4096	100

Note: k_m denotes the maximum number of groups; K_m denotes the maximum number of parties per group; q', q, and q_{ks} represent the LWE encryption modulus, LWE bootstrapping modulus, and LWE key switching modulus, respectively; B_{ks}, d_{ks}, B_{br}, and d_{br} denote the gadget decomposition base and degree for key switching and bootstrapping, respectively; n is the LWE dimension; Q is the NTRU/RLWE ciphertext modulus; N is the degree of the polynomial in NTRU/RLWE ciphertexts.

Table 2. Parameters set of 128 bits security

Set	(k_m, K_m)	q'	q	q_{ks}	B_{ks}	d_{ks}	n	Q	B_{br}	d_{br}	N	Estimate Security
I′	(2, 25)	2^{14}	4096	2^{30}	1024	3	600	2^{60}	512	7	8192	128
II′	(4, 10)	2^{14}	4096	2^{30}	1024	3	600	2^{60}	512	7	8192	128
III′	(8, 8)	2^{14}	4096	2^{30}	1024	3	600	2^{60}	256	8	8192	128

Note: Same as Table 1

7.3. Experimental Results

All experiments are conducted on a server equipped with an Intel(R) Xeon(R) Platinum 8481C processor (3.8GHz) and 64GB of memory, with the operating system Ubuntu 22.04.5 LTS. The algorithms proposed in this paper are implemented based on the OpenFHE open-source library. Table 3 and Table 4 show the execution time required for the proposed algorithm to perform a homomorphic NAND gate computation under different parameter sets (I, II, III and I′, II′, III′), respectively.

Table 3. Bootstrapping time of 100 bits security

Set	k	Serialized (s)	Parallelized (s)
I	2	42.58	1.87
II	3	50.37	1.91
	4	82.06	2.31
III	5	146.79	4.18
	6	174.12	4.84
	7	205.59	5.70
	8	223.56	6.13

Note: k denotes the number of groups; Serialized refers to the execution time of serial bootstrapping, and Parallelized refers to the execution time of parallel bootstrapping.

Table 4. Bootstrapping time of 128 bits security

Set	k	Serialized (s)	Parallelized (s)
I′	2	47.36	2.58
II′	3	84.34	2.76
	4	113.50	2.87
III′	5	158.14	4.35
	6	193.99	5.16
	7	222.93	5.96
	8	253.63	6.60

Note: Same as Table 4.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

As illustrated in the BootMGCT algorithm, the number of intra-group parties does not affect the number of homomorphic computations required for ciphertext bootstrapping. The algorithm needs to execute the BREval algorithm $k \cdot d_{br}$ times. Under the same parameter set, the time for serialized bootstrapping increases with the number of groups, which can be verified by the data in Tables 3 and Table 4. When $k = 7$, the time for a single bootstrapping operation exceeds 200 seconds. This indicates that under constrained parameter selection (the ratio of Q to N cannot be excessively large, so d_{br} needs to be increased to reduce noise), serialized bootstrapping of multi-group ciphertexts based on NTRU is infeasible. In contrast, if the bootstrapping algorithm is executed in parallel, the bootstrapping time will be significantly reduced. Figure 2 intuitively illustrates the variation trend of bootstrapping time. When bootstrapping is performed in parallel, for both parameters for 100-bit security and 128-bit security, the bootstrapping time can be controlled within 10 seconds when the number of groups is less than 8. This result shows that by fully leveraging the parallel computing resources of the server, the proposed scheme can achieve bootstrapping efficiency close to that of single-key ciphertexts, greatly enhancing the practicality in multi-group scenarios.

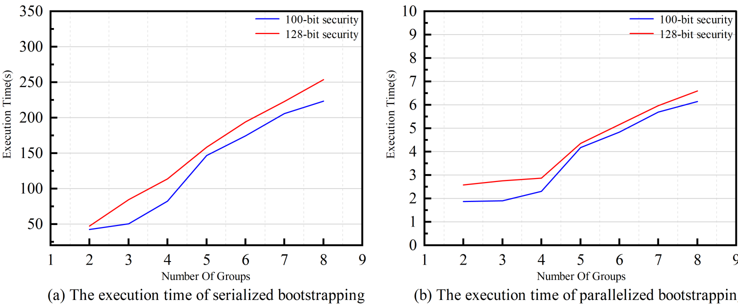

Figure 2. Serial/Parallel Execution Time of BootMGCT algorithm under 100/128 bits Security.

8. Conclusions

This paper proposes a novel multi-group homomorphic encryption scheme based on LWE and NTRU. By linearizing the NTRU joint key through additive secret sharing technology, the scheme designs a multi-group hybrid product algorithm, enabling parallel bootstrapping of multi-group LWE ciphertexts for encrypted bits. In a multi-core server environment, the time cost of bootstrapping a multi-group ciphertext is comparable to that of bootstrapping a single-key ciphertext, significantly improving the practicality of the homomorphic encryption scheme. Additionally, specific parameter selections are provided to ensure the scheme can effectively resist sublattice attacks. Finally, the scheme is implemented on OpenFHE, and its effectiveness is verified. Experimental results show that under the 100-bit security, refreshing a collaborative computing ciphertext involving 50 parties (divided into 2 groups with 25 parties each) only takes less than 2 seconds. Therefore, this scheme provides a new solution for application scenarios requiring group-based computing.

Author Contributions: Conceptualization, Y.L. and F.K.; software, Y.L.; validation, F.K.; formal analysis, B.H.; data curation, J.W. and S.L.; writing—original draft preparation, Y.L.; writing—review and editing, B.H.; visualization, J.W. and S.L.; supervision, B.H.; funding acquisition, J.W., S.L. and B.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by Open Project Program of Guangxi Key Laboratory of Digital Infrastructure (GXDINBC202406) and National Natural Science Foundation of China (61962005).

Data Availability Statement: The experimental data have been presented in this paper, and further data can be obtained from the authors upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

References

1. Author 1, T. The title of the cited article. *Journal Abbreviation* **2008**, *10*, 142–149.
2. Rivest, R. L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **1978**, *21*, 120–126.
3. Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Prague, Czech Republic, May 2–6, 1999; pp. 223-–238.
4. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the forty-first annual ACM symposium on Theory of computing, Bethesda, MD, USA , May 31 – June 2, 2009; pp. 169–178.
5. Gentry, C.; Halevi, S.; Smart, N. P. Better bootstrapping in fully homomorphic encryption. In Proceedings of the International Workshop on Public Key Cryptography, Darmstadt, Germany, May 21–23, 2012; pp. 1–16.
6. Halevi, S.; Shoup, V. Bootstrapping for HElib. *Journal of Cryptology* **2021**, *34*.
7. Chen, H.; Han, K. Homomorphiclowerdigits removal andimprovedFHEbootstrapping. In Proceedings of the 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 – May 3, 2018; pp. 315–337.
8. Kim, J.; Seo, J.; Song, Y. Simpler and faster BFV bootstrapping for arbitrary plaintext modulus from CKKS. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Salt Lake City, UT, USA, October 14–18, 2024; pp. 2535–2546.
9. Bae, Y.; Cheon, J. H.; Kim, J.; Stehlé, D. Bootstrapping bits with CKKS. In Proceedings of the 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26—30, 2024; pp. 94—123.
10. Chillotti, I.; Gama, N.; Georgieva, M. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Proceedings of the 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4–8, 2016; pp.3-–33.
11. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology* **2020**, *33*, 34–91.
12. Bonte, C.; Iliashenko, I.; Park, J.; Pereira, H. V. L. FINAL: faster FHE instantiated with NTRU and LWE. In Proceedings of the 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-–9, 2022; pp. 188-–215.
13. Boneh, D.; Gennaro, R.; Goldfeder, S.; Jain, A. Threshold cryptosystems from threshold fully homomorphic encryption. In Proceedings of the 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19—23, 2018; pp. 565-–596.
14. Mouchet, C.;Bertrand, E.; Hubaux, J. P. An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. *Journal of Cryptology* **2023**, *36*.
15. Park, J. Homomorphic encryption for multiple users with less communications. *IEEE Access* **2021**, *9*, 135915–135926.
16. López-Alt, A.; Tromer, E.; Vaikuntanathan, V. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Proceedings of the forty-fourth annual ACM symposium on Theory of computing, New York, USA, May 19–22, 2012; pp. 1219–1234.
17. Chen, H.; Dai, W.; Kim, M.; Song, Y. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, November 11–15, 2019; pp. 395–412.
18. Chen, H.; Chillotti, I.; Song, Y. Multi-key homomorphic encryption from TFHE. In Proceedings of the 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8—12, 2019; pp. 446—472.
19. Kwak, H.; Lee, D.; Song, Y.; Wagh, S. A general framework of homomorphic encryption for multiple parties with non-interactive key-aggregation. In Proceedings of the 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5—8, 2024; pp. 403—430.
20. Xiang, B.; Zhang, J.; Deng, Y.; Dai, Y.; Feng, D. Fast blind rotation for bootstrapping FHEs. In Proceedings of the 43rd Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-–24, 2023; pp. 3-–36.
21. Li, Z.; Lu, X.; Wang, Z.; Wang, R.; Liu, Y.; Zheng, Y.; Zhao, L.; Wang, K.; Hou, R. Faster NTRU-Based Bootstrapping in Less Than 4 Ms. *TCHES* **2024**, *2024*, 418–451.
22. Albrecht, M. R.; Player, R.; Scott, S. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology* **2015**, *9*, 169–203.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

23. Ducas, L.; van Woerden, W. NTRU fatigue: how stretched is overstretched?. In Proceedings of the 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6—10, 2021; pp. 3-–32.
24. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory* **2014**, *6*, 1–36.
25. Fan, J.; Vercauteren, F. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. Available online: https://eprint.iacr.org/2012/144 (accessed on January 8, 2026).
26. Cheon, J. H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3–7, 2017; pp. 409—437.
27. Ducas, L.; Micciancio, D. FHEW: bootstrapping homomorphic encryption in less than a second. In Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26–30, 2015; pp. 617-–640.
28. Asharov, G.; Jain, A.; López-Alt, A.; Tromer, E. Multiparty computation with low communication, computation and interaction via threshold FHE. In Proceedings of the 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15–19, 2012; pp. 483-–501.
29. Lee,Y.; Micciancio, D.; Kim, A.; Choi, R.; Deryabin, M. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In Proceedings of the 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023; pp. 227-–256.
30. Park,J.; Rovira, S. Efficient TFHE bootstrapping in the multiparty setting. *IEEE Access* **2023**, *11*, 118625–118638.
31. Kwak,H.;Min, S.; Song, Y. Towards practical multi-key TFHE: parallelizable, key-compatible, quasi-linear complexity. In Proceedings of the 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15—17, 2024; pp. 354—385.
32. Xu,K.; Tan, B. H. M.; Wang, L. P.; Aung, K. M. M. Multi-key fully homomorphic encryption from NTRU and (R) LWE with faster bootstrapping. *Theoretical Computer Science* **2023**, *968*, 114026.
33. Kim, J.; Lee, C. A polynomial time algorithm for breaking NTRU encryption with multiple keys. *Designs, Codes and Cryptography* **2023**, *91*, 2779–2789.
34. Park, J.; Van Leeuwen, B.; Zajonc, O. FINALLY: A multi-key FHE scheme based on NTRU and LWE. Cryptology ePrint Archive, Paper 2024/1505, 2024. Available online: https://eprint.iacr.org/2024/1505 (accessed on January 8, 2026).
35. Regev, O. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM* **2009**, *56*, 1–40.
36. Lyubashevsky, V.; Peikert, C.; Regev, O. On ideal lattices and learning with errors over rings. *Journal of the ACM* **2013**, *60*, 1–35.
37. Genise, N.; Gentry, C.; Halevi, S.; Li, B. Homomorphic encryption for finite automata. In Proceedings of the 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-–12, 2019; pp. 473-–502.
38. Albrecht, M.; Bai, S.; Ducas, L. A subfield lattice attack on overstretched NTRU assumptions: Cryptanalysis of some FHE and graded encoding schemes. In Proceedings of the 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14–18, 2016; pp. 153—178.
39. Alperin-Sheriff, J.; Peikert, C. Faster bootstrapping with polynomial error. In Proceedings of the 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2014; pp. 297—314.
40. Mouchet, C.; Troncoso-Pastoriza, J. Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies* **2021**, *2021*, 291–311.
41. Kraitsberg, M.; Lindell, Y.; Osheter, V.; Smart, N. P. Adding distributed decryption and key generation to a ring-LWE based CCA encryption scheme. In Proceedings of the 24th Australasian Conference, ACISP 2019, Christchurch, New Zealand, July 3—5, 2019; pp. 192-–210.
42. Beaver, D. Efficient multiparty protocols using circuit randomization. In Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology, Berlin, Heidelberg, 1991; pp. 420–432.
43. Keller, M.; Orsini, E.; Scholl, P. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24–28, 2016; pp. 830–842.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2026 doi:10.20944/preprints202601.0638.v1

44. Keller, M.; Pastro, V.; Rotaru, D. Overdrive: Making SPDZ great again. In Proceedings of the 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 – May 3, 2018; pp. 158—189.
45. Keller, M.; Sun, K. Secure quantized training for deep learning. In Proceedings of the 39th International Conference on Machine Learning, 2022; pp. 10912–10938.
46. Escudero, D. An introduction to secret-sharing-based secure multiparty computation. Cryptology ePrint Archive, Paper 2022/062, 2022. Available online: https://eprint.iacr.org/2022/062 (accessed on January 8, 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.