mathematics

MDPI

Article

Multi-Key Homomorphic Encryption Scheme with Multi-Output Programmable Bootstrapping

Lingwu Li and Ruwei Huang *

School of Computer and Electronic Information, Guangxi University, Nanning 530004, China
* Correspondence: ruweih@126.com

Abstract: Multi-key Homomorphic Encryption (MKHE) scheme can homomorphically evaluate ciphertexts encrypted by different keys, which can effectively protect the privacy information of data holders in the joint computing of cloud services. Since the first full Homomorphic encryption scheme was proposed, bootstrapping is the only way to realize the arbitrary depth homomorphic computation of MKHE schemes. But bootstrap operation is quite expensive. In order to implement fast bootstrapping in MKHE schemes, previous works proposed multi-key TFHE schemes to implement low-latency bootstrapping and output a univariate function of messages after bootstrapping, called Programmable Bootstrapping (PBS). However, these schemes can only encrypt single-bit messages. PBS only outputs a function. And after a homomorphic operation, a bootstrap is required, which undoubtedly results in an increase in the cost of the whole multi-key homomorphic encryption operation. In this paper, we propose a MKHE scheme for multi-output PBS. For this purpose, we study the encryption method and homomorphic operation steps of MKHE, and add BFV homomorphic encryption multiplication and multi-key ciphertext relinearization. We separate the homomorphic operation from bootstrapping. We homomorphically evaluate test polynomials for multiple functions. In contrast to previous MKHE schemes, we support the output of multiple message-related functions with a single bootstrapping operation on the ciphertext. It is no longer limited to encrypting single-bit plaintext, and an effective ciphertext packaging technology is added. According to the analysis given in this paper, it is known that in the scenario of multi-party joint computation, the proposed scheme can be implemented with less bootstrapping when the same number of functions are homomorphically operated. This will effectively reduce the computational overhead.

Keywords: multi-key homomorphic encryption; packaged ciphertext; TFHE; programmable bootstrapping

MSC: 68W99

Citation: Li, L.; Huang, R. Multi-Key Homomorphic Encryption Scheme with Multi-Output Programmable Bootstrapping. *Mathematics* **2023**, *11*, 3239. https://doi.org/10.3390/math11143239

Academic Editor: Cheng-Chi Lee

Received: 19 June 2023
Revised: 14 July 2023
Accepted: 18 July 2023
Published: 24 July 2023

1. Introduction

With the development of cloud computing technology, more and more individuals and enterprises choose to submit their data to cloud servers for computing, thereby reducing expenses. However, data sent to cloud servers face the risk of data leakage. If the user does not want to disclose private data to the third party, encrypting the data before transmission is an option, but traditional encryption technology does not support computation on ciphertext.

Homomorphic encryption (HE) is an effective solution to the above problems. It allows for the computation of the ciphertext without knowing the plaintext and the keys, and the calculation result of the ciphertext is homomorphic to the calculation result of the corresponding plaintext. While the third party only holds the encrypted data, it is difficult to obtain the plaintext information.

The idea of homomorphic encryption has been proposed for a long time, but because of the complexity of the construction, the ciphertext calculation algorithm and times are

very limited. Not until 2009 did Gentry [1] propose the first fully homomorphic encryption (FHE). This breakthrough has made homomorphic encryption more practical. After that, many branches of homomorphic encryption appeared, such as BGV [2], BFV [3,4], GSW [5], TFHE [6], CKKS [7], et al. However, these basic HE schemes are single key schemes, that is, ciphertexts participating in homomorphic computing are generated under one key. Actually, different parties cannot share a key to encrypt data, otherwise it is no different from directly sending plaintext. Therefore, each party should hold different keys in theory, and the cloud server can complete the homomorphic calculation of data encrypted with different keys, which is called multi-key homomorphic encryption (MKHE).

Lopez-Alt et al. [8] first proposed a multi-key homomorphic encryption scheme based on NTRU in 2012, and subsequently conducted a lot of research on MKHE in various homomorphic encryption branches. Among them, the earlier and most MKHE schemes [9–12] are implemented based on the GSW scheme. Then, in 2017, Chen et al. proposed the BGV type MKHE scheme [13] by combining the methods of BGV and GSW. In 2019, Chen et al. proposed the MKHE scheme [14] based on TFHE. However, the computational efficiency of these MKHE schemes is not ideal, which is the problem of MKHE schemes to be solved. And this is also a major reason why homomorphic encryption methods cannot be widely used in real scenarios.

At present, the key method for effective fully homomorphic computation is still the bootstrap method proposed by Gentry [1], which is used to refresh the noise of the ciphertext, so that the ciphertext can continue to participate in the next homomorphic evaluation. Specifically, the ciphertext of the homomorphic encryption scheme is added with noise, and the noise of the ciphertext will expand rapidly when the homomorphic evaluation is performed. When the noise increases beyond the specified range, the decryption of the ciphertext will fail, and the expected calculation result will not be achieved. The bootstrap method is to encrypt the ciphertext into another ciphertext when the noise of the ciphertext reaches the critical stage but does not exceed the limit, and the inner ciphertext is restored to the code of the plaintext by using the homomorphic evaluation decryption function to reduce the noise, so as to meet the homomorphic evaluation again. Then, by performing a bootstrapping every time the ciphertext noise reaches the upper limit, the circuit of any depth can be homomorphic, or "pure" fully homomorphic. But bootstrapping is also very expensive. Therefore, how to improve the efficiency of bootstrapping and make the multi-key fully homomorphic schemes more practical is an open problem.

Chillotti et al. proposed the fast bootstrapping HE schemes [6,15,16], which are called TFHE because their ciphertexts are mapped to torus. They are all based on the hardness assumptions of Learning with Errors (LWE) [17] and Ring-LWE (RLWE) [18]. Their schemes introduce external products, look-up tables and other methods in bootstrapping, so that the bootstrapping method has low latency and is better than other homomorphic encryption schemes in time and availability. Scheme [14] is a multi-key variant of TFHE, which adopts the advantages of TFHE to realize the MKHE scheme of fast bootstrapping. The scheme designs a method of a hybrid product, which is used for blind rotation in the calculation of multi-key ciphertexts, so that the bootstrapping is faster and the noise control is better. However, their scheme is only a basic MKHE scheme, the function of bootstrapping is relatively single, and every execution of the NAND gate circuit must run a bootstrapping operation. The efficiency is low in practice. After that, Chillotti et al. [19,20] improved TFHE, so that the plaintext message space was no longer limited to binary, and introduced the multiplication operation of BFV-type to improve the precision number of plaintext space. At the same time, the scheme satisfied the packing of ciphertexts and expanded the function of bootstrapping. It realizes multi-functional programmable bootstrapping (evaluates the function of the ciphertext while refreshing the ciphertext noise) and does not need to pad zero on the most significant bit of the ciphertext space. And a bootstrapping operation can homomorphically evaluate multiple functions, which makes the homomorphic computation more time-efficient. But these schemes are single-

key schemes. Therefore, how to construct bootstrapping methods that homomorphically evaluate multiple decryption functions in the MKHE schemes is an interesting question.

1.1. Contributions

In this paper, based on the single-key scheme of [19,20], we improve the MKHE scheme of scheme [14] and construct a new multi-key variant of TFHE. The advantages of fast bootstrapping of TFHE schemes are maintained, and more functions are added.

- Ciphertext packing. In our scheme, the space of encryptable plaintext messages has multiple bits. Each party encrypts the messages with its own key to generate the RLWE ciphertext. In contrast, scheme [14] can only perform encrypted computation on binary messages. According to the characteristics of the ciphertext slot, the number of messages that each party can encrypt is not limited to one, so as to realize the ciphertext packing of the scheme.
- Bootstrapping is not required for every homomorphic operation. In this paper, we separate the homomorphic operation of multi-key ciphertexts from bootstrapping. Consequently, it is not necessary to perform a bootstrapping to refresh the ciphertext noise after one multi-key ciphertext homomorphic computation, while the known multi-key TFHE schemes require a bootstrapping operation after each gate run.
- Programmable bootstrapping with multiple outputs. Since each party can encrypt the plaintext message space to achieve high accuracy, the Look-up table (LUT) of multiple functions can be set according to the plaintext space when performing multi-party joint computation. By doing this, you can homomorphically evaluate multiple functions while performing a single bootstrapping operation.

Finally, we provide the analysis of the operation of our scheme. Compared with the existing methods, our method maintains the same time complexity and realizes more functions of bootstrap operation. As we all know, bootstrapping is currently the only way to achieve fully homomorphic encryption, and it is also the most expensive method. As a result, our scheme can evaluate more functions with fewer bootstrapping times, which will greatly reduce the computational overhead. This is also the first attempt of programmable bootstrapping of multi-output functions in multi-key homomorphic encryption schemes.

1.2. Methodology Overview

The multi-key homomorphic encryption scheme in this paper is based on the standard LWE and RLWE hardness assumptions. $\mathbb{Z}$ is the set of integers, $\mathbb{B}$ is the set of $\{0,1\}$ and $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$ is the cyclotomic polynomial. The lowercase bold letters denote vectors (More notations are described in Section 2.1). The encrypted message is placed in the most significant bits of the ciphertext coefficient. Let the size of the ciphertext space be q. Due to the fact that bootstrapping refreshes the noise of LWE ciphertext, multi-key ciphertext computation operates on RLWE ciphertext. Therefore, each participant independently generates one LWE key and one RLWE key. Assuming there are k participants, the LWE key and RLWE key of the ith party are $\mathbf{s}_i \in \mathbb{B}^n$ and $t_i \in \mathbb{B}[X]/(X^N+1)$, respectively. The ith participant generates LWE ciphertext based on $\mathbf{s}_i$, and then packages it into RLWE ciphertext under t_i to send it out. The receiver expands a ciphertext into a multi-key ciphertext $\overline{\mathbf{ct}} = (b, a_1, \ldots, a_k) \in \mathcal{R}_q^{k+1}$ based on the number of participants, and decrypts it using a joint key $\mathbf{t} = (1, t_1, \ldots, t_k) \in \mathcal{R}_2^{k+1}$, namely $\mu \approx \lfloor \langle \overline{\mathbf{ct}}, \mathbf{t} \rangle \rceil_q$ where μ is the encoding associated with the plaintext message.

Homomorphic addition does not require special processing. Homomorphic product uses a tensor product similar to BFV, where the product of two ciphertexts is $\overline{\mathbf{ct}} \otimes \overline{\mathbf{ct}} \in \mathcal{R}_q^{(k+1)\times(k+1)}$. If the ciphertext is decrypted, the key $\mathbf{t} \otimes \mathbf{t} \in \mathcal{R}_2^{(k+1)\times(k+1)}$ is required. But the resulting ciphertext cannot be homomorphically evaluated again or decrypted directly using a joint key. Therefore, it is necessary to convert the ciphertext term corresponding to the nonlinear term $t_i \cdot t_j$ of the keys into the ciphertext of key $\mathbf{t}$, and the returned ciphertext $\overline{\mathbf{ct}}' \in \mathcal{R}_q^{k+1}$ satisfies $\langle \overline{\mathbf{ct}}', \mathbf{t} \rangle \approx \langle \overline{\mathbf{ct}} \otimes \overline{\mathbf{ct}}, \mathbf{t} \otimes \mathbf{t} \rangle$. Assume that all

parties share a Common Reference String (CRS) $\mathbf{a} \in \mathcal{R}_q^d$ and set up a gadget decomposition tool $\mathbf{g} \in \mathbb{Z}^d$. The ith party generates the public key $\mathbf{b}_i \approx -\mathbf{a} \cdot t_i \in \mathcal{R}_q^d$ and generates the ciphertext $\mathbf{F}_i = (\mathbf{f}_{i,0}, \mathbf{f}_{i,1}, \mathbf{f}_{i,2}) \in \mathcal{R}_q^{d\times 3}$ with the uni-encryption method. Then, through the ciphertext $\mathbf{F}_i$, the ciphertext item corresponding to $t_i \cdot t_j$ can be converted into three terms, corresponding to the keys $1, t_i$ and t_j, respectively. The noise of the ciphertext increases after homomorphic calculation. In order to decrypt correctly or continue to do homomorphic calculation, it is necessary to refresh the ciphertext noise by bootstrapping. Bootstrapping is accomplished by homomorphic computation of the decryption formula. First, set a test polynomial $P(X)$ and initialize the RLWE ciphertext. Because the dimension of the RLWE ciphertext polynomial is N, according to the reflexivity of the cyclotomic polynomial, the ciphertext space that can be refreshed most by bootstrapping is $2N$. Therefore, the ciphertext to be bootstrapped will undergo module switching, assuming it is $\mathbf{ct}^{'} = (b, a_{1,1}, \ldots, a_{1,N}, \ldots, a_{k,1}, \ldots, a_{k,N}) \in \mathbb{Z}_{2N}^{kN+1}$, and the corresponding decryption key is $\mathbf{t}^{'} = (1, t_{1,1}, \ldots, t_{1,N}, \ldots, t_{k,1}, \ldots, t_{k,N}) \in \mathbb{B}^{kN+1}$, that is, the coefficient combination of all participants' RLWE keys. If the coefficients of the test polynomial $P(X)$ are set as functions related to the exponent and homomorphically calculate $P(X) \cdot X^{-\langle \mathbf{ct}^{'}, \mathbf{t}^{'} \rangle}$, then the first term of the polynomial is the function related to the plaintext. To compute $\langle \mathbf{ct}^{'}, \mathbf{t}^{'} \rangle \approx b + a_{1,1} \cdot t_{1,1} + \cdots + a_{k,N} \cdot t_{k,N}$, it needs to know the keys; however, the keys are not publishable, so the keys $t_{i,j}$ are encrypted by uni-encryption to generate ciphertexts $\mathbf{F}_{i,j} = (\mathbf{f}_{i,j,0}, \mathbf{f}_{i,j,1}, \mathbf{f}_{i,j,2}) \in \mathcal{R}_q^{d\times 3}$, and then the hybrid product calculation is performed. The hybrid product homomorphically computes $t_{i,j} \cdot X^{a_{i,j}}$, and the keys generated in this paper follow uniform binary sampling, using the CMux gate to compute $(1 - t_{i,j}) + t_{i,j} \cdot X^{-a_{i,j}}$, so $X^{-\langle \mathbf{ct}^{'}, \mathbf{t}^{'} \rangle} = X^{-b} \cdot \prod_{i=1}^{k} \prod_{j=1}^{N} (1 - t_{i,j} + t_{i,j} \cdot X^{-a_{i,j}})$. The bootstrapping result is an LWE ciphertext, which is finally converted into an RLWE ciphertext using the multi-key switching keys, waiting for the next decryption or homomorphic computation.

This scheme can calculate multiple functions at the same time, mainly based on the design of the test polynomial $P(X)$ based on [20], and set the ϵ bits to zero in the least significant bits of the ciphertext space $2N$. Consequently, the decryption function $\langle \mathbf{ct}^{'}, \mathbf{t}^{'} \rangle = \triangle \cdot m + 2^{\epsilon} \cdot e$ is homomorphically calculated, where $\triangle$ is the scaling factor. Set the test polynomial $P(X)$ for the plaintext space p, extracting samples from the ciphertext of the decrypted function after homomorphic evaluation. This process results in 2^{ϵ} function ciphertexts related to m.

1.3. Related Works

MKHE scheme was first proposed by Lopez Alt et al. [8] to implement a dynamic Multi-Party Computation (MPC) protocol based on the NTRU public key cryptosystem. Subsequently, on the basis of the GSW scheme [5], Clear et al. [9] proposed an MKHE scheme supporting multi-identity, whose security is based on the standard assumption LWE problem [17]. Mukherjee et al. [10] simplified the scheme of [9] and used MKHE to construct MPC. The schemes of [9,10] need to preprocess the number of users involved in the homomorphic calculation, and are unable to add new users during the calculation process. This type is called single-hop. Then Peikert et al. [11] and Brakerski et al. [12] proposed the multi-hop MKHE scheme, respectively; however, Peikert et al.'s scheme limited the number of participants, and Brakerski et al.'s scheme extended the ciphertext via the bootstrapping method. The efficiency of the homomorphic operation is low. Chen et al. [14] constructed a multi-key variant of TFHE to achieve fast bootstrapping in the MKHE scheme, while also homomorphically evaluating a function in bootstrapping. However, this scheme can only encrypt binary numbers and does not support packaging technology, and bootstrapping is performed after each evaluation of the circuit gate. Chen et al. [13] and Li et al. [21] designed a multi-key variant of the BGV scheme by generating a linearized key based on the MK-GSW scheme, so that the plaintext space is not limited to the binary number set, and ciphertext packaging was achieved. Then Chen et al. [22] expanded their work, and constructed the MKHE scheme based on the homomorphic encryption scheme of

BFV and CKKS, which optimized the relinearization technology. This technology was more efficient in ciphertext computing, but did not achieve programmable bootstrapping. Bootstrapping is needed to implement any number of homomorphic operations. In the existing multi-key homomorphic encryption schemes, there are hierarchical homomorphic encryption operations, that is, the number of homomorphic operations is limited. Among the schemes that support bootstrapping, only the scheme of TFHE class supports programmable bootstrapping. However, the existing MKHE schemes only support homomorphic operation of binary gates, and there is only one output function of programmable bootstrapping. In the work of Chillotti et al. [20], it is proposed to introduce BFV-type ciphertext multiplication into TFHE and realize multi-output programmable bootstrapping at the same time. However, this scheme is a single-key scheme, that is, it can only calculate the ciphertext under the same key, and cannot be applied to the case of multi-party joint computation. To this end, we will optimize this scheme to implement unique functions in a multi-key homomorphic encryption scheme. In Table 1, the functional comparison between the proposed scheme and related MKHE schemes is summarized.

Table 1. Functional comparison of MKHE schemes.

Scheme	Hardness Assumption	Homomorphic Evaluation	Ciphertext Packaging	Multi-Output PBS
CCS19 [14]	LWE and RLWE	NAND gate	No	No
CDKS19 [22]	RLWE	Add and Mult	Yes	No
Ours	LWE and RLWE	Add and Mult	Yes	Yes

1.4. Organization

In Section 2, we define or reference some background knowledge, including the basic knowledge of sample extraction and look-up tables required by the bootstrapping module in Section 3. We describe the basic components of multi-key homomorphic encryption in this paper, including key-switching keys generation, ciphertext relinearization and hybrid product. In Section 4, we describe the construction of our multi-key homomorphic encryption scheme, including encryption and decryption of ciphertext, homomorphic evaluation and bootstrapping. In Section 5, we conduct an analysis, including a supplementary analysis of safety, noise, and performance. Some optimization directions are discussed in Section 6. The conclusion is provided in Section 7.

2. Background Knowledge

In this subsection, we mainly introduce the low-level construction knowledge of MKFHE.

2.1. Notation

Throughout this paper, we use $\mathbb{Z}$ to denote the set of integers, $\mathbb{B}$ as the set of $\{0,1\}$, bold lowercase letters to denote vectors, and bold uppercase letters to denote matrices. $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$ denotes the ring of integer polynomials modulo the cyclotomic polynomial X^N+1, where N is a power of two. For a positive integer q, we denote $\mathcal{R}_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N+1)$ for the integer polynomial ring $\mathcal{R}$ with coefficients modulo q, that is, the coefficient of the polynomial is reduced to $[-\frac{q}{2}, \frac{q}{2}) \cap \mathbb{Z}$, and generally $\mathbb{Z}/q\mathbb{Z}$ is expressed as $\mathbb{Z}_q$. $\mathcal{R}_q^n$ denotes the integer polynomial ring $\mathcal{R}_q$ with integer dimension n. Let ψ denote the uniform distribution over the set of integer polynomials, where the coefficients are value 0 or 1 and modulo the cyclotometric polynomial X^N+1. $\lfloor \cdot \rceil$ denotes rounding to the nearest integer value. $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the inner product of two vectors $\mathbf{a}$ and $\mathbf{b}$. For a positive integer k, set $[k] = \{1, 2, \ldots, k\}$ is denoted by an index set. If $\mathcal{D}$ is a probability distribution, we use $d \leftarrow \mathcal{D}$ to denote the sampling d according to distribution $\mathcal{D}$. Let $\mathcal{D}_\alpha$ be Gaussian distribution with variance α^2 where α is a small standard deviation. Let ω_α denote the random distribution over the set of integer polynomials, where the coefficient values are sampled over $\mathcal{D}_\alpha$. $\mathrm{U}(S)$ denotes the uniform distribution on S, which is a finite set. For two nonnegative real functions $f(n)$ and $g(n)$, denote $f(n) = \tilde{O}(g(n))$ if

there exists a positive constant c_1, c_2, N such that $f(n) \leq c_1 \cdot g(n) \cdot \log^{c_2} g(n)$ is satisfied for any $n \geq N$.

2.2. LWE and RLWE

The LWE problem and the RLWE problem were respectively introduced by Regev et al. [17], Lyubaskevsky et al. [18] and the simplified special-case version [23]. In TFHE, three main types of ciphertexts are used: LWE, RLWE and RGSW. RGSW is mainly used in the calculation of the external product of ciphertexts. In this paper, we do not use the external product, so we mainly use LWE and RLWE.

Theorem 1 (LWE Sample). *For security parameter λ, let $n = n(\lambda)$ be an integer dimension, let $q = q(\lambda) \geq 2$ be an integer, $\chi = \chi(\lambda)$ be a distribution over $\mathbb{Z}$ and $\alpha = \alpha(\lambda)$ be a discretized Gaussian parameter. For secret key $\mathbf{s}$ sampled uniformly over χ^n and error e is sampled uniformly over $\mathcal{D}_\alpha$. An LWE sample is a pair $(b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$, where $b = -\langle \mathbf{a}, \mathbf{s} \rangle + e \pmod q$ and $\mathbf{a}$ is sampled uniformly over $\mathbb{Z}_q^n$.*

Theorem 2 (RLWE Sample). *For security parameter λ, let $N = N(\lambda)$ be an integer with a power of 2, $q = q(\lambda) \geq 2$ be an integer, $\psi = \psi(\lambda)$ be a distribution over $\mathcal{R}$ and $\alpha = \alpha(\lambda)$ be a discretized Gaussian parameter. Secret key s is sampled uniformly over ψ. The error e is sampled uniformly over ω_α. An RLWE sample is a pair $(b, a) \in \mathcal{R}_q^2$, where $b = -a \cdot s + e \pmod q$ and a is sampled uniformly over $\mathcal{R}_q$.*

We define the following two problems based on LWE and RLWE samples:

- Search (R)LWE Problem: For a uniform random secret $\mathbf{s}$, given any number of LWE (or RLWE) independent sampling distribution, find the corresponding LWE secret (or RLWE secret).
- Decisional (R)LWE Problem: For a fixed LWE secret (or RLWE secret), the LWE (or RLWE) samples are distinguished from the samples sampled from $\mathbb{Z}_q^{n+1}$ (or $\mathcal{R}_q^2$) uniform distribution.

Lemma 1. *In LWE and RLWE problems, there is no difference between a random vector and a vector $(b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$ or $(b, a) \in \mathcal{R}_q^2$ from our perspective. We cannot obtain any valuable information here, so we consider the Decisional (R)LWE Problem is hard.*

We assume that the message $m \in \mathbb{Z}_q$. Add the message to b. So, $b = -\langle \mathbf{a}, \mathbf{s} \rangle + m + e \pmod q$, and obtain the LWE ciphertext $(b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$ of m. For the same assumption message $m \in \mathcal{R}_q$, we add the message to b that $b = -a \cdot s + m + e \pmod q$, resulting in the RLWE ciphertext $(b, a) \in \mathcal{R}_q^2$ of m. An additional scaling factor is added when encrypting in this paper to store the plaintext message in the most significant bits.

For one LWE ciphertext $\mathbf{c} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$ and a key $\mathbf{s} \in \mathbb{B}^n$, we define the phase function $\varphi_{\mathbf{s}}(\mathbf{c})$ as $\varphi_{\mathbf{s}}(\mathbf{c}) = b + \langle \mathbf{a}, \mathbf{s} \rangle \pmod q$. When decrypting, we use $\mathbf{c}$ and $\mathbf{s}$ to calculate $\varphi_{\mathbf{s}}(\mathbf{c}) = \langle \mathbf{c}, (1, \mathbf{s}) \rangle$, and then approximately solve for the message $m \in \mathbb{Z}_q$. The same can be done similarly for RLWE ciphertexts.

2.3. Part of the Components of TFHE

2.3.1. Gadget Decomposition

The gadget decomposition tool can approximate large numbers and effectively control the growth of noise in homomorphic evaluation. We define the gadget decomposition as a function from $\mathcal{R}_q$ to $\mathcal{R}^d$. Let $\mathbf{g} = (B^0, \ldots, B^{d-1}) \in \mathbb{Z}^d$ be a gadget vector where base B is an integer and d is degree. Suppose there is a polynomial ring element $a = \mathcal{R}_q$, and a small polynomial vector $\mathbf{g}^{-1}(a) = (u_0, \ldots, u_{d-1}) \in \mathcal{R}^d$ is obtained by gadget decomposition, where $u_i \in [-\frac{B}{2}, \frac{B}{2})$. So, $a = \mathbf{g} \cdot \mathbf{g}^{-1}(a) = \sum_{i=0}^{d-1} B^i \cdot u_i \pmod q$.

2.3.2. Modulus Switching

Modulus switching mainly changes the modulus of the ciphertext into a different modulus. For two moduli $2N$ and q, the LWE ciphertext $\mathbf{ct} \in \mathbb{Z}_q^{n+1}$ is input, and the LWE ciphertext $\mathbf{ct}' \in \mathbb{Z}_{2N}^{n+1}$ is output after modulus switching, without changing the size of the plaintext message and the key.

2.3.3. Sample Extract

The sample extract algorithm, based on an input index i, extracts an LWE ciphertext, which is an LWE encryption of the constant coefficient of the ith term of the polynomial $\sum_{i=0}^{N-1} m_i X^i$. Specifically, this algorithm is called RLWE-to-LWE. Assuming that $t \in \mathcal{R}_q^2$ is an RLWE secret and $\mathbf{s} \in \mathbb{Z}_q^{N+1}$ is an LWE secret, where $\mathbf{s}$ is the combination of correlation coefficients extracted from t, the algorithm does not add additional noise. The expression for this operation is as follows: $SampleExtract_i(RLWE_t(\sum_{i=0}^{N-1} m_i X^i)) \rightarrow LWE_{\mathbf{s}}(m_i)$.

2.3.4. Look-Up Table

This is used primarily to represent the function $f : \mathbb{Z}_N \rightarrow \mathbb{Z}_q$. The test polynomial $F = f_0 + f_1 X + \ldots + f_{N-1} X^{N-1}$ is encoded using Look-Up Table (LUT) and then it is encapsulated into an RLWE ciphertext. During the bootstrapping, the ciphertext is evaluated assuming that $F \cdot X^{-i}$, where i is the homomorphic decryption function of the ciphertext, and the LWE ciphertext is extracted at position '0' through sample extract, revealing that this is the LWE ciphertext with plaintext message f_i. By constructing an appropriate LUT, a function can be evaluated during bootstrapping refresh of ciphertext. This article references the construction method of [20] to achieve the evaluation of multiple functions f during one bootstrapping operation of LWE ciphertext in multi-party computations.

2.4. Multi-Key Homomorphic Encryption

A multi-key homomorphic encryption system allows the computation of ciphertexts encrypted with different keys. Let $\mathcal{M}$ be the message space with arithmetic structure. A multi-key homomorphic encryption scheme MKHE consists of five PPT algorithms (Setup, KeyGen, Enc, Dec, Eval). Assume that an index id is set to each party.

- **Setup**: $pp \leftarrow MKHE.Setup(1^\lambda)$. Takes the security parameter λ as an input, returns the public parameter pp.
- **Key Generation**: $(sk,\ pk) \leftarrow MKHE.KeyGen(pp)$. Generates a pair of private keys and public keys. We assume that the private keys and public keys set the index id corresponding to each party.
- **Encryption**: $\overline{\mathbf{ct}}_{id} \leftarrow MKHE.Enc(\mu_{id}, pk_{id})$. Encrypts a message $\mu_{id} \in \mathcal{M}$ and returns a ciphertext $\overline{\mathbf{ct}}_{id} \in \{0,\ 1\}^*$. Similarly, we assume that the index id of each ciphertext corresponds to the ciphertext under the corresponding key.
- **Decryption**: $\mu \leftarrow MKHE.Dec(\overline{\mathbf{ct}}, \{sk_{id}\}_{id\in[k]})$. Given a ciphertext $\overline{\mathbf{ct}}$ with the corresponding sequence of secret keys $\{sk_{id}\}_{id\in[k]}$. Decrypts the ciphertext into a message $\mu \in \mathcal{M}$.
- **Homomorphic Evaluation**: $\overline{\mathbf{ct}} \leftarrow MKHE.Eval(\mathcal{C}, \{\overline{\mathbf{ct}}_{id}\}_{id\in[k]}, \{pk_{id}\}_{id\in[k]})$. Given a circuit $\mathcal{C}$ and multi-key ciphertexts $\overline{\mathbf{ct}}_1, \ldots, \overline{\mathbf{ct}}_k$ with the corresponding set of public keys $pk_1, \ldots, pk_k$, it returns a ciphertext $\overline{\mathbf{ct}}$. We assume that the output ciphertext contains information about the relevant parties involved.

Correctness. For $1 \leq id \leq k$, according to $\overline{\mathbf{ct}}_{id} \leftarrow MKHE.Enc(\mu_{id}, pk_{id})$, the ciphertexts of k parties are generated. If the ciphertext of any party is decrypted directly, the $\mu_{id} \leftarrow MKHE.Dec(\overline{\mathbf{ct}}_{id}, \{sk_{id}\}_{id\in[k]})$ can be obtained. Let $\mathcal{C} : \mathcal{M}^k \rightarrow \mathcal{M}$ be a circuit, the $\overline{\mathbf{ct}}$ is obtained by ciphertext computation $MKHE.Eval(\mathcal{C}, \{\overline{\mathbf{ct}}_{id}\}_{id\in[k]}, \{pk_{id}\}_{id\in[k]})$ on the ciphertext of the k party according to the circuit $\mathcal{C}$. Then, the computed ciphertext is decrypted $MKHE.Dec(\overline{\mathbf{ct}}_{id}, \{sk_{id}\}_{id\in[k]})$ to obtain $\mathcal{C}(m_1, \ldots, m_k)$ with an overwhelming probability; we call this MKHE scheme correct.

Semantic Security. Assuming we have any two messages, $\mu_1, \mu_2 \in \mathcal{M}$. As parameters $MKHE.Setup(1^\lambda)$ and keys $MKHE.KeyGen(pp)$ are generated, distributions between two ciphertexts $\{MKHE.Enc(\mu_{id\in\{1,2\}}), pk_{id\in\{1,2\}}\}$ should be computationally indistinguishable.

3. The Building Blocks of Basic Scheme

This section describes the basic building blocks for building MKHE in LWE and RLWE, including key switching, relinearization, and hybrid product.

3.1. Basic Modules for LWE Ciphertext and RLWE Ciphertext

This section first describes the parameter settings, basic encryption methods, and key switching for the generation of LWE ciphertext and RLWE ciphertext.

- Setup(1^λ): Given λ as the input security parameter, generate the dimension n of the LWE, the uniform distribution χ, Gaussian distribution parameter α, the ciphertext modulus q, and set the variable ϵ. Generate the dimension N of RLWE, the key distribution ψ, Gaussian distribution parameter α and the ciphertext modulus q. Set a CRS $\mathbf{a} \leftarrow \mathrm{U}(\mathcal{R}_q^d)$, let the LWE public parameter $pp = (n, \chi, q, \alpha, \epsilon)$ and RLWE public parameter $pp^{'} = (N, \psi, \alpha, q, \mathbf{a})$, returns parameter $pp'' = (pp, pp')$.

In this paper, the bootstrapping algorithm is performed on the LWE ciphertext to refresh the ciphertext noise. The variable ϵ selects the refreshed bits during module switching, and 2^ϵ represents the number of functions that can be output in batch during bootstrapping. Our basic scheme is to build on the CRS model and obtain the vector $\mathbf{a} \in \mathcal{R}_q^d$ by sampling according to the generated public parameters pp'. We assume that any party generates keys and ciphertexts based on common parameters as input, so as to support arithmetic operations between ciphertexts under different keys.

- KeyGen(pp''): Sample the LWE secret $\mathbf{s} \leftarrow \chi^n$, set the LWE secret key $\mathbf{s}' = (1, \mathbf{s}) \in \mathbb{Z}_q^{n+1}$. Sample the RLWE secret $t \leftarrow \psi$, set the RLWE secret key $\mathbf{t}' = (1, t) \in \mathcal{R}_q^2$. Sample $\mathbf{e} \leftarrow \omega_\alpha^d$ as an error vector and set the $\mathbf{b} = -t \cdot \mathbf{a} + \mathbf{e} \pmod q \in \mathcal{R}_q^d$ as a public key. Returns the triple $(\mathbf{s}, t, \mathbf{b})$.

The coefficients of the MKHE basic keys can be sampled from uniform distribution or Gaussian distribution, and the keys used in this paper mainly follow uniform binary distribution sampling. If different sampling methods are used to generate this scheme, replace the appropriate parameters and modify the CMux gate described in the next section.

- Enc($m, \mathbf{s}$): To encrypt a message $m \in \mathbb{Z}_p$. This is the standard LWE encryption. Generate samples $\mathbf{a} \leftarrow \mathrm{U}(\mathbb{Z}_q^n)$ and $e \leftarrow \mathcal{D}_\alpha$. Let $b = -\langle \mathbf{a}, \mathbf{s} \rangle + e + \triangle \cdot m \pmod q$ and returns the ciphertext $\mathbf{ct} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$.

Suppose q is a ciphertext space, p is a plaintext space, and $p < q$. Load the plaintext message into the most significant bits of the ciphertext space, and then add noise to the least significant bits. Therefore, the scaling factor is $\triangle = \frac{q}{p}$, as long as the noise in the ciphertext does not change the plaintext message, it can be decrypted normally, namely $|e| < \frac{\triangle}{2}$.

- SwitchKeyGen($\mathbf{s}_1, s_2$): Generate LWE-to-RLWE key-switching keys. Enter the LWE key $\mathbf{s}_1 = (s_{1,1}, \ldots, s_{1,n}) \in \mathbb{Z}_q^n$ and the RLWE key $s_2 \in \mathcal{R}_q$. For $i \in [n]$, generate sample $\mathbf{A}_i \leftarrow \mathrm{U}(\mathcal{R}_q^d)$ and $\mathbf{e}_i \leftarrow \omega_\alpha^d$, let $\mathbf{b}_i = -\mathbf{A}_i s_2 + s_{1,i} \cdot \mathbf{g} + \mathbf{e}_i \pmod q$, makes $\mathbf{KS}_i = [\mathbf{b}_i | \mathbf{A}_i] \in \mathcal{R}_q^{d\times 2}$, and return the key-switching keys $\mathbf{KSK} = \{\mathbf{KS}_i\}_{i\in[n]} \in (\mathcal{R}_q^{d\times 2})^n$.

Security. The ith term $\mathbf{KS}_i$ of the key-switching keys adds the value related to the ith term of the LWE key $\mathbf{s}_1 \in \mathbb{Z}_q^n$ to the product of the uniform distribution $\mathbf{A}_i \in \mathcal{R}_q^d$ and the RLWE key $s_2 \in \mathcal{R}_q$ in the first column, and adds noise $\mathbf{e}_i$. This is similar to encrypting the ith term of the LWE key $\mathbf{s}_1$ under the RLWE key s_2 to form a RLWE ciphertext. Assuming that the key-switching keys items are sampled according to the RLWE parameter (N, ψ, α), the RLWE decision problem shows that the advantage of distinguishing the key-switching

keys $\mathbf{KS}_i = [\mathbf{b}_i|\mathbf{A}_i] \in \mathcal{R}_q^{d\times 2}$ from the independent uniform distribution $\mathrm{U}(\mathcal{R}_q^{d\times 2})$ is almost negligible. It is difficult to extract the information of the LWE key $s_{1,i}$ from $\mathbf{KS}_i$.

- PKSwitch($\{\mathbf{ct}_i\}_{i=1}^p, \{id_i\}_{i=1}^p, \mathbf{KSK}$): Given the LWE-to-RLWE packing key $\mathbf{KSK}$, p LWE ciphertexts $\{\mathbf{ct}_i\}_{i=1}^p = \{(b_i, \mathbf{a}_i)\}_{i=1}^p \in (\mathbb{Z}_q^{n+1})^p$ and p corresponding index id_i for $i \in [p]$, packing the LWE ciphertexts into a RLWE ciphertext. Compute $(b_i^{'}, a_i^{'}) = \sum_{j=1}^n \mathbf{g}^{-1}(a_{i,j})\cdot\mathbf{KS}_j \cdot X^{id_i} \pmod q$, let $b = \sum_{i=1}^p b_i \cdot X^{id_i} + \sum_{i=1}^p b_i^{'} \pmod q$, $a = \sum_{i=1}^p a_i^{'} \pmod q$. Return the packaged RLWE ciphertext $\mathbf{ct} = (b, a) \in \mathcal{R}_q^2$.

Proof. Assuming $\mathbf{ct}_i = (b_i, \mathbf{a}_i) \in \mathbb{Z}_q^{n+1}$ is a ciphertext about the same LWE key $\mathbf{s} \in \mathbb{Z}_q^n$, set the index to id_i, where $i \in [p]$. $\mathbf{SK} = \{\mathbf{KS}_i\}_{i\in[n]} \in (\mathcal{R}_q^{d\times 2})^n$ are the key-switching keys from $\mathbf{s}$ to $t \in \mathcal{R}_q$. The following will list the correctness of the packing of the ciphertext calculation:

$$\begin{aligned}
\sum_{i=1}^p \langle \mathbf{ct}_i, (1, \mathbf{s})\rangle \cdot X^{id_i} =& \sum_{i=1}^p (b_i + \sum_{j=1}^n a_{i,j} \cdot s_j) \cdot X^{id_i} \\
=& \sum_{i=1}^p b_i \cdot X^{id_i} + \sum_{i=1}^p \sum_{j=1}^n a_{i,j} \cdot s_j \cdot X^{id_i} \\
\approx& \sum_{i=1}^p b_i \cdot X^{id_i} + \sum_{i=1}^p \sum_{j=1}^n \langle \mathbf{g}^{-1}(a_{i,j}), s_j \cdot \mathbf{g}\rangle \cdot X^{id_i} \\
\approx& \sum_{i=1}^p b_i \cdot X^{id_i} + \sum_{i=1}^p \sum_{j=1}^n \langle \mathbf{g}^{-1}(a_{i,j}) \cdot \mathbf{KS}_j, (1, t)\rangle \cdot X^{id_i} \\
=& \sum_{i=1}^p b_i \cdot X^{id_i} + \sum_{i=1}^p b_i^{'} + \sum_{i=1}^p a_i^{'} \cdot t \\
\approx& \langle \mathbf{ct}, (1, t)\rangle \pmod q.
\end{aligned}$$

□

- MKSwitch($\overline{\mathbf{ct}}, \{\mathbf{KSK}_i\}_{i\in[k]}$): Given the LWE ciphertext $\overline{\mathbf{ct}} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{Z}_q^{kN+1}$ under the concatenated key and a sequence of key-switching keys $\{\mathbf{KSK}_i\}_{i\in[k]}$, let $(b_i^{'}, \mathbf{a}_i^{'}) = \sum_{j=1}^N \mathbf{g}^{-1}(a_{i,j})\cdot\mathbf{KS}_{i,j} \pmod q$ for $i \in [k]$ and $b^{''} = b + \sum_{i=1}^k b_i^{'} \pmod q$, $\mathbf{a}_i^{''} = \sum_{i=1}^k \mathbf{a}_i^{'} \pmod q$. Returns the RLWE ciphertext $\overline{\mathbf{ct}}^{'} = (b^{''}, \mathbf{a}_1^{''}, \ldots, \mathbf{a}_k^{''}) \in \mathcal{R}_q^{k+1}$ after the key-switching.

Proof. Suppose that the LWE ciphertext under the concatenated key $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$ is $\overline{\mathbf{ct}} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{Z}_q^{kN+1}$, where $\mathbf{a}_i = (a_{i,1}, \ldots, a_{i,N}) \in \mathbb{Z}_q^N$ for $i \in [k]$, $\{\mathbf{KSK}_i\}_{i\in[k]}$ are the key-switching keys generated by k participants respectively, and the key of the LWE ciphertext is transformed from $\mathbf{s} \in \mathbb{Z}_q^{kN}$ to $\mathbf{t} = (t_1, \ldots, t_k) \in \mathcal{R}_q^k$ without changing the plaintext message. The correctness of the multi-key-switching calculation is listed below:

$$\begin{aligned}
\langle \overline{\mathbf{ct}}, (1, \mathbf{s})\rangle =& b + \sum_{i=1}^k \sum_{j=1}^N a_{i,j} \cdot s_{i,j} \approx b + \sum_{i=1}^k \sum_{j=1}^N \langle \mathbf{g}^{-1}(a_{i,j}), s_{i,j} \cdot \mathbf{g}\rangle \\
\approx& b + \sum_{i=1}^k \sum_{j=1}^N \langle \mathbf{g}^{-1}(a_{i,j}) \cdot \mathbf{KS}_{i,j}, (1, t_i)\rangle = b + \sum_{i=1}^k b_i^{'} + \sum_{i=1}^k \sum_{j=1}^N a_{i,j}\cdot t_i \\
=& b + \sum_{i=1}^k b_i^{'} + \sum_{i=1}^k \mathbf{a}_i^{'} t_i \approx \langle \overline{\mathbf{ct}}^{'}, (1, \mathbf{t})\rangle \pmod q.
\end{aligned}$$

□

- UniEnc(m, t): Given a RLWE key $t \in \mathcal{R}_q$, enter a plaintext message $m \in \mathcal{R}_q$. Generate the ciphertext $\mathbf{F} = [\mathbf{f}_0|\mathbf{f}_1|\mathbf{f}_2] \in \mathcal{R}_q^{d\times 3}$ as follows:

1. Sample $r \leftarrow \psi, \mathbf{f}_1 \leftarrow \mathrm{U}(\mathcal{R}_q^d)$ and error $\mathbf{e}_1 \leftarrow \omega_\alpha^d$. Set $\mathbf{f}_0 = -t \cdot \mathbf{f}_1 + r \cdot \mathbf{g} + \mathbf{e}_1 \pmod q \in \mathcal{R}_q^d$;
2. Sample error $\mathbf{e}_2 \leftarrow \omega_\alpha^d$, set $\mathbf{f}_2 = r \cdot \mathbf{a} + m \cdot \mathbf{g} + \mathbf{e}_2 \pmod q \in \mathcal{R}_q^d$.

This algorithm is a symmetric encryption, which can encrypt a ring element, and the generated ciphertext consists of three polynomial vectors. Compared with the general RGSW ciphertext in $\mathcal{R}_q^{2d\times 2}$, its ciphertext size is about a quarter smaller. The first two columns of the ciphertext can be viewed as encrypting r with the key t, and the third column can be viewed as encrypting the message m with r, where r follows the ψ distribution. We will use uni-encryption to perform tensor products of multiple keys and hybrid products for bootstrapping.

Security. First, given a plaintext message $m \in \mathcal{R}_q$, let RLWE parameters be (N, ψ, α, q) and declare a distribution $\mathcal{D}_0 = \{(\mathbf{a}, \mathbf{b}, \mathbf{f}_0, \mathbf{f}_1, \mathbf{f}_2) : pp^{'} \leftarrow \mathrm{Setup}(1^\lambda), \ \mathbf{a} \leftarrow \mathrm{U}(\mathcal{R}_q^d), (t, \mathbf{b}) \leftarrow \mathrm{KeyGen}(pp^{''}), [\mathbf{f}_0|\mathbf{f}_1|\mathbf{f}_2] \leftarrow \mathrm{UniEnc}(m, t)\}$ over $\mathcal{R}_q^{d\times 5}$, denote by CRS, public key, and uni-encryption of m. Because the first four items are related to the RLWE key t, and $\mathbf{f}_2$ is independent of the RLWE key t. According to the hardness of the RLWE problem, we can change the definition of the first four items and reveal that $\mathcal{D}_0$ is computationally indistinguishable from distribution $\mathcal{D}_1 = \{(\mathbf{a}, \mathbf{b}, \mathbf{f}_0, \mathbf{f}_1, \mathbf{f}_2) : \mathbf{a}, \mathbf{b}, \mathbf{f}_0, \mathbf{f}_1 \leftarrow \mathrm{U}(\mathcal{R}_q^d), \mathbf{f}_2 = r \cdot \mathbf{a} + m \cdot \mathbf{g} + \mathbf{e}_2 \pmod q\}$ over $\mathcal{R}_q^{d\times 5}$. Then, because $r \leftarrow \psi$ follows the same distribution as the RLWE key t, we also change the definition of $\mathbf{f}_2$ according to the hardness of the RLWE problem and get that $\mathcal{D}_1$ is computationally indistinguishable from distribution $\mathcal{D}_2 = \{(\mathbf{a}, \mathbf{b}, \mathbf{f}_0, \mathbf{f}_1, \mathbf{f}_2) : \mathbf{a}, \mathbf{b}, \mathbf{f}_0, \mathbf{f}_1, \mathbf{f}_2 \leftarrow \mathrm{U}(\mathcal{R}_q^d)\}$ over $\mathcal{R}_q^{d\times 5}$. Observe that the uniform distribution $\mathcal{D}_2$ is independent of the given plaintext message m, so it can be considered that the uni-encryption scheme is semantically secure.

3.2. Relinearization and Hybrid Product

This paper proposes to use uni-encryption to calculate the product of multiple parties' extended ciphertexts and the key part of bootstrapping. In the case of calculating these ciphertexts with different keys, a uni-encryption scheme can be effectively homomorphic, so that each party's keys meet the semantic security standards. This section will introduce the two-part components proposed in this paper.

- RLKeyGen(t): Given a key $t \in \mathcal{R}_q$. Calculate and return $\mathbf{RLK} \leftarrow \mathrm{UniEnc}(t, t)$.
- ReLin($\overline{\mathbf{ct}}, \{(\mathbf{RLK}_i, \mathbf{b}_i)\}_{i\in[k]}$): Input a multi-key RLWE ciphertext $\overline{\mathbf{ct}} \in \mathcal{R}_q^{(k+1)\times(k+1)}$, and the relinearization keys and public keys $\{\mathbf{RLK}_i = [\mathbf{f}_{i,0}|\mathbf{f}_{i,1}|\mathbf{f}_{i,2}], \mathbf{b}_i\}_{i\in[k]}$ of the k participants. Assuming ciphertext $\overline{\mathbf{ct}} = (ct_{i,j})_{0\le i,j\le k}$, let $\mathbf{b}_0 = -\mathbf{a}$. The calculation of the multiplicity of ciphertext associated with the kth party concatenated key follows the following method:
  Let $v_{i,j} = \langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{b}_j\rangle \pmod q$, $ct_0^{'} \leftarrow ct_{0,0}$, $ct_i^{'} \leftarrow ct_{i,0} + ct_{0,i}$ for $i \in [k]$. And then for $i, j \in [k]$, iterative computations $ct_0^{'} = ct_0^{'} + \langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,0}\rangle \pmod q$, $ct_i^{'} = ct_i^{'} + \langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,1}\rangle \pmod q$ and $ct_j^{'} = ct_j^{'} + \langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{f}_{i,2}\rangle \pmod q$. Returns the ciphertext $\overline{\mathbf{ct}}^{'} = (ct_0^{'}, ct_1^{'}, \ldots, ct_k^{'}) \in \mathcal{R}_q^{k+1}$ after the multiplicative re-linear product.

Proof. Assuming that the participants have a total of k parties, the concatenated key is $\mathbf{t} = (t_1, \ldots, t_k) \in \mathcal{R}_q^k$, given two concatenated ciphertexts, $\overline{\mathbf{ct}}_1 = (c_{1,0}, \ldots, c_{1,k}) \in \mathcal{R}_q^{k+1}$ and $\overline{\mathbf{ct}}_2 = (c_{2,0}, \ldots, c_{2,k}) \in \mathcal{R}_q^{k+1}$ under the concatenated key, where $c_{1,1}$ and $c_{2,1}$ are plaintext message items b_1 and b_2 of RLWE ciphertext, respectively. $\{(\mathbf{RLK}_i = [\mathbf{f}_{i,0}|\mathbf{f}_{i,1}|\mathbf{f}_{i,2}], \mathbf{b}_i)\}_{i\in[k]}$ are the relinearization keys and public keys published by k parties. Let $ct_{i,j} = c_{1,i} \cdot c_{2,j} \pmod q \in \mathcal{R}_q$, so $\overline{\mathbf{ct}}_1 \otimes \overline{\mathbf{ct}}_2 = (ct_{i,j})_{0\le i,j\le k} \in \mathcal{R}_q^{(k+1)\times(k+1)}$. According to the component, $\overline{\mathbf{ct}}^{'} \in \mathcal{R}_q^{(k+1)}$ is initialized first. And then add $\sum_{i=1}^k \sum_{j=1}^k \langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,0}\rangle$ to the first term, add $\sum_{i=1}^k \sum_{j=1}^k \langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,1}\rangle$ and $\sum_{i=1}^k \sum_{j=1}^k \langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{f}_{i,2}\rangle$ to the last k entries, respectively. In the iterative calculation of each term, $\langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,0}\rangle + \langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,1}\rangle \cdot t_i \approx v_{i,j} \cdot r_i \pmod q$, $\langle \mathbf{g}^{-1}(ct_{i,j}\mathbf{f}_{i,2}\rangle \cdot t_j \approx \mathbf{g}^{-1}(ct_{i,j}) \cdot r_i \cdot a \cdot t_j + ct_{i,j} \cdot t_i \cdot t_j \approx -v_{i,j} \cdot r_i +$

$ct_{i,j} \cdot t_i \cdot t_j \pmod q$, so $(\langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,0}\rangle, \langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,1}\rangle, \langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{f}_{i,2}\rangle) \cdot (1, t_i, t_j) \approx ct_{i,j} \cdot t_i \cdot t_j \pmod q$. The correctness of the ciphertext relinearization is calculated as follows:

$$\begin{aligned}\langle \overline{\mathbf{ct}}^{'}, (1,\mathbf{t})\rangle =& ct_0^{'} + \sum_{i=1}^{k} ct_i^{'} \cdot t_i = ct_{0,0} + \sum_{i=1}^{k}\sum_{j=1}^{k}\langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,0}\rangle \\ &+ (\sum_{i=1}^{k}(ct_{i,0} + ct_{0,i}) + \sum_{i=1}^{k}\sum_{j=1}^{k}\langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,1}\rangle) \cdot t_i + \sum_{i=1}^{k}\sum_{j=1}^{k}\langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{f}_{i,2}\rangle \cdot t_j \\ =& ct_{0,0} + \sum_{i=1}^{k}(ct_{i,0} + ct_{0,i}) \cdot t_i + \sum_{i=1}^{k}\sum_{j=1}^{k} ct_{i,j} \cdot t_i \cdot t_j \\ =& \langle \overline{\mathbf{ct}}_1 \otimes \overline{\mathbf{ct}}_2, (1,\mathbf{t}) \otimes (1,\mathbf{t})\rangle \pmod q.\end{aligned}$$

□

- Prod($\overline{\mathbf{ct}}, \mathbf{F}_i, \{\mathbf{b}_j\}_{j\in[k]}$): Input multi-key RLWE ciphertext $\overline{\mathbf{ct}} \in \mathcal{R}_q^{k+1}$ and the ith party's uni-encryption ciphertext $\mathbf{F}_i \leftarrow \text{UniEnc}(m, t_i)$ and k participants public keys $\{\mathbf{b}_j\}_{j\in[k]}$. Assuming RLWE ciphertext $\overline{\mathbf{ct}} = (ct_0, ct_1, \ldots, ct_k) \in \mathcal{R}_q^{k+1}$, let $\mathbf{b}_0 = -\mathbf{a}$, for $0 \le i \le k$. The calculation follows the following method:
  Let $v_j = \langle \mathbf{g}^{-1}(ct_j), \mathbf{b}_j\rangle \pmod q$, return the ciphertext after the hybrid product $\overline{\mathbf{ct}}^{'} = (ct_0^{'}, ct_1^{'}, \ldots, ct_k^{'}) \in \mathcal{R}_q^{k+1}$, where $ct_0^{'} = \langle \mathbf{g}^{-1}(ct_0), \mathbf{f}_{i,2}\rangle + \sum_{j=0}^{k}\langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,0}\rangle \pmod q$, $ct_i^{'} = \langle \mathbf{g}^{-1}(ct_i), \mathbf{f}_{i,2}\rangle + \sum_{j=0}^{k}\langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,1}\rangle \pmod q$ and $ct_j^{'} = \langle \mathbf{g}^{-1}(ct_j), \mathbf{f}_{i,2}\rangle \pmod q$ for $j \in [k]\backslash\{i\}$.

Proof. Assuming that the participants have a total of k parties, the concatenated key is $(1,\mathbf{t}) = (t_0 = 1, t_1, \ldots, t_k) \in \mathcal{R}_q^{k+1}$. According to the component conditions, the first term of the hybrid product output result is $\langle \mathbf{g}^{-1}(ct_0), \mathbf{f}_{i,2}\rangle + \sum_{j=0}^{k}\langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,0}\rangle \pmod q$, the ith term is $\langle \mathbf{g}^{-1}(ct_i), \mathbf{f}_{i,2}\rangle + \sum_{j=0}^{k}\langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,1}\rangle \pmod q$, the remaining $(k-1)$ terms are $\langle \mathbf{g}^{-1}(ct_j), \mathbf{f}_{i,2}\rangle \pmod q$, respectively. Because of the $\langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,0}\rangle + \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,1}\rangle \cdot t_i \approx v_j \cdot r_i \pmod q$, $\langle \mathbf{g}^{-1}(ct_i), \mathbf{f}_{i,2}\rangle \cdot t_j \approx \mathbf{g}^{-1}(ct_i) \cdot r_i \cdot a \cdot t_j + ct_{i,j} \cdot m \cdot t_j \approx -v_j \cdot r_i + ct_i \cdot m \cdot t_j \pmod q$, the correctness of the ciphertext hybrid product is computed as follows:

$$\begin{aligned}\langle \overline{\mathbf{ct}}^{'}, (1,\mathbf{t})\rangle =& \sum_{j=0}^{k} ct_j^{'} \cdot t_j = \sum_{j=0}^{k}\langle \mathbf{g}^{-1}(ct_j, \mathbf{f}_{i,2}\rangle \cdot t_j + \sum_{j=0}^{k}\langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,0}\rangle + \sum_{j=0}^{k}\langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,1}\rangle \cdot t_i \\ \approx& \sum_{j=0}^{k} ct_j \cdot m \cdot t_j = m \cdot \langle \overline{\mathbf{ct}}, (1,\mathbf{t})\rangle \pmod q.\end{aligned}$$

□

- CMux($\overline{\mathbf{ct}}_1, \overline{\mathbf{ct}}_2, \mathbf{F}_i, \{\mathbf{b}\}_{j\in[k]}$): Given two multi-key RLWE ciphertexts $\overline{\mathbf{ct}}_1, \overline{\mathbf{ct}}_2 \in \mathcal{R}_q^{k+1}$, and the uni-encrypted ciphertext $\mathbf{F}_i$ of the ith party and the public keys $\{\mathbf{b}\}_{j\in[k]}$ of the k parties. Return the ciphertext $\overline{\mathbf{ct}}^{'} = \overline{\mathbf{ct}}_1 + \text{Prod}(\overline{\mathbf{ct}}_2 - \overline{\mathbf{ct}}_1, \mathbf{F}_i, \{\mathbf{b}_j\}_{j\in[k]})$.

As described above, our keys sampling follows a uniform binary distribution, and the CMux gate chooses the output $\overline{\mathbf{ct}}_1$ or $\overline{\mathbf{ct}}_2$ according to the key $t_i \in \mathbb{B}$.

4. MKHE with Multi-Output Bootstrap

4.1. Description

This section describes the multi-key scheme with single bootstrapping and multi-output for the proposed scheme.

- MKHE.Setup(1^λ): Given the security parameter λ. Run Setup(1^λ) to generate the LWE public parameter $pp = (n, \chi, q, \alpha, \epsilon)$ and the RLWE public parameter $pp' = (N, \psi, \alpha, q, \mathbf{a})$. Return the public parameter $pp'' = (pp, pp')$.

- MKHE.KeyGen(p''): Suppose that each party independently generates its own keys based on the input parameter pp'' and follows the following method:
  1. Run KeyGen(p'') to generate the LWE secrets, RLWE secrets and public keys as the triple $(\mathbf{s}_i, t_i, \mathbf{b}_i)$. Assuming $t_i = t_{i,0} + t_{i,1}X + \ldots + t_{i,N-1}X^{N-1}$, let $\mathbf{t}_i^{'} = (t_{i,0}, t_{i,1}, \ldots, t_{i,N-1})$ and $\mathbf{PK}_i = \mathbf{b}_i$. Return the LWE secret $\mathbf{s}_i$.
  2. Run SwitchKeyGen($\mathbf{s}_i, t_i$) to generate packing key-switching keys $\mathbf{PKSK}_i = \{\mathbf{PKS}_{i,j}\}_{j\in[n]}$ and return.
  3. Run RLKeyGen(t_i) to generate the relinearization keys $\mathbf{PLK}_i = [\mathbf{f}_{i,0}|\mathbf{f}_{i,1}|\mathbf{f}_{i,2}]$.
  4. Run UniEnc($t_{i,j}, t_i$) to generate $\mathbf{F}_{i,j} = [\mathbf{f}_{i,j,0}|\mathbf{f}_{i,j,1}|\mathbf{f}_{i,j,2}]$ for $j \in [N]$, let $\mathbf{BK}_i = \{\mathbf{F}_{i,j}\}_{j\in[N]}$.
  5. Run SwitchKeyGen($\mathbf{t}_i^{'}, t_i$) to generate key-switching keys $\mathbf{KSK}_i = \{\mathbf{KS}_{i,j}\}_{j\in[N]}$.

  Public the quadruples $(\mathbf{PK}_i, \mathbf{PLK}_i, \mathbf{BK}_i, \mathbf{KSK}_i)$ of public keys, relinearization keys, bootstrapping keys, and key-switching keys.
- MKHE.Enc($m, \mathbf{s}_i, \mathbf{PKSK}_i$): Take a message $m \in \mathbb{Z}_q$, secret $\mathbf{s}_i$ and packing key-switching keys $\mathbf{PKSK}_i$. Run Enc($m, \mathbf{s}_i$) to generate LWE ciphertext $\mathbf{ct}_i^* \in \mathbb{Z}_q^{n+1}$, then run PKSwitch($\{\mathbf{ct}_i^*\}, \{0\}, \mathbf{PKSK}_i$) to pack LWE ciphertext into a RLWE ciphertext, generate RLWE ciphertext $\mathbf{ct}_i = (b_i, a_i) \in \mathcal{R}_q^2$.

In this paper, the ciphertext of the ith party is packed into the RLWE ciphertext, and the index id_i is 0 when there is only one LWE ciphertext. If each party has multiple LWE ciphertexts, the product of multiple LWE ciphertexts can be achieved according to the change id_i. See [20] for more details.

- MKHE.Dec($\overline{\mathbf{ct}}, t_1, \ldots, t_k$): Given a ciphertext $\overline{\mathbf{ct}} \in \mathcal{R}_q^{k+1}$ and a set of keys $t_1, \ldots, t_k$ of the associated partys, set key $\mathbf{t} = (1, t_1, .., t_k) \in \mathcal{R}_q^{k+1}$. Compute $\lfloor \frac{1}{\triangle} \lfloor \langle \overline{\mathbf{ct}}, \mathbf{t} \rangle \rceil_q \rceil$ to decrypt the RLWE ciphertext.

Next, the computation of the MKHE scheme will be described. Before this, the ciphertext needs to be preprocessed, and the RLWE ciphertexts of all parties are extended to the ciphertext under the concatenated key. By default, this paper preprocesses all the RLWE ciphertext before homomorphic calculation. Assuming that the number of parties is k, the extended ciphertext should satisfy the concatenated secret key $\mathbf{t} = (1, t_1, .., t_k) \in \mathcal{R}_q^{k+1}$. Rearrange and combine the input ciphertext $\mathbf{ct}_i = (b_i, a_{i,id_1}, \ldots, a_{i,id_{k_i}}) \in \mathcal{R}_q^{k_i+1}$, the associated index tuple is $(id_1, .., id_{k_i}) \in [k]^{k_i}$, where $k_i \leq k$. Then the extended ciphertext is $\overline{\mathbf{ct}}_i = \left(b_i, a_{i,id_1}^{'}, \ldots, a_{i,id_k}^{'}\right) \in \mathcal{R}_q^{k+1}$, padding empty slots with zero. So, $a_{i,j}^{'} = \begin{cases} a_{i,id_l} & \text{if } j = id_l \text{ for } l \in [k_i], \\ 0 & \text{otherwise;} \end{cases}$, for $j \in [k]$. We can conclude that $\langle \overline{\mathbf{ct}}_i, \mathbf{t} \rangle = \langle \mathbf{ct}_i, (1, t_{id_1}, .., t_{id_{k_i}}) \rangle$.

- MKHE.Add($\overline{\mathbf{ct}}_1, \overline{\mathbf{ct}}_2$): Given two RLWE ciphertexts $\overline{\mathbf{ct}}_1, \overline{\mathbf{ct}}_2 \in \mathcal{R}_q^{k+1}$, comput the ciphertext $\overline{\mathbf{ct}} = \overline{\mathbf{ct}}_1 + \overline{\mathbf{ct}}_2 \pmod q$ and return $\overline{\mathbf{ct}}$.
- MKHE.Mult($\overline{\mathbf{ct}}_1, \overline{\mathbf{ct}}_2, \{(\mathbf{RLK}_i, \mathbf{PK}_i)\}_{i\in[k]}$): Given two RLWE ciphertexts $\overline{\mathbf{ct}}_1, \overline{\mathbf{ct}}_2 \in \mathcal{R}_q^{k+1}$, relinearization keys and public keys $\{(\mathbf{RLK}_i, \mathbf{PK}_i)\}_{i\in[k]}$ by all parties involved. First calculate $\overline{\mathbf{ct}}^{'} = \lfloor \frac{\overline{\mathbf{ct}}_1 \otimes \overline{\mathbf{ct}}_2}{\triangle} \rceil_q$, then run $\overline{\mathbf{ct}} \leftarrow \text{ReLin}(\overline{\mathbf{ct}}^{'}, \{(\mathbf{RLK}_i, \mathbf{PK}_i)\}_{i\in[k]}$ and return $\overline{\mathbf{ct}}$.

After completing the homomorphic addition or multiplication of two ciphertexts, the noise of the ciphertext will grow rapidly. In the next step, we reference the method of [14] homomorphic accumulator to complete bootstrapping. That is, the decryption circuit of the extended LWE ciphertext is evaluated to realize the refresh of the noise. We obtain the ciphertext $\overline{\mathbf{ct}} \in \mathcal{R}_q^{k+1}$ after homomorphic calculation, and use the sample extract algorithm to convert RLWE ciphertext into LWE ciphertext. Run $SampleExtract_i(\overline{\mathbf{ct}})$. According to the index extracted from the packed ciphertext, the key of the LWE ciphertext obtained is the permutation and combination of the polynomial coefficients of the concatenated key $\mathbf{t}$. Finally, the LWE ciphertext $\overline{\mathbf{ct}}^{'} \in \mathbb{Z}_q^{kN+1}$ is returned.

- MKHE.BS($\overline{\mathbf{ct}}^{'}$, $\{(\mathbf{PK}_i, \mathbf{BK}_i, \mathbf{KSK}_i)\}_{i\in[k]}$, $P_{(f_1,\ldots,f_{2^\epsilon})}$, ϵ): Given a multi-key LWE ciphertext $\overline{\mathbf{ct}}^{'} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{Z}_q^{kN+1}$, group $\{(\mathbf{PK}_i, \mathbf{BK}_i, \mathbf{KSK}_i)\}_{i\in[k]}$ formed by public keys, bootstrapping keys and key-switching keys of the k parties, LUT functions $P_{(f_1,\ldots,f_{2^\epsilon})}$ and modulus switching parameters ϵ.
  1. Compute $b^{'} = \lfloor b \cdot \frac{2N}{q} \cdot 2^{-\epsilon} \rceil \cdot 2^{\epsilon} \pmod{2N}$, $\mathbf{a}_i^{'} = \lfloor \mathbf{a}_i \cdot \frac{2N}{q} \cdot 2^{-\epsilon} \rceil \cdot 2^{\epsilon} \pmod{2N}$ for $i \in [k]$, where $\mathbf{a}_i^{'} = (a_{i,1}^{'}, \ldots, a_{i,N}^{'}) \in \mathbb{Z}_{2N}^{N}$.
  2. According to the LUT function, $P_{(f_1,\ldots,f_{2^\epsilon})}$ generates a trivial RLWE ciphertext $\mathbf{ct} = (\triangle \cdot X^{-b^{'}} \cdot P_{(f_1,\ldots,f_{2^\epsilon})}, \mathbf{0}) \in \mathcal{R}_q^{k+1}$.
  3. Let $\mathbf{BK}_i = \{\mathbf{F}_{i,j} = [\mathbf{f}_{i,j,0}|\mathbf{f}_{i,j,1}|\mathbf{f}_{i,j,2}]\}_{j\in[N]}$. Given $i \in [k]$ and $j \in [N]$, recursive run generation $\overline{\mathbf{ct}}^{''} \leftarrow \mathrm{CMux}(\mathbf{ct}, \mathbf{ct} \cdot X^{-a_{i,j}^{'}}, \mathbf{F}_{i,j}, \{\mathbf{PK}_l\}_{l\in[k]})$.
  4. Given $i \in [2^\epsilon]$, run $SampleExtract_{i-1}(\overline{\mathbf{ct}}^{''})$ to iterative extraction and generate the LWE ciphertext $\mathbf{c}_i \in \mathbb{Z}_q^{kN+1}$.
  5. Let $\mathbf{KSK}_h = \{\mathbf{KS}_{h,l}\}_{l\in[N]}$ for $h \in [k]$, run $\overline{\mathbf{ct}}_i \leftarrow \mathrm{MKSwitch}(\mathbf{c}_i, \{\mathbf{KSK}_h\}_{h\in[k]})$ for $i \in [2^\epsilon]$. Return 2^ϵ RLWE ciphertexts $\{\overline{\mathbf{ct}}_i\}_{i\in[2^\epsilon]} \in (\mathcal{R}_q^{k+1})^{2^\epsilon}$ with respect to the concatenated keys.

Using ϵ bits, the least significant bits can be used as the index of the bootstrapping function; thus, the plaintext message space will be correspondingly reduced, set plaintext message space $p = \frac{q}{\triangle \cdot 2^{\epsilon+1}}$. LUT function $P_{(f_1,\ldots,f_{2^\epsilon})} \in \mathcal{R}_q$ is a polynomial composed of 2^ϵ functions, where $0 < 2^\epsilon < \triangle^{'}$ for scaling factor $\triangle^{'} = \frac{2N}{p}$. Set ordinary RLWE ciphertext $\mathbf{ct}$ as an accumulator and perform CMux gate operation on it. $\mathbf{F}_{i,j}$ is the uni-encryption for $t_{i,j} \in \mathbb{B}$ where $i \in [k]$ and $j \in [N]$. Using $\mathbf{F}_{i,j}$ as the selection parameter, $\mathbf{ct}$ or $\mathbf{ct} \cdot X^{a_{i,j}^{'}}$ can be homomorphically selected by the method of mixed product. Let plaintext space $m \in [0, p-1]$, $\mathbf{t}' = (1, \mathbf{t}_1^{'}, \ldots, \mathbf{t}_k^{'}) \in \mathbb{Z}_q^{kN+1}$. The calculation shows that $\langle \overline{\mathbf{ct}}^{''}, \mathbf{t}' \rangle \approx \triangle \cdot P_{(f_1,\ldots,f_{2^\epsilon})} \cdot X^{-b^{'} - \sum_{i=1}^{k} \langle \mathbf{a_i}, \mathbf{t}_i^{'} \rangle} \approx \triangle \cdot P_{(f_1,\ldots,f_{2^\epsilon})} \cdot X^{-\langle \overline{\mathbf{ct}}^{'}, \mathbf{t}' \rangle} \approx \triangle \cdot P_{(f_1,\ldots,f_{2^\epsilon})} \cdot X^{-\triangle^{'} \cdot m}$. Therefore, the LUT function is rotated by $\triangle^{'} \cdot m$, and the entries of coefficients $f_1(m), \ldots, f_{2^\epsilon}(m)$ are moved to the first 2^ϵ terms. Then the first 2^ϵ LWE ciphertexts are extracted and they are the result of homomorphic evaluation.

Finally, the ciphertext under the LWE key was replaced by the ciphertext under the RLWE key in the key-switching, waiting for the next homomorphic decryption or the multi-key ciphertext homomorphic calculation again. After bootstrapping, the ciphertext $\mathbf{ct} \in \mathbb{Z}_q^{kN+1}$ satisfies $\langle \mathbf{ct}, \mathbf{t}' \rangle \approx \triangle \cdot f(m) \pmod{q}$, the key is $\mathbf{t}' = (1, \mathbf{t}_1^{'}, \ldots, \mathbf{t}_k^{'}) \in \mathbb{Z}_q^{kN+1}$. Replace the LWE key with the RLWE key $(1, \mathbf{t}) = (1, t_1, \ldots, t_k) \in \mathcal{R}_q^{k+1}$ with the key-switching keys. Then the corresponding ciphertext satisfies $\langle \overline{\mathbf{ct}}, (1, \mathbf{t}) \rangle \approx \triangle \cdot f(m) \pmod{q}$.

Security. This paper uses uni-encryption to generate bootstrapping keys and key-switching keys. It has been indicated in Section 3.2 that the keys meet semantic security standards. In order to enable the homomorphic encryption system to still have enough space for homomorphic computation (homomorphic addition or homomorphic multiplication) after a homomorphic computation of the decryption function, and to achieve any depth of homomorphic computation, we need to use the bootstrapping keys cyclically. So, as with many bootstrapping homomorphic encryption schemes, we propose an additional circular security assumption. Because the generation of these keys meets the semantic security, it is difficult to distinguish these ciphertexts from other ciphertexts. Consequently, we believe that the circular security hypothesis is secure.

4.2. Distributed Decryption

In an ideal homomorphic encryption scheme, each party only has its own key and does not know the keys of other parties. However, when decrypting a multi-key ciphertext, all the keys of the parties are needed, so it is not practical to complete the decryption

without revealing the keys of all parties. In practical applications, such as MPC schemes, efficient protocols can be designed for joint decryption. This paper cites [24] to implement a simple distributed decryption based on noise flooding technology. Specific parameters and security can be referred to in this paper. A noise distribution φ with variance larger than the standard error distribution ψ of the basic scheme is first set, and a noise is added to the calculation of each party. Distributed decryption is roughly divided into two parts. The first part sends the items in RLWE ciphertext $\overline{\mathbf{ct}} = (b, a_1, \ldots, a_k) \in \mathcal{R}_q^{k+1}$ except for those with plaintext messages to the corresponding participants, and each participant partially decrypts them and sends them out again. The second part is to connect the partially decrypted messages. The specific structure is as follows:

- MKHE.PartDec(a_i, t_i): Given the $(1+i)$ term $a_i \in \mathcal{R}_q$ of the RLWE ciphertext that needs to be decrypted, as well as the RLWE key $t_i \in \mathcal{R}_q$ of the ith party, sample an error $e_i \leftarrow \varphi$. Generate message $m_i = a_i \cdot t_i + e_i \pmod q$ and return.
- MKHE.Merge($b, \{\mu_i\}_{i\in[k]}$): Provide the first item $b \in \mathcal{R}_q$ of RLWE ciphertext, $\{\mu_i\}_{i\in[k]}$ are the partially decrypted messages from all participants. Calculate $\overline{\mu} = b + \sum_{i=1}^{k} \mu_i \cdot t_i \pmod q$. Return $m = \lfloor \overline{\mu} / \triangle \rceil_q$.

5. Analysis

5.1. Security

As mentioned above, the basic encryption method in this article is based on the LWE and RLWE assumptions, using the uni-encryption method to generate relinearization keys and bootstrapping keys to complete the calculation of multi-key ciphertext. Consequently, the selected LWE parameters and RLWE parameters should meet the security level of at least λ bits. The basic principle is to add the encoded plaintext to a random encryption of zero to generate ciphertext, and then we can perform homomorphic evaluation on this ciphertext. The security of encryption generated key-switching keys, relinearization keys, and bootstrap keys methods are evaluated in Section 3.1.

- LWE problem. The LWE parameters (n, χ, q, α) are obtained according to the parameter generation, secret $\mathbf{s} = (s_1, \ldots, s_n) \leftarrow \chi^n$. Let $\mathcal{D}_\alpha$ as an error distribution over $\mathbb{Z}_q$. The decisional learning with errors problem is to distinguish distributions $\mathcal{D}_0$ and $\mathcal{D}_1$, among them $\mathcal{D}_0 = \{(b, \mathbf{a}) : \mathbf{a} = (a_1, \ldots, a_n) \leftarrow \mathrm{U}(\mathbb{Z}_q^n),\ e \leftarrow \mathcal{D}_\alpha, b = -\sum_{i=1}^{n} a_i \cdot s_i + e \pmod q\}$, $\mathcal{D}_1 = \{(b, \mathbf{a}) : \mathbf{a} \leftarrow \mathrm{U}(\mathbb{Z}_q^n), b \leftarrow \mathrm{U}(\mathbb{Z}_q)\}$.
- RLWE problem. The RLWE parameters (N, ψ, α, q) are obtained according to the parameter generation, secret $t \leftarrow \psi$. Let ω_α as an error distribution over $\mathcal{R}_q$. The decisional ring learning with errors problem is to distinguish distributions $\mathcal{D}_0$ and $\mathcal{D}_1$, among them $\mathcal{D}_0 = \{(b, a) : a \leftarrow \mathrm{U}(\mathcal{R}_q),\ e \leftarrow \omega_\alpha,\ b = -a \cdot t + e \pmod q\}$, $\mathcal{D}_1 = \{(b, a) : a \leftarrow \mathrm{U}(\mathcal{R}_q), b \leftarrow \mathrm{U}(\mathcal{R}_q)$.

We base security on the LWE assumption. Firstly, if the adversary can distinguish between LWE encrypted vectors and uniform random vectors on $\mathbb{Z}_q^{n+1}$, then the adversary can solve the LWE problem. However, when the security level is at least λ bits, the LWE problem is hard, so the adversary cannot distinguish effectively. Secondly, if the adversary can effectively select the LWE encryption vector on $\mathbb{Z}_q^{n+1}$, but ciphertext generated by this encryption is independent of the plaintext message, making it difficult for the adversary to find plaintext messages from the ciphertext. The same holds for the RLWE assumption, and the adversary cannot solve the RLWE problem efficiently.

5.2. Noise Analysis

In Section 2.3 we introduced the base B and degree d of the gadget decomposition tool, and we know that the decomposition vectors are uniformly distributed over the interval $(-\frac{1}{B}, \frac{1}{B}] \cap \mathbb{Z}$. Set the variance to $\mathfrak{B} = \begin{cases} 1/12 \cdot \left(B^2 - 1\right) \text{ for } B \text{ is odd,} \\ 1/12 \cdot \left(B^2 + 2\right) \text{ for } B \text{ is even;} \end{cases}$. Applying the decomposition will produce errors that are uniformly distributed in the interval $(-\frac{1}{2 \cdot B^d}, \frac{1}{2 \cdot B^d}]$, and we set the variance to $\xi^2 = \frac{1}{12 \cdot B^{2d}}$. Gadget decomposition tools are used in the key

switching and uni-encryption of the proposed scheme. We assume that the coefficients of the polynomial have the same independent random distribution, and noise estimates are provided next.

LWE encryption. To encrypt a message $m \in \mathbb{Z}_p$, obtain ciphertext $\mathbf{ct} = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$ where samples $\mathbf{a} \leftarrow \mathrm{U}(\mathbb{Z}_q^n)$, $e \leftarrow \mathcal{D}_\alpha$ and $b = -\langle \mathbf{a}, \mathbf{s} \rangle + e + \triangle \cdot m \pmod q$. Calculating phase $\varphi_{\mathbf{s}}(\mathbf{ct}) = e + \triangle \cdot m \pmod q$. Hence, the noise of LWE encryption $e_{LWEenc} = e$. The variance is $V_{LWEenc} = \alpha^2$.

LWE ciphertext packing. According to Section 3.1, RLWE ciphertext $\mathbf{ct} = (b, a) \in \mathcal{R}_q^2$ is generated by packing ciphertext. The calculating phase is

$$\begin{aligned}
\varphi_t(\mathbf{ct}) =& \sum_{i=1}^{p} b_i \cdot X^{id_i} + \sum_{i=1}^{p} \sum_{j=1}^{n} \langle \mathbf{g}^{-1}(a_{i,j}) \cdot \mathbf{KS}_j, (1, t) \rangle \cdot X^{id_i} \\
=& \sum_{i=1}^{p} (b_i + \sum_{j=1}^{n} \langle \mathbf{g}^{-1}(a_{i,j}) \cdot (-\mathbf{A}_j t + s_j \cdot \mathbf{g} + \mathbf{e}_j, \mathbf{A}_j), (1, t) \rangle) \cdot X^{id_i} \\
=& \sum_{i=1}^{p} (b_i + \sum_{j=1}^{n} (a_{i,j} \cdot s_j + \mathbf{g}^{-1}(a_{i,j}) \cdot \mathbf{e}_j)) \cdot X^{id_i} \\
=& \sum_{i=1}^{p} (e_i + \triangle \cdot m_i + \sum_{j=1}^{n} (e_{i,j}^{'} \cdot s_j + \mathbf{g}^{-1}(a_{i,j}) \cdot \mathbf{e}_j)) \cdot X^{id_i} \pmod q.
\end{aligned}$$

Then, $\sum_{i=1}^{p} e_i X^{id_i}$ is the noise-generated polynomial of p LWE ciphertext, $\sum_{i=1}^{p} \sum_{j=1}^{n} (e_{i,j}^{'} \cdot s_j + \mathbf{g}^{-1}(a_{i,j}) \cdot \mathbf{e}_j) \cdot X^{id_i}$ is the noise added by the packed ciphertext. $e_{i,j}^{'} = \langle \mathbf{g}^{-1}(a_{i,j}), \mathbf{g} \rangle - a_{i,j}$ is the noise created by decomposition; it is concluded that packaging ciphertext $\sum_{i=1}^{p} m_i X^{id_i}$ noise variance for $V_{pk} = \alpha^2 + n(\frac{1}{2}\xi^2 + \mathfrak{B} d \alpha^2)$.

Relinearization. According to Section 3.2, the multi-key ciphertext $\overline{\mathbf{ct}}^{'} = (ct_0^{'}, ct_1^{'}, \ldots, ct_k^{'}) \in \mathcal{R}_q^{k+1}$ is generated by relinearizing the ciphertext $\overline{\mathbf{ct}} = (ct_{i,j})_{0 \le i,j \le k} \in \mathcal{R}_q^{(k+1)\times(k+1)}$. In each iteration of uni-encryption, $\langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,0} \rangle + \langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{f}_{i,1} \rangle \cdot t_i = v_{i,j} \cdot r_i + e_{i,j}^{'} + \mathbf{g}^{-1}(v_{i,j}) \cdot \mathbf{e}_{i,0} \pmod q$, $\langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{f}_{i,2} \rangle \cdot t_j = -v_{i,j} \cdot r_i + \langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{e}_j \cdot r_i + \mathbf{e}_{i,2} \cdot t_j \rangle + (ct_{i,j} + e_{i,j}^{''}) \cdot t_i \cdot t_j \pmod q$. Where $e_{i,j}^{'} = \langle \mathbf{g}^{-1}(v_{i,j}), \mathbf{g} \rangle - v_{i,j}$ and $e_{i,j}^{''} = \langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{g} \rangle - ct_{i,j}$ denote the noise generated by the decomposition. The phase $\varphi_{\mathbf{t}}(\overline{\mathbf{ct}}^{'}) = \varphi_{\mathbf{t} \otimes \mathbf{t}}(\overline{\mathbf{ct}}) + \sum_{i=1}^{k} \sum_{j=1}^{k} (e_{i,j}^{'} + \mathbf{g}^{-1}(v_{i,j}) \cdot \mathbf{e}_{i,0} + \langle \mathbf{g}^{-1}(ct_{i,j}), \mathbf{e}_j \cdot r_i + \mathbf{e}_{i,2} \cdot t_j \rangle + e_{i,j}^{''} \cdot t_i \cdot t_j) \pmod q$ is calculated by summing the two equations. To obtain noise variance, $V_{relin} = k^2(\xi^2 + (N + N^2)\mathfrak{B} \cdot d \cdot \alpha^2 + \frac{N^2}{4}\xi^2) \approx k^2 N^2 (\mathfrak{B} d \alpha^2 + \xi^2)$.

Multiplication. Given two RLWE ciphertexts $\overline{\mathbf{ct}}_1, \overline{\mathbf{ct}}_2 \in \mathcal{R}_q^{k+1}$, calculate $\langle \overline{\mathbf{ct}}_i, \mathbf{t}^{'} \rangle = \triangle \cdot m_i + e_i + q \cdot H_i$ for $i \in \{1, 2\}$, where $H_i = \lfloor \frac{\langle \overline{\mathbf{ct}}_i, \mathbf{t}^{'} \rangle}{q} \rceil$. So, the variance $var(H_i) = \frac{1}{q^2} \cdot \frac{q^2 - 1}{12} \cdot (1 + \frac{1}{2} \cdot k \cdot N) \approx \frac{kN}{24}$. Next, calculate the ciphertext tensor product $\langle \overline{\mathbf{ct}}_1, \mathbf{t}^{'} \rangle \cdot \langle \overline{\mathbf{ct}}_2, \mathbf{t}^{'} \rangle = (\triangle \cdot m_1 + e_1 + q \cdot H_1) \cdot (\triangle \cdot m_2 + e_2 + q \cdot H_2) = \triangle \cdot (\triangle \cdot m_1 m_2 + m_1 e_2 + m_2 e_1) + e_1 e_2 + q \cdot (\triangle \cdot m_1 H_2 + e_1 H_2 + \triangle \cdot m_2 H_1 + e_2 H_1 + q \cdot H_1 H_2) \pmod{\triangle \cdot q}$ before taking the module. Then, divide it by $\triangle$ and round it to get the phase $\varphi_{\mathbf{t}'}(\overline{\mathbf{ct}}_1 \otimes \overline{\mathbf{ct}}_2) = \triangle \cdot m_1 m_2 + m_1 e_2 + m_2 e_1 + \frac{e_1 e_2}{\triangle} + q(m_1 H_2 + m_2 H_1 + \frac{q}{\triangle} \cdot H_1 H_2) + \frac{q}{\triangle}(e_1 H_2 + e_2 H_1) + e_r \pmod q$ of tensor product, where $q(m_1 H_2 + m_2 H_1 + \frac{q}{\triangle} \cdot H_1 H_2)$ overlaps the module q, e_r is the rounding error. Then, the total noise term is $e_{multi} = m_1 e_2 + m_2 e_1 + \frac{e_1 e_2}{\triangle} + + \frac{q}{\triangle}(e_1 H_2 + e_2 H_1) + e_r$. Compute the major term $\frac{q}{\triangle}(e_1 H_2 + e_2 H_1)$ and obtain the noise variance $V_{multi} \approx \frac{kN^2 q^2}{24 \cdot \triangle^2}(\alpha_1^2 + \alpha_2^2)$.

Bootstrapping. As described in Section 4.1, suppose that RLWE ciphertext $\mathbf{ct} = (ct_0, ct_1, \ldots, ct_k) \in \mathcal{R}_q^{k+1}$ encapsulates 2^ϵ plaintext function values, which are noise free. To refresh the noise of LWE ciphertext $\overline{\mathbf{ct}}^{'} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{Z}_q^{kN+1}$, it is necessary to calculate kN times hybrid product.

1. Hybrid product. The bootstrapped ciphertext is a uni-encrypted set of LWE keys $t \in \mathbb{B}$, and the phase $\varphi_{\mathbf{t}}(\mathbf{ct}) = \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(ct_j), \mathbf{f}_2 \rangle \cdot t_j + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_0 \rangle + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_1 \rangle \cdot t_i \pmod q$ is computed. Among them $\sum_{j=0}^{k} \langle \mathbf{g}^{-1}(ct_j), \mathbf{f}_2 \rangle \cdot t_j = \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(ct_j), (r \cdot \mathbf{a} + t \cdot \mathbf{g} + \mathbf{e}_2) \cdot t_j \rangle = \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(ct_j), (r \cdot (-\mathbf{b} + \mathbf{e}) + t \cdot \mathbf{g} \cdot t_j + \mathbf{e}_2 \cdot t_j) \rangle = t \cdot \sum_{j=0}^{k} ct_j \cdot t_j - \sum_{j=0}^{k} v_j \cdot r + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(ct_j), (r \cdot \mathbf{e} + t \cdot e^{'} \cdot t_j + \mathbf{e}_2 \cdot t_j) \rangle \pmod q$, and $\sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_0 \rangle + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_1 \rangle \cdot t_i = \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), -t_i \cdot \mathbf{f}_1 + r \cdot \mathbf{g} + \mathbf{e}_1 \rangle + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_1 \rangle \cdot t_i = \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), r \cdot \mathbf{g} + \mathbf{e}_1 \rangle = \sum_{j=0}^{k} v_j \cdot r + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), (r \cdot e'' + \mathbf{e}_1) \rangle \pmod q$. Hence, $\varphi_{\mathbf{t}}(\mathbf{ct}) = t \cdot \sum_{j=0}^{k} ct_j \cdot t_j + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(ct_j), (r \cdot \mathbf{e} + t \cdot e^{'} \cdot t_j + \mathbf{e}_2 \cdot t_j) \rangle + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), (r \cdot e'' + \mathbf{e}_1) \rangle \pmod q$. To obtain the calculated noise, we use the following equation: $e_{hybrid} = \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(ct_j), (r \cdot \mathbf{e} + \mathbf{e}_2 \cdot t_j) \rangle + \sum_{j=0}^{k} (t \cdot e^{'} \cdot t_j + r \cdot e'') + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), \mathbf{e}_1 \rangle$. e' and e'' are noises generated by the decomposition. So, it is necessary to complete the noise variance $V_{hybrid} \approx kN(N\mathfrak{B}d\alpha^2 + \xi^2)$ of the hybrid product once.
2. CMux gate. Since the secret keys are sampled from a uniform binary distribution, $\mathbf{ct} = \mathbf{ct} \cdot X^{-a^{'}_{i,j} \cdot t_{i,j}} = \mathbf{ct} + t_{i,j} \cdot (\mathbf{ct} \cdot X^{-a^{'}_{i,j}} - \mathbf{ct})$ are computed iteratively for $i \in [k],\ j \in [N]$. Then, the uni-encrypted ciphertext of $t_{i,j}$ is used for homomorphic computation. Let $\mathbf{c} = \mathbf{ct} \cdot X^{-a^{'}_{i,j}} - \mathbf{ct} = (c_0, c_1, \ldots, c_k) \in \mathcal{R}_q^{k+1}$, calculate the hybrid product once to get the phase $\varphi_{\mathbf{t}}(\mathbf{ct}) = \sum_{j=0}^{k} ct_j \cdot t_j + t_{i,j} \cdot \sum_{j=0}^{k} c_j \cdot t_j + e_{hybrid}$. $\mathbf{c}$ is regarded as the result of the homomorphic addition of two RLWE keys. Since the initial RLWE ciphertext is noise-free, the amount of noise increase after each run of the CMux gate can be considered as e_{hybrid}. Run the CMux gate kN times in a bootstrapping operation; then, the final noise variance is $V_{CMux} \approx 3kNV_{hybrid} \approx 3k^2N^2(N\mathfrak{B}d\alpha^2 + \xi^2)$.
3. Key Switching. The above computation is then extracted to generate LWE ciphertexts. Let one of them be $\mathbf{c} \in \mathbb{Z}_q^{kN+1}$, and the key switching converts it into RLWE ciphertexts $\mathbf{c}' \in \mathcal{R}_q^{k+1}$. So there's

$$\begin{aligned}
\langle \mathbf{c}', (1, t) \rangle =& b + \sum_{i=1}^{k} b_i^{'} + \sum_{i=1}^{k} \mathbf{a}_i^{'} t_i = b + \sum_{i=1}^{k} \sum_{j=1}^{N} \mathbf{g}^{-1}(a_{i,j}) \cdot \mathbf{KS}_{i,j} \cdot (1, t_i) \\
=& b + \sum_{i=1}^{k} \sum_{j=1}^{N} \langle \mathbf{g}^{-1}(a_{i,j}), (-\mathbf{A}_{i,j} \cdot t_i + s_{i,j} \cdot \mathbf{g} + \mathbf{e}_{i,j}, \mathbf{A}_{i,j} \cdot t_i) \rangle \\
=& b + \sum_{i=1}^{k} \sum_{j=1}^{N} \langle \mathbf{g}^{-1}(a_{i,j}), (s_{i,j} \cdot \mathbf{g} + \mathbf{e}_{i,j}) \rangle \\
=& b + \sum_{i=1}^{k} \sum_{j=1}^{N} a_{i,j} \cdot s_{i,j} + \sum_{i=1}^{k} \sum_{j=1}^{N} (e_{i,j}^{'} \cdot s_{i,j} + \langle \mathbf{g}^{-1}(a_{i,j}), \mathbf{e}_{i,j} \rangle) \pmod q,
\end{aligned}$$

where $e_{i,j}^{'}$ is the noise generated by the decomposition. Then, the noise variance of the key exchange is $V_{swith} = kN(\frac{1}{2}\xi^2 + N\mathfrak{B}d\alpha^2)$.

The noise variance of the final bootstrapping operation is approximately $V_{boot} = V_{CMux} + V_{swith} \approx 3k^2N^2(N\mathfrak{B}d\alpha^2 + \xi^2)$.

5.3. Performance Analysis

The MKHE scheme proposed in this paper is an extension of the multi-key TFHE scheme. The main purpose is to extend the functionality of bootstrap in the multi-key homomorphic encryption scheme, that is, to evaluate multiple functions homomorphically under a single refresh noise. At the same time, as much as possible, the computational efficiency is not degraded. Therefore, this paper mainly compares with the scheme [14]. Table 2 will list the comparison results.

Table 2. Comparison of main parameters of multi-key TFHE scheme. Where k is the number of parties, n is the dimension assumed by (R)LWE, ϵ is the number of output function bits and $\epsilon \geq 0$.

Scheme	Ciphertext Space Complexity	Homomorphic Evaluation Time Complexity	Bootstrapping Time Complexity	The Number of PBS Outputs
CCS19 [14]	$\tilde{O}(kn)$	$\tilde{O}(k^2n^2)$	$\tilde{O}(k^2n^2)$	1
Ours	$\tilde{O}(kn)$	$\tilde{O}(k^2n)$	$\tilde{O}(k^2n^2)$	2^ϵ

From the table comparison analysis, we can see that the space and time performance of bootstrapping in this paper is consistent with the scheme of CCS19. Since each computation of the NAND gate of CCS19 requires a bootstrap, their homomorphic operation time complexity coincides with the bootstrap time complexity. In this paper, the homomorphic operation is separated from bootstrapping, so bootstrapping is not required after each homomorphic operation.The plaintext space of the CCS19 scheme is binary, and the utilization of the test polynomial encoded by LUT in bootstrapping is only $\frac{2}{n}$. The plaintext space of this paper is p, which is no longer restricted to binary values. According to the reasonable setting of parameter ϵ, the proposed scheme can evaluate 2^ϵ functions at the same time in a bootstrap operation without adding additional noise, and the utilization of the LUT-encoded test polynomial is improved to $\frac{p \cdot 2^\epsilon}{n}$. Therefore, in the scenario of multi-party joint computation, when homomorphically computing the same number of functions, the proposed scheme can be implemented with fewer bootstrapping operations, which will effectively reduce the computational overhead.

6. Discussion

This scheme is a multi-key variant of the TFHE scheme, based on [20]. The selection of parameters can be referred to in this paper. This scheme is a basic multi-key scheme, which realizes the packing of ciphertext, homomorphic addition, tensor product and bootstrapping method of multi-output functions. We prove that the homomorphic calculation and bootstrapping method of ciphertext are effective, so it is easy to expand more homomorphic encryption functions and optimize based on this scheme.

A bootstrapping scheme without padding. Scheme [20] implements a high-precision homomorphic encryption scheme. Their bootstrapping method allows the most significant bit of the ciphertext to be non-zero, thereby increasing the plaintext message space and enabling the encryption of larger plaintexts. The basic function of this scheme is based on [20], so it is easy to expand the unfilled multi-key bootstrapping method in our scheme. The disadvantage is that their no-fill scheme requires more bootstrapping operations, while the bootstrapping operation in the homomorphic encryption scheme is quite expensive, thus greatly increasing the time and computational complexity. One improvement direction is to increase the number of bits in the plaintext space while simultaneously reducing computational complexity.

Faster evaluation LUT. This scheme applies LUT in the bootstrapping module. Two methods of packaging and calculating the LUT in the TFHE bootstrapping scheme are implemented in paper [15]. The ciphertext construction in this paper is in line with the TFHE scheme. Therefore, their LUT packing technique can be referred to in order to achieve a faster bootstrapping calculation of our scheme.

A faster MKHE scheme. The multi-key ciphertext homomorphic computation and bootstrapping method of this scheme are based on [14,22]. The hybrid product of [14] and the relinearization algorithm of [22] were optimized in [25,26], respectively, to improve the computational speed. It is easy to see that our scheme can also use their optimization schemes to improve the computation speed of the ciphertext multiplication and bootstrapping part of the homomorphic encryption.

7. Conclusions

The homomorphic encryption scheme can calculate the ciphertext. So, it can effectively reduce the risk of data leakage of the data holder in the cloud computing environment. However, most of the current homomorphic encryption schemes are designed for a single key. In practical scenarios, many outsourced computing requires homomorphic operations of data provided by different owners. Therefore, it is not limited to encrypting messages with a single key. In the existing MKHE schemes which support PBS, there are problems that a ciphertext can only encrypt binary message and only one function can be output at a time with PBS. In this paper, we specifically describe the MKHE scheme that supports multi-output PBS, so that multi-key ciphertexts can store high-precision plaintext messages. The tensor product and its relinearization are added to the computation of the multi-key ciphertexts. We separate the homomorphic operation from bootstrapping and implement a fast bootstrapping function similar to TFHE. At the same time, multiple functions can be homomorphically evaluated in a bootstrapping calculation, which enables faster homomorphic computation when there are multiple computing requirements. Our scheme also supports the packing technique of ciphertexts. Finally, we present the performance analysis. The results show that the scheme has better application scenarios.

In the discussion section, we discuss some improvement directions of this scheme. In addition, we can also consider how to homomorphically evaluate multiple functions at the bootstrapping time without reducing the plaintext space. Based on the concepts presented in this paper, we propose a CRS-free multi-key homomorphic encryption scheme. This scheme can be effectively applied in multi-party computation (MPC) or neural networks for enhanced privacy and security.

Author Contributions: Conceptualization, L.L. and R.H.; methodology, L.L.; validation, L.L. and R.H.; formal analysis, L.L.; writing—original draft, L.L.; funding acquisition, R.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation Project of China under Grant No. 62062009 and the Guangxi Innovation-driven Development Project under Grant Nos. AA17204058-17 and AA18118047-7.

Data Availability Statement: Not applicable.

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

References

1. Gentry, C. Fully Homomorphic Encryption Using Ideal Lattices. In Proceedings of the 41st ACM Symposium on Theory of Computing, STOC '09, New York, NY, USA, 31 May–2 June 2009; pp. 169–178. [CrossRef]
2. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory* **2014**, *6*, 1–36. [CrossRef]
3. Brakerski, Z. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In Proceedings of the Advances in Cryptology—CRYPTO 2012, Santa Barbara, CA, USA, 19–23 August 2012; Safavi-Naini, R., Canetti, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 868–886. [CrossRef]
4. Fan, J.; Vercauteren, F. Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive. Paper 2012/144. 2012. Available online: https://eprint.iacr.org/2012/144 (accessed on 10 May 2023).
5. Gentry, C.; Sahai, A.; Waters, B. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Proceedings of the Advances in Cryptology—CRYPTO 2013, Santa Barbara, CA, USA, 18–22 August 2013; Canetti, R., Garay, J.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 75–92. [CrossRef]
6. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In Proceedings of the Advances in Cryptology—ASIACRYPT 2016, Hanoi, Vietnam, 4–8 December 2016; Cheon, J.H., Takagi, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 3–33. [CrossRef]
7. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. In Proceedings of the Advances in Cryptology—ASIACRYPT 2017, Hong Kong, China, 3–7 December 2017; Takagi, T., Peyrin, T., Eds.; Springer: Cham, Switzerland, 2017; pp. 409–437. [CrossRef]
8. 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, STOC '12, New York, NY, USA, 19–22 May 2012; pp. 1219–1234. [CrossRef]

9. Clear, M.; McGoldrick, C. Multi-identity and Multi-key Leveled FHE from Learning with Errors. In Proceedings of the Advances in Cryptology—CRYPTO 2015, Santa Barbara, CA, USA, 16–20 August 2015; Gennaro, R., Robshaw, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 630–656. [CrossRef]
10. Mukherjee, P.; Wichs, D. Two Round Multiparty Computation via Multi-key FHE. In Proceedings of the Advances in Cryptology—EUROCRYPT 2016, Vienna, Austria, 8–12 May 2016; Fischlin, M., Coron, J.S., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 735–763. [CrossRef]
11. Peikert, C.; Shiehian, S. Multi-key FHE from LWE, Revisited. In Proceedings of the Theory of Cryptography, Tel Aviv, Israel, 10–13 January 2016; Hirt, M., Smith, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 217–238. [CrossRef]
12. Brakerski, Z.; Perlman, R. Lattice-Based Fully Dynamic Multi-key FHE with Short Ciphertexts. In Proceedings of the Advances in Cryptology—CRYPTO 2016, Santa Barbara, CA, USA, 14–18 August 2016; Robshaw, M., Katz, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 190–213. [CrossRef]
13. Chen, L.; Zhang, Z.; Wang, X. Batched Multi-hop Multi-key FHE from Ring-LWE with Compact Ciphertext Extension. In Proceedings of the Theory of Cryptography, Baltimore, MD, USA, 12–15 November 2017; Kalai, Y., Reyzin, L., Eds.; Springer: Cham, Switzerland, 2017; pp. 597–627. [CrossRef]
14. Chen, H.; Chillotti, I.; Song, Y. Multi-Key Homomorphic Encryption from TFHE. In Proceedings of the Advances in Cryptology—ASIACRYPT 2019, Kobe, Japan, 8–12 December 2019; Galbraith, S.D., Moriai, S., Eds.; Springer: Cham, Switzerland, 2019; pp. 446–472. [CrossRef]
15. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. In Proceedings of the Advances in Cryptology—ASIACRYPT 2017, Hong Kong, China, 3–7 December 2017; Takagi, T., Peyrin, T., Eds.; Springer: Cham, Switzerland, 2017; pp. 377–408. [CrossRef]
16. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptol.* **2020**, *33*, 34–91. [CrossRef]
17. Regev, O. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM* **2009**, *56*. [CrossRef]
18. Lyubashevsky, V.; Peikert, C.; Regev, O. On Ideal Lattices and Learning with Errors over Rings. *J. ACM* **2013**, *60*, 1–35. [CrossRef]
19. Chillotti, I.; Joye, M.; Ligier, D.; Orfila, J.B.; Tap, S. CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE. In Proceedings of the WAHC 2020—8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual, 15 December 2020. Available online: https://inria.hal.science/hal-03926650 (accessed on 20 April 2023)
20. Chillotti, I.; Ligier, D.; Orfila, J.B.; Tap, S. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In Proceedings of the Advances in Cryptology—ASIACRYPT 2021, Singapore, 6–10 December 2021; Tibouchi, M., Wang, H., Eds.; Springer: Cham, Switzerland, 2021; pp. 670–699. [CrossRef]
21. Li, N.; Zhou, T.; Yang, X.; Han, Y.; Liu, W.; Tu, G. Efficient Multi-Key FHE With Short Extended Ciphertexts and Directed Decryption Protocol. *IEEE Access* **2019**, *7*, 56724–56732. [CrossRef]
22. 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, CCS '19, New York, NY, USA, 11–15 November 2019; pp. 395–412. [CrossRef]
23. Brakerski, Z.; Vaikuntanathan, V. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In Proceedings of the Advances in Cryptology—CRYPTO 2011, Santa Barbara, CA, USA, 14–18 August 2011; Rogaway, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 505–524. [CrossRef]
24. Asharov, G.; Jain, A.; López-Alt, A.; Tromer, E.; Vaikuntanathan, V.; Wichs, D. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In Proceedings of the Advances in Cryptology—EUROCRYPT 2012, Cambridge, UK, 15–19 April 2012; Pointcheval, D., Johansson, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 483–501. [CrossRef]
25. Kwak, H.; Min, S.; Song, Y. Towards Practical Multi-Key TFHE: Parallelizable, Key-Compatible, Quasi-Linear Complexity. IACR Cryptology ePrint Archive, Paper 2022/1460. 2022. Available online: https://eprint.iacr.org/2022/1460 (accessed on 12 March 2023).
26. Kim, T.; Kwak, H.; Lee, D.; Seo, J.; Song, Y. Asymptotically Faster Multi-Key Homomorphic Encryption from Homomorphic Gadget Decomposition. IACR Cryptology ePrint Archive, Paper 2022/347. 2022. Available online: https://eprint.iacr.org/2022/347 (accessed on 15 March 2023).