# Efficient RLWE-Based Multi-key Fully Homomorphic Encryption Without Key-Switching

Xiaoliang Che[1,2], Yao Liu[2], Shangwen Zheng[2], Longfei Liu[1,2], Tanping Zhou[1(✉)], Xiaoyuan Yang[1,2], and Xu An Wang[1]

[1] College of Cryptographic Engineering, Engineering University of PAP, Xi'an 710086, Shaanxi, China

[2] Key Laboratory of Network and Information Security of the PAP, Xi'an 710086, Shaanxi, China

**Abstract.** The previous leveled BGV-type MKFHE schemes (e.g. CZW17, LZY+19) based on the standard RLWE assumption are implemented by using key-switching and modulus-switching techniques. However, the frequent usage of key-switching causes the low efficiency of homomorphic multiplication operation. The CDKS19 scheme proposed two new simpler and faster relinearization algorithms, which supported the homomorphic computation with certain circuit depth. However, the construction that satisfies the fully homomorphic computation was not designed, and its relinearization performance can be further optimized.

In this paper, a more efficient leveled BGV-type MKFHE scheme without key-switching is constructed. Firstly, the generation method of evaluation key is improved, and two optimized generation algorithms of relinearization key are proposed. Secondly, following the relinearization algorithm framework of CDKS19, two efficient relinearization algorithms are proposed. The new algorithms are much faster to re-linearize the product of ciphertexts. Finally, using the optimized relinearization algorithms to replace the key-switching technology logically, and combining the modulus-switching technology, an efficient leveled MKFHE is constructed.

The results show that our MKFHE scheme is IND-CPA secure based on the standard RLWE assumption, and supports any parties dynamically join the homomorphic computation at any time. Moreover, the time complexity of relinearization and decryption is less than that of CDKS19. So it is a leveled BGV-type MKFHE scheme with more efficient homomorphic computation.

**Keywords:** Multi-key fully homomorphic encryption · Key-switching technique · Relinearization algorithm · The time complexity

## 1 Introduction

The typical FHE schemes can only support homomorphic computation of ciphertext for a single party, that is, all ciphertexts participating in computation correspond to the


J. Xiong et al. (Eds.): MobiMedia 2021, LNICST 394, pp. 323–342, 2021.
https://doi.org/10.1007/978-3-030-89814-4_24

one secret key. However, in many scenarios, it is usually necessary to calculate the data uploaded to the cloud by multi parties in the network. In order to meet this practical demand, multi-key fully homomorphic encryption (MKFHE) [1] has been proposed. It can realize the effective integration of multi-party data under the condition of protecting data security, which has important research meaning and high application value. One of the most appealing applications of MKFHE is to construct on-the-fly multiparty computation (MPC) protocols [2, 3].

### 1.1 Background

Multi-key fully homomorphic encryption can solve the problem of homomorphic operation between ciphertexts of different parties, and the results of the operation can be jointly decrypted by the secret keys. MKFHE can realize the multi-party data security calculation, which is mainly divided into four types: NTRU-type MKFHE, GSW-type MKFHE, TFHE-type MKFHE, and BGV-type MKFHE.

In 2012, López-Alt et al. first proposed the NTRU-type MKFHE based on the NTRU cryptosystem [4], which was optimized later in DHS16 [5]. In PKC2017, Chongchitmate et al. gave a basic framework for constructing MKFHE with circuit privacy characteristics and proposed an MKFHE scheme CO17 [6] that can protect the circuit privacy. However, the security of NTRU-type MKFHE is based on a new and somewhat non-standard assumption on RLWE. Based on the prime cyclotomic polynomial ring, Che et al. constructed an efficient NTRU-type MKFHE scheme CZL$^+$20 [7] by using the ciphertext dimension extension technology, which eliminates the key-switching process in the relinearization and effectively reduces the key size. But the ciphertext size of the scheme increases.

In CRYPTO2015, Clear and McGoldrick proposed the first GSW-type MKFHE scheme CM15 based on LWE problem [8], which proposes a transformation model from FHE to MKFHE. This transformation model is widely adopted by most MKFHE schemes based on LWE or RLWE problems. In EUROCRYPT 2016, Mukherjee and Wichs presented a construction of MKFHE scheme MW16 [9] based on LWE that simplifies the scheme of CM15 and admits a simple 1-round threshold decryption protocol. Based on this threshold MKFHE, they successfully constructed a 2-round MPC protocol upon it in the common random string (CRS) model. The schemes CM15 and MW16 need to determine all the involved parties before the homomorphic computation and do not allow any new party to join in, which is called single-hop MKFHE [10]. In TCC2016, Peikert and Shiehian proposed a notion of multi-hop MKFHE PS16 [10], in which the calculated ciphertexts can be used in further homomorphic computations involving additional parties. That is, any parties can dynamically join the homomorphic computation at any time. However, the disadvantage is that the number of parties is limited. In CRYPTO2016, A similar notion named fully dynamic MKFHE BP16 [11] was proposed by Brakerski and Perlman. A slight difference is that the bound of the number of parties does not need to be input during the setup procedure in fully dynamic MKFHE. The length of extended ciphertext only increases linearly with the number of parties. However, in the process of homomorphic computation, the scheme needs to use the parties' joint public key to run the bootstrap process, so the efficiency of ciphertext computation is low.

In ASIACRYPT2016, Chillotti et al. constructed the fully homomorphic scheme CGGI16 [12] based on a variant of GSW13 [13] on the T = (0,1] ring. In the scheme, the external product of TGSW ciphertext (matrix) and TLWE ciphertext (vector) is used to replace the product of TGSW ciphertext (matrix) and TGSW ciphertext (matrix). Therefore, the addition operation on polynomial exponent is more efficient, such that the time of the bootstrap process and the size of the bootstrap key are greatly reduced. In ASIACRYPT2017, Chilotti et al. optimized the accumulation process in the CGGI16 scheme and proposed CGGI17 [14], which reduced the bootstrap time to 13ms. In ASIACRYPT2019, Chen et al. designed an efficient ciphertext expansion algorithm based on CGGI17, realized the efficient expansion evaluation key, and proposed an MKFHE scheme CCS19 [15]. The ciphertext length of the scheme increases linearly with the number of parties. And also, they compiled an MKFHE software library MKTFHE, which has important guiding significance for the application of MKFHE schemes. However, these TFHE-type MKFHE schemes do not support the packaging technique, thereby resulting in a large expansion rate similar to TFHE.

In TCC2017, Chen et al. proposed the first BGV-type multi-hop MKFHE scheme CZW17 [16]. They used the GSW-type expansion algorithm to encrypt the secret key to generate the joint evaluation key of the party set. CZW17 supports the ciphertext packaging technology based on the Chinese Remainder Theory (CRT) and can be used to construct a 2-round MPC protocol. In 2019, Li et al. [17] put forward a nested ciphertext extension method, which reduces the size of the evaluation key and the ciphertext. In 2019, Chen et al. optimized the relinearization process and constructed an efficient MKFHE scheme [18], that is called the CDKS19 scheme. Because of its efficient homomorphic computation, it is applied to the neural network to perform the privacy computation. Our work is focused on the BGV-type MKFHE scheme.

### 1.2 Our Contributions

Fully homomorphic encryption schemes [e.g. 13, 19, 20] are the very attractive cryptography primitive, but they are limited to a single party. The multi-key FHE scheme can realize the homomorphic operation of multiple parties, but its efficiency of homomorphic operation is lower than that of single party FHE scheme. The RLWE-based FHE scheme has high security and good operation speed, so it has numerous theoretical and practical applications. For example, the CKKS17 [21] scheme has been widely applied for its efficient homomorphic operation. However, Li and Micciancio [22] found the security problems existing in CKKS17 and gave remedial measures. Recently, Cheon et al. published an announcement [23] and fixed the security loopholes of CKKS17. Another example is the CDKS19 [18] scheme, which is applied to the privacy-preserving operation of neural networks for its fast relinearization. Our work is to learn from the advantages of BGV-type FHE, and further improve them to design a more efficient leveled MKFHE scheme.

(1) Aiming at the CDKS19 scheme, we analyze the shortcomings of its specific relinearization algorithm. By reducing the public key and adding the auxiliary key, we improve the generation method of the evaluation key and propose two efficient generation algorithms of the relinearization key.

(2) Using the optimized generation algorithms of relinearization key, we modify the relinearization algorithm following the CDSK19 framework, so that the size of evaluation key and relinearization key is greatly reduced, and the efficiency of relinearization is higher.
(3) According to the optimized relinearization algorithms, we design a leveled MKFHE scheme combining with the modulus-switching technology. The scheme does not need to perform the key-switching process, so it is more efficient in the homomorphic computation process.

### 1.3 Overview of Our Construction

The relinearization idea of CDKS19 scheme is as follows. For the party $1 \leq i, j \leq k$, the method combines the $i$-th evaluation key $\mathbf{D}_i = [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}|\mathbf{d}_{i,2}] \in R_q^{d\times 3}$ with the $j$-th public key $\mathbf{b}_j \approx -s_j \cdot \mathbf{a}(\mathrm{mod}q) \in R_q^d$ to generate the relinearization key $\mathbf{K}_{i,j} \in R_q^{d\times 3}$ such that $\mathbf{K}_{i,j} \cdot (1, s_i, s_j) \approx s_i s_j \cdot \mathbf{g}(\mathrm{mod}q)$. For the set with $k$ parties, taking the homomorphic multiplication of parties $i$ and $j$ as an example, the product of their ciphertexts is $\overline{\mathbf{ct}} = (\mathbf{c}_i \otimes \mathbf{c}_j) = \{c_{i,j}\}_{0\leq i,j\leq k}$, and the corresponding joint secret key for decryption is $\mathbf{s} = (1, s_1, s_2, \cdots, s_k) \in \chi^{k+1}$. The purpose of relinearization is to make $\langle \overline{\mathbf{ct}}, \mathbf{s} \otimes \mathbf{s} \rangle = \langle \mathbf{c}_{\mathrm{new}}, \mathbf{s} \rangle$ hold, where $\mathbf{c}_{\mathrm{new}} \in R_q^{k+1}$ is a new ciphertext. Only two parties' keys are used in the relinearization, and it does not need to perform the RGSW ciphertext expansion algorithm in CZW17 [16] and LZY+19 [17] schemes, so the computation efficiency is high. However, the size of evaluation key $\mathbf{D}_i \in R_q^{d\times 3}$ is still large, and the relinearization key $\mathbf{K}_{i,j}$ of evaluation key is also large.

In this paper, we choose the appropriate polynomial public parameter $a \leftarrow R_q$ to reduce the public key size, that is $b_j \approx -s_j \cdot a(\mathrm{mod}\ q) \in R_q$. By improving the generation method of evaluation key, we reduce the size of evaluation key $\mathbf{D}_i$ such that $\mathbf{K}_{i,j} \in R_q^3$, which greatly simplifies the relinearization key and improves the generation efficiency of evaluation key. The first method is to select another random vector $\mathbf{a} \leftarrow U(R_q^d)$, generate the party's auxiliary key $\mathbf{p}_i = s_i\mathbf{a} + \mathbf{e} \in R_q^d$, and then output the evaluation key $\mathbf{D}_i = [\mathbf{d}_0|d_1] \in R_q^{d+1}$. Combining with the $j$-th public key $b_j \approx -s_j \cdot a(\mathrm{mod}q) \in R_q$, we calculate $[k_{i,j,0}|k_{i,j,1}] = \mathbf{g}^{-1}(b_j) \cdot [\mathbf{d}_0|\mathbf{a}]$ and $k_{i,j,2} = d_1$. Finally, we obtain the relinearization key $\mathbf{K}_{i,j} = [k_{i,j,0}|k_{i,j,1}|k_{i,j,2}] \in R_q^3$. The second method is to select a modulus $P$ and the random vector $\mathbf{d} \leftarrow U(R_{P\cdot q}^d)$, generate the party's auxiliary key $\mathbf{p}_i = s_i\mathbf{d} + \mathbf{e} \in R_{P\cdot q}^d$. We calculate $\mathbf{d}_0 = -\mathbf{p}_i + \mathbf{e}_1 + r_i \cdot \mathbf{g}(\mathrm{mod}P \cdot q)$ and $d_1 = r_i \cdot a + e_2 + P \cdot \mu(\mathrm{mod}P \cdot q)$, and then obtain the evaluation key $\mathbf{D}_i = [\mathbf{d}_0|d_1] \in R_q^{d+1}$. By calculating $[k_{i,j,0}|k_{i,j,1}] = \mathbf{g}^{-1}(b_j) \cdot [\mathbf{d}_0|\mathbf{d}]$ and $k_{i,j,2} = d_1$, we finally get $\mathbf{K}_{i,j} = [k_{i,j,0}|k_{i,j,1}|k_{i,j,2}] \in R_{P\cdot q}^3$. Through the above two methods, by reducing the using times of function $\mathbf{g}$, we decrease the size of evaluation key and relinearization key, so that to improve the generation efficiency of evaluation key. Furthermore, in Sect. 3, two optimized relinearization algorithms are proposed. Combined with the modulus switching technology, an efficient leveled MKFHE scheme is designed in Sect. 4.

## 2 Preliminaries

### 2.1 Basic Notation

We denote vectors in bold, e.g. $\mathbf{a}$, and matrices in upper-case bold, e.g. $\mathbf{A}$. We denote by $\langle \mathbf{u}, \mathbf{v} \rangle$·the usual dot product of two vectors $\mathbf{u}, \mathbf{v}$. For a security parameter $\lambda$ and a positive integer $m$, let $\Phi_m(X)$ denote the $m$-th cyclotomic polynomial with the degree $n = \phi(m)$, where $\phi(\cdot)$ is the Euler's function. We work over rings $R = \mathbb{Z}[X]/\Phi_m$ and $R_q = R/qR$ for a prime integer $q = q(\lambda)$. Addition and multiplication in these rings are done component-wise in their coefficients, and the coefficients in $R_q$ are reduced in $[-q/2, q/2)$(except for $q = 2$). Let $\psi = \psi(\lambda)$ be a $B$-bound error distribution over $R$ whose coefficients are in the range $[-B, B]$. For a probability distribution $D$, $x \leftarrow D$ denotes that $x$ is sampled from $D$, and $x \leftarrow U(D)$ denotes that $x$ is sampled uniformly from $D$. For $a \in R$, we use $\|a\|_\infty = \max_{0 \le i \le n-1} |a_i|$ to denote the standard $l_\infty$-norm and use $\|a\|_1 = \sum_{i=0}^{n-1} |a_j|$ to denote the standard $l_1$-norm.

### 2.2 Leveled Multi-key FHE

We now introduce the cryptographic definition of a leveled multi-key FHE, which is defined in CZW17.

**Definition 1 (Multi-key FHE) [16].** Let $\mathcal{C}$ be a class of circuits. A leveled multi-key FHE scheme $\mathcal{E} = (Setup, KeyGen, Enc, Eval, Dec)$ is described as follows:

- Setup$(1^\lambda, 1^K, 1^L)$: Given the security parameter $\lambda$, the circuit depth $L$, and the number of distinct users $K$ that can be tolerated in an evaluation, outputs the public parameters $pp$.
- KeyGen$(pp)$: Given the public parameters $pp$, derives and outputs a public key $pk_i$, a secret key $sk_i$, and the evaluation keys $evk_i$ of party $i (i = 1, \ldots, K)$.
- Enc$(pk_i, \mu)$: Given a public key $pk_i$ and message $\mu$, outputs a ciphertext $ct_i$.
- Dec$((sk_{i_1}, sk_{i_2}, \ldots, sk_{i_k}), ct_S)$: Given a ciphertext $ct_S$ corresponding to a set of users $S = \{i_1, i_2, \ldots, i_k\} \subseteq [K]$, and their secret keys $sk_S = \{sk_{i_1}, sk_{i_2}, \ldots, sk_{i_k}\}$, outputs the message $\mu$.
- Eval$(\mathcal{C}, (ct_{S_1}, pk_{S_1}, evk_{S_1}), \ldots, (ct_{S_t}, pk_{S_t}, evk_{S_t}))$: On input a Boolean circuit $\mathcal{C}$ along with $t$ tuples $(ct_{S_i}, pk_{S_i}, evk_{S_i})_{i=1,\ldots,t}$, each tuple comprises of a ciphertext $ct_{S_i}$ corresponding to a user set $S_i$, a set of public keys $pk_{S_i} = \{pk_j, \forall j \in S_i\}$, and the evaluation keys $evk_{S_i}$, outputs a ciphertext $ct_S$ corresponding to a set of secret keys indexed by $S = \cup_{i=1}^{t} S_i \subseteq [K]$.

### 2.3 RLWE Problem

Refers [24] and [25] for explaining the RLWE problem in more details. We use this discrete distribution as the RLWE error distribution. Here we define the RLWE distribution and decisional problem associated with it. Let $R^\vee$ be the dual fractional ideal of $R$ and write $R_q = R^\vee/qR^\vee$. For a positive integer modulus $q \ge 2$, $s \in R_q^\vee$, and an error distribution $\psi$, we define $A_{N,q,\psi}(s)$ as the RLWE distribution obtained by sampling $a \leftarrow R_q$

uniformly at random, $e \leftarrow \psi$ and returning $(a, a \cdot s + e) \in R_q \times R_q^\vee$. The (decision) ring learning with errors, denoted by $\mathrm{RLWE}_{N,q,\chi}(D)$, is a problem to distinguish arbitrarily many independent samples chosen according to $A_{N,q,\chi}(s)$ for a random choice of $s$ sampled from the distribution $s \leftarrow \chi$ over $R^\vee$ from the same number of uniformly random and independent samples from $R_q \times R_q^\vee$.

### 2.4 Two Techniques

**Gadget Decomposition [18].** Let $\mathbf{g} = (\partial_i) \in Z^d$ be a gadget vector and $q$ an integer. The gadget decomposition, denoted by $\mathbf{g}^{-1}$, is a function from $R_q$ to $R_2^d$ which transforms an element $a \in R_q$ into a vector $\mathbf{u} = (u_0, u_1, \cdots, u_{d-1}) \in R^d$ of small polynomials such that $a = \sum_{i=0}^{d-1} \partial_i \cdot u_i (\bmod q)$. The gadget decomposition technique is widely used in the construction of HE schemes, such as bit decomposition [13, 26], base decomposition [12, 14], and RNS-based decomposition [27]. Our implementation exploits the bit decomposition for efficiency.

**Modulus-Switching [19, 28].** Since the error involved in the ciphertext grows with homomorphic operations, modulus switching which can change the inner modulus $q_{l+1}$ of ciphertext $\mathbf{c}_1$ to a smaller number $q_l$ is used to reduce the error term roughly by the ratio $q_{l+1}/q_l$, while preserving the correctness of decryption under the same secret key.

- ModulusSwitch$(\mathbf{c}_1, q_{l+1}, q_l)$: On input $\mathbf{c}_1 \in R_{q_{l+1}}^{n_1}$ and another smaller modulus $q_l$, output $\mathbf{c}_2 \in R_{q_l}^{n_1}$ which is the closest element to $(q_l/q_{l+1}) \cdot \mathbf{c}_1$.

### 2.5 Relinearization Algorithm in CDKS19

The CDKS19 scheme provides two special relinearization methods. They have similar structure patterns, so we only analyze the first method. The details are as follows. (See refer [18] for details).

1. **Parameter selection**. For a given security parameter $\lambda$, set the RLWE dimension be $n$, ciphertext modulus be $q$, polynomial distribution $\chi$ with small coefficients and error distribution $\psi$ over $R$. Generate a random vector $\mathbf{a} \leftarrow U(R_q^d)$. For the party $i$, Sample the secret key $s_i \leftarrow \chi$. Sample an error vector $\mathbf{e} \leftarrow \psi^d$ and set the public key as $\mathbf{b}_i = -s_i \cdot \mathbf{a} + \mathbf{e} (\bmod q)$ in $R_q^d$.
2. **Evaluation key generation.**

- EvkGen$(s_i)$: For the party $i$, input his secret key $s_i \leftarrow \chi$, generate the ciphertext $\mathbf{D}_i = [\mathbf{d}_{i,0} | \mathbf{d}_{i,1} | \mathbf{d}_{i,2}] \in R_q^{d \times 3}$ as the evaluation key.

(1) Sample $r_i \leftarrow \chi$;
(2) Sample $\mathbf{d}_{i,1} \leftarrow U(R_q^d)$ and $\mathbf{e}_1 \leftarrow \psi^d$, and set $\mathbf{d}_{i,0} = -s_i \cdot \mathbf{d}_{i,1} + \mathbf{e}_1 + r_i \cdot \mathbf{g} (\bmod q)$;
(3) Sample $\mathbf{e}_2 \leftarrow \psi^d$ and set $\mathbf{d}_{i,2} = r_i \cdot \mathbf{a} + \mathbf{e}_2 + s_i \cdot \mathbf{g} (\bmod q)$.

**Note**: the reason of introducing function $\mathbf{g}$ is to control the error growth, and the disadvantage is to increase the storage space.

(3) **Relinearization key generation.** For every party, they need to provide his public key and evaluation key to the cloud server. For the $i$-th party, he uses $\mathbf{b}_j$ of $j$–th party to generate his relinearization key as follows.

- Convert($\mathbf{D}_i, \mathbf{b}_j$): It takes as the input a pair of an uni-encryption $\mathbf{D}_i = [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}|\mathbf{d}_{i,2}] \in R_q^{d\times 3}$ and a public key $\mathbf{b}_j \in R_q^d$ generated by (possibly different) parties $i$ and $j$. Let $\mathbf{k}_{i,j,0}$ and $\mathbf{k}_{i,j,1}$ be the vectors in $R_q^d$ such that $\mathbf{k}_{i,j,0}[\varsigma] = \left\langle \mathbf{g}^{-1}(\mathbf{b}_j[\varsigma]), \mathbf{d}_{i,0} \right\rangle$ and $\mathbf{k}_{i,j,1}[\varsigma] = \left\langle \mathbf{g}^{-1}(\mathbf{b}_j[\varsigma]), \mathbf{d}_{i,1} \right\rangle$ for $1 \le \varsigma \le d$, i.e., $[\mathbf{k}_{i,j,0}|\mathbf{k}_{i,j,1}] = \mathbf{M}_j \cdot [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}]$ where $\mathbf{M}_j \in R_q^d$ is the matrix whose $\varsigma$-th row is $\mathbf{g}^{-1}(\mathbf{b}_j[\varsigma]) \in R^d$. Let $\mathbf{k}_{i,j,2} = \mathbf{d}_{i,2}$ and return the relinearization key $\mathbf{K}_{i,j} = [\mathbf{k}_{i,j,0}|\mathbf{k}_{i,j,1}|\mathbf{k}_{i,j,2}] \in R_q^{d\times 3}$.

$$\left[\mathbf{k}_{i,j,0}|\mathbf{k}_{i,j,1}\right] = \begin{bmatrix} \mathbf{g}^{-1}(\mathbf{b}_j[1]) \\ \vdots \\ \mathbf{g}^{-1}(\mathbf{b}_j[d]) \end{bmatrix} \left[\mathbf{d}_{i,0}|\mathbf{d}_{i,1}\right] \in R_q^{d\times 2}, \ \left[\mathbf{k}_{i,j,2}\right] = \left[\mathbf{d}_{i,2}\right] \in R_q^d$$

4. **Relinearization algorithm.** For the homomorphic multiplication of parties $i$ and $j$, their ciphertexts follow the ciphertext expansion method in scheme LZY+19, e.g. $\mathbf{c}_i = (c_{i_0}|\, c_{i_1}|\cdots|c_{i_k}) \in R_{q_l}^{k+1}$. Their joint secret keys are all $\mathbf{s} = (1, s_1, s_2, \cdots, s_k) \in R_q^{k+1}$. Decrypting their ciphertext product $\overline{\mathbf{ct}} = (\mathbf{c}_i \otimes \mathbf{c}_j) = \{c_{i,j}\}_{0\le i,j\le k}$, we can get

$$\left\langle \overline{\mathbf{ct}}, \mathbf{s}\otimes\mathbf{s} \right\rangle = c_{0,0} + \sum_{i=1}^{k} (c_{0,i} + c_{i,0}) s_i$$

$$+ \sum_{i,j=1}^{k} \mathbf{g}^{-1}(c_{i,j}) \cdot \mathbf{K}_{i,j} \cdot (1, s_i, s_j) \in R_q^{k+1}$$

The function of the relinearization algorithm is to make $\left\langle \overline{\mathbf{ct}}, \mathbf{s}\otimes\mathbf{s} \right\rangle = \left\langle \overline{\mathbf{ct}'}, \mathbf{s} \right\rangle$ hold for the new ciphertext $\overline{\mathbf{ct}}' = (c_0', c_1', \cdots, c_k') \in R_q^{k+1}$ (corresponding to the joint secret key $\mathbf{s} = (1, s_1, s_2, \cdots, s_k) \in R_q^{k+1}$).

So, the relinearization algorithm is described as follows.

1. $c_0' \leftarrow c_{0,0}$ ;
2: for $1 \le i \le k$ do
$\quad c_i' \leftarrow c_{i,0} + c_{0,i} (\mathrm{mod}\, q)$
3: for $1 \le i, j \le k$ do
$\quad (c_0', c_i', c_j') \leftarrow (c_0', c_i', c_j') + \mathbf{g}^{-1}(c_{i,j}) \cdot \mathbf{K}_{i,j} (\mathrm{mod}\, q)$

Chen et al. [15] and Li et al. [17] designed multi-key variants of BGV [19] by generating a relinearization key based on the multi-key GSW scheme. However, it consists of $O(k^2)$ key-switching keys from $s_i \cdot s_j$ to the ordinary key each of which has $O(k)$ components. CDKS19 is an extension of these researches in the sense that our relinearization method and other optimization techniques can be applied to BGV as well. Compared

with CCS19 [15] and LZY+19 [17], the memory (bit-size) complexity of evaluation key in CDKS19 reduces to $O(kn)$, and the computational costs of the homomorphic multiplication reduces to $O(k^2n)$.

## 3 Optimizations of the Relinearization

In the CDKS19 scheme, the main reason for increasing the dimension of ciphertext is the introduction of function **g**, which is used to reduce the error to avoid generating the wrong evaluation key. In this section, we optimize the relinearization algorithm of CDKS19 to improve the efficiency of homomorphic multiplication decryption.

### 3.1 Two Optimized Generation Algorithms of Evaluation Key

**1. Method 1**

(1) **Parameter reselection**. For a given security parameter $\lambda$, set the RLWE dimension be $n$ and the modulus be $q$. The party set $K$ contains $k$ parties, for the party $i \in [K]$, uniformly sample the polynomial $s_i$ at random with the coefficients $\{-1, 0, 1\}$ in $R$, generate his secret key $sk_i := (1, s_i) \in R_3^2$. Choose $\mathbf{a} \leftarrow U(R_q^d)$, $a \leftarrow U(R_q)$ and $e \leftarrow \psi$, $\mathbf{e} \leftarrow \psi^d$. Generate his public key $pk_i := (b_i, a) \in R_q^2$, where $b_i = -s_i \cdot a + e (\mathrm{mod}\, q)$, and his auxiliary key $\mathbf{p}_i := s_i\mathbf{a} + \mathbf{e} \in R_q^d$.

Output the $(sk_i, pk_i, \mathbf{p}_i)$ of party $i$. Therefore, the memory space of $sk_i$ and $pk_i$ is lower than that of the CDKS19 scheme.

(2) **Evaluation key generation**. The $i$-th party uses his auxiliary key $\mathbf{p}_i$ to generate the evaluation key.

- UniEnc$(\mu; \mathbf{p}_i) : \mathbf{D} = [\mathbf{d}_0 | d_1]$:
  1) Sample $r_i \leftarrow \chi$.
  2) Sample $\mathbf{e}_1 \leftarrow \psi^d$ and set $\mathbf{d}_0 = -\mathbf{p}_i + \mathbf{e}_1 + \mathbf{g} \cdot r_i (\mathrm{mod} q)$.
  3) Sample $e_2 \leftarrow \psi$ and set $d_1 = -r_i \cdot a + e_2 + \partial_\varsigma \mu (\mathrm{mod}\, q)$, where $0 \le \varsigma \le d-1$.
- EvkGen$(s)$: Given the secret $s_i$, return $\mathbf{D}_i \leftarrow$ UniEnc$(s_i; \mathbf{p}_i)$. We call $\mathbf{D}_i$ the evaluation key of party $i$, that is $ek_i := \mathbf{D}_i$.

Every parties provide their $(pk_i, \mathbf{p}_i, ek_i)$ to the cloud server for homomorphic computing.

**Security.** The evaluation key generation algorithm is IND-CPA secure under the RLWE assumption of parameter $(n, q, \chi, \psi)$. We prove it by showing that the distribution $\{a, (pk_i, \mathbf{p}_i), (\mathbf{d}_0, \mathbf{a}, d_1)\}$ is computationally indistinguishable from the uniform distribution over $R_q \times R_q^{d+2} \times R_q^{d\times 2+1}$ for an arbitrary $\mu \in R$.

Firstly, we change the public key $b_i \leftarrow U(R_q)$ and auxiliary key $\mathbf{p}_i \leftarrow U(R_q^d)$ according to the RLWE assumption.

Secondly, we change the partial evaluation key $\mathbf{d}_0 \leftarrow U(R_q^d)$.

Thirdly, since $d_1$ is RLWE assumption hard about $r_i \leftarrow \chi$, and is independent of the choice of $s_i$, we change $d_1 \leftarrow U(R_q)$. So, the new distribution $\{(pk_i, \mathbf{p}_i) \leftarrow U(R_q^{d+2}),$

$(\mathbf{d}_0, d_1) \leftarrow U(R_q^{d+1})\}$ is independent of the given plaintext $\mu$, we conclude that the algorithm is IND-CPA secure.

(3) **Relinearization key generation**.

- Convert($\mathbf{D}_i, b_j$): Input an evaluation key $\mathbf{D}_i = [\mathbf{d}_{i,0}|d_{i,1}]$ and a public key $b_j \in R_q$ generated by (possibly different) parties $i$ and $j$. Let $k_{i,j,0}$ and $k_{i,j,1}$ be the polynomials in $R_q$ such that $k_{i,j,0} = \langle \mathbf{g}^{-1}(b_j), \mathbf{d}_{i,0} \rangle$ and $k_{i,j,1} = \langle \mathbf{g}^{-1}(b_j), \mathbf{a} \rangle$ for $0 \le \varsigma \le d-1$, i.e., $[k_{i,j,0}|k_{i,j,1}] = \mathbf{g}^{-1}(b_j) \cdot [\mathbf{d}_{i,0}|\mathbf{a}]$. Let $k_{i,j,2} = d_{i,1}$ and return the relinearization key $\mathbf{K}_{i,j} = [k_{i,j,0}|k_{i,j,1}|\, k_{i,j,2}] \in R_q^3$.

**Correctness.** A shared relinearization key consists of encryptions of $s_i, s_j$ for all pairs $1 \le i, j \le k$. We first claim that, if $\mathbf{D}_i$ is an uni-encryption of $s_i \in R$ encrypted by the $i$-th party and $b_j$ is the public key of the $j$-th party, then the output $\mathbf{K}_{i,j} \leftarrow$ Convert($\mathbf{D}_i, b_j$) of the conversion algorithm is an encryption of $s_i s_j \in R$ with respect to the secret $(1, s_i, s_j)$. It is derived from the following formulas:

$$
\begin{aligned}
&\mathbf{K}_{i,j} \cdot (1, s_i, s_j) = \mathbf{g}^{-1}(b_j)[\mathbf{d}_0|\mathbf{a}] \begin{bmatrix} 1 \\ s_i \end{bmatrix} + s_j k_{i,j,1} \\
&= \mathbf{g}^{-1}(b_j) \cdot \left[(\mathbf{g} \cdot r_i - s_i \cdot \mathbf{a} + \mathbf{e}_1) + s_i \cdot \mathbf{a}\right] - r_i b_j + s_j e_2 + \partial_\varsigma \cdot s_i s_j \\
&= \left[\mathbf{g}^{-1}(b_j)\right]_{1\times d} \cdot [\mathbf{e}_1]_{d\times 1} + b_j r_i + r_i e_j - r_i b_j + s_j e_2 + \partial_\varsigma \cdot s_i s_j \\
&= \mathbf{g}^{-1}(b_j) \cdot \mathbf{e}_i + s_j e_j + r_i e_j + \partial_\varsigma \cdot s_i s_j \\
&= \partial_\varsigma \cdot s_i s_j + e_{\text{small}}
\end{aligned}
$$

**2. Method 2**

This method is based on the encryption mode of CKKS17 scheme to improve the generation algorithm of evaluation key. In the first method, the traditional BGV-type encryption is used, so the form of decryption is $\langle \mathbf{c}, sk \rangle = m + te(\text{mod}q)$. The function $\mathbf{g}$ used in the relinearization algorithm is to reduce the error. However, if we get a decryption form like $\langle \mathbf{c}, sk \rangle = Pm + te(\text{mod}P \cdot q)$, we can make the error $P$ times smaller than that of the original form so that the using times of function $\mathbf{g}$ can be decreased.

(1) **Parameters reselection**. For a given security parameter $\lambda$, set the RLWE dimension be $n$, the modulus be $q$, and an integer $P = P(\lambda, q)$. For the party $i \in [K]$, uniformly sample the polynomial $s_i$ at random with the coefficients $\{-1, 0, 1\}$ in $R$, generate his secret key $sk_i := (1, s_i) \in R_3^2$. Choose $a \leftarrow U(R_q)$, $\mathbf{d} \leftarrow U(R_{P\cdot q}^d)$ and $e \leftarrow \psi$, $\mathbf{e} \leftarrow \psi^d$. Generate his public key $pk_i := (b_i, a) \in R_q^2$, where $b_i = -s_i \cdot a + e(\text{mod}q)$, and his auxiliary key $\mathbf{p}_i := s_i\mathbf{d} + \mathbf{e} \in R_{P\cdot q}^d$.

(2) **Evaluation key generation.** The $i$-th party uses his auxiliary key $\mathbf{p}_i$ to generate the evaluation key.

- UniEnc($\mu; \mathbf{p}_i$) : $\mathbf{D} = [\mathbf{d}_0|d_1]$:
  1) Sample $r_i \leftarrow \chi$.
  2) Sample $\mathbf{e}_1 \leftarrow \psi^d$, and set $\mathbf{d}_0 = -\mathbf{p}_i + \mathbf{e}_1 + \mathbf{g} \cdot r_i (\text{mod}P \cdot q)$.
  3) Sample $e_2 \leftarrow \psi$ and set $d_1 = -r_i \cdot a + e_2 + P \cdot \mu (\text{mod}P \cdot q)$.

- EvkGen($s$): Given the secret $s_i$, return $ek_i := \mathbf{D}_i \leftarrow \text{UniEnc}(s_i; \mathbf{p}_i)$. Every parties provide their $(pk_i, \mathbf{p}_i, ek_i)$ to the cloud server for homomorphic computing.

**Security.** As the proof of method 1, the evaluation key generation algorithm in method 2 is also IND-CPA secure under the RLWE assumption of parameter $(n, q, \chi, P, \psi)$.

(3) **Relinearization key generation.**

- Convert($\mathbf{D}_i, b_j$): Input the evaluation key $\mathbf{D}_i = [\mathbf{d}_{i,0}|d_{i,1}]$ and a public key $b_j \in R_q$ generated by (possibly different) parties $i$ and $j$. Let $k_{i,j,0}$ and $k_{i,j,1}$ be the polynomials in $R_q$ such that $k_{i,j,0} = \langle \mathbf{g}^{-1}(b_j), \mathbf{d}_{i,0} \rangle$ and $k_{i,j,1} = \langle \mathbf{g}^{-1}(b_j), \mathbf{d} \rangle$ for $1 \le \varsigma \le d$, i.e., $[k_{i,j,0}|k_{i,j,1}] = \mathbf{g}^{-1}(b_j) \cdot [\mathbf{d}_{i,0}|\mathbf{d}]$. Let $k_{i,j,2} = d_{i,1}$ and return the relinearization key $\mathbf{K}_{i,j} = [k_{i,j,0}|k_{i,j,1}|\ k_{i,j,2}] \in R_{P\cdot q}^3$.

**Correctness.** A shared relinearization key consists of encryptions of $s_i, s_j$ for all pairs $1 \le i, j \le k$. We first claim that, if $\mathbf{D}_i$ is an uni-encryption of $s_i \in R$ encrypted by the $i$-th party and $b_j$ is the public key of the $j$-th party, then the output $\mathbf{K}_{i,j} \leftarrow \text{Convert}(\mathbf{D}_i, b_j)$ of the conversion algorithm is an encryption of $s_i s_j \in R$ with respect to the secret $\big(1, s_i, s_j\big)$. It is derived from the following formulas:

$$\begin{aligned}
&\mathbf{K}_{i,j} \cdot \big(1, s_i, s_j\big) = \mathbf{g}^{-1}(b_j)\big[\mathbf{d}_{i,0}|\mathbf{d}\big]\begin{bmatrix}1\\ s_i\end{bmatrix} + s_j d_{i,1} \\
&= \mathbf{g}^{-1}(b_j) \cdot (\mathbf{e}_1 + r_i \cdot \mathbf{g}) - r_i b_j + s_j e_2 + P \cdot s_i \cdot s_j \\
&= \mathbf{g}^{-1}(b_j) \cdot \mathbf{e}_1 + r_i e_j + s_j e_2 + P \cdot s_i \cdot s_j \\
&= P \cdot s_i \cdot s_j + e'_{\text{small}}
\end{aligned}$$

### 3.2 The Optimized Relinearization

The goal of relinearization is to re-linearize the tensor product of ciphertexts. So in this section, we design the optimized relinearization algorithm following the algorithm framework of CDKS19.

Let $K$ be an ordered set containing all indexes of users that the ciphertext corresponding to, and we assume that the indexes are arranged from small to large and $K$ has no duplicate elements, thus we can describe a ciphertext as a tuple $ct = \{\mathbf{c}, K, l\}$. For the ciphertext product $\overline{\mathbf{ct}} = (\mathbf{c}_i \otimes \mathbf{c}_j) = \{c_{i,j}\}_{0\le i,j\le k} \in R_q^{(k+1)\times(k+1)}$, we would re-linearize it to $\overline{\mathbf{ct}}' = (c'_i)_{0\le i\le k} \in R_q^{k+1}$.

- Relin($\overline{\mathbf{ct}}; \{(\mathbf{D}_i, b_j)\}_{1\le i\le k}$): Given an extended ciphertext $\overline{\mathbf{ct}} = (c_{i,j})_{0\le i,j\le k}$ and $k$ pairs of evaluation/public keys $\{(\mathbf{D}_i, b_j)\}_{1\le i\le k}$, generate a ciphertext $\overline{\mathbf{ct}}' \in R_q^{k+1}$ as follows:

1. Compute $\mathbf{K}_{i,j} \leftarrow \text{Convert}(\mathbf{D}_i, b_j)$ for all $0 \le i, j \le k$ and set the relinearization key as $\overline{rlk} = \{\mathbf{K}_{i,j}\}_{0\le i,j\le k}$.
2. Run the following Algorithm to relinearize $\overline{\mathbf{ct}}$.

Input: $\overline{\mathbf{ct}} = (c_{i,j})_{0\le i,j\le k}$, $\overline{rlk} = \{\mathbf{K}_{i,j}\}_{0\le i,j\le k}$.
Output: $\overline{\mathbf{ct}'} = (c'_i)_{0\le i\le k} \in R_q^{k+1}$.
1: $c'_0 \leftarrow c_{0,0}$
2: for $1 \le i \le k$ do
3: $c'_i \leftarrow c_{i,0} + c_{0,i} (\mathrm{mod}\, q)$
4: end for
5: for $1 \le i, j \le k$ do
6: Method 1:
$(c'_0, c'_i, c'_j) \leftarrow (c'_0, c'_i, c'_j) + \sum_{\varsigma=0}^{d-1} \left(c_{i,j}\right)_\varsigma \cdot \mathbf{K}_{i,j} (\mathrm{mod}\, q)$ ,where $\left(c_{i,j}\right)_\varsigma$ is a polynomial with the coefficients of 0 or 1.
Method 2:
$(c'_0, c'_i, c'_j) \leftarrow (c'_0, c'_i, c'_j) + P^{-1} \cdot c_{i,j} \cdot \mathbf{K}_{i,j} (\mathrm{mod}\, P \cdot q)$.
7: end for

**Correctness**

**For the method 1.** Since the evaluation key $\mathbf{D}_i \leftarrow \mathrm{UniEnc}(s_i; s_i)$ of the $i$-th party is an uni-encryption of $\mu_i = s_i$, we obtain that $\mathbf{K}_{i,j} \cdot \left(1, s_i, s_j\right) \approx \partial_\varsigma \cdot s_i s_j (\mathrm{mod} q)$. From the definition of $\overline{\mathbf{ct}'} \in R_q^{k+1}$ and the joint secret key $\mathbf{s} = (1, s_1, s_2, \cdots, s_k) \in R_q^{k+1}$, we get

$$
\begin{aligned}
&< \overline{\mathbf{ct}'}, \mathbf{s} >= c'_0 + \textstyle\sum_{i=1}^{k} c'_i \cdot s_i \\
&= c_{0,0} + \textstyle\sum_{i=1}^{k} (c_{i,0} + c_{0,i}) s_i \\
&+ \textstyle\sum_{i,j=1}^{k} \sum_{\varsigma=0}^{d-1} \left(c_{i,j}\right)_\varsigma \cdot \mathbf{K}_{i,j} \cdot \left(1, s_i, s_j\right) \ (\mathrm{mod} q) \\
&\approx c_{0,0} + \textstyle\sum_{i=1}^{k} (c_{i,0} + c_{0,i}) s_i + \sum_{i,j=1}^{k} c_{i,j} \cdot s_i s_j \\
&= < \overline{\mathbf{ct}}, \mathbf{s} \otimes \mathbf{s} > (\mathrm{mod} q)
\end{aligned}
$$

**For the method 2.** The evaluation key is $\mathbf{D}_i \leftarrow \mathrm{UniEnc}(s_i; s_i)$, and $\mu_i = s_i$. We obtain that $\mathbf{K}_{i,j} \cdot \left(1, s_i, s_j\right) \approx P \cdot s_i s_j (\mathrm{mod} q)$. So we get

$$
\begin{aligned}
&< \overline{\mathbf{ct}'}, \mathbf{s} >= c'_0 + \textstyle\sum_{i=1}^{k} c'_i \cdot s_i \\
&= c_{0,0} + \textstyle\sum_{i=1}^{k} (c_{i,0} + c_{0,i}) s_i \\
&+ P^{-1} \textstyle\sum_{i,j=1}^{k} c_{i,j} \cdot \mathbf{K}_{i,j} \cdot \left(1, s_i, s_j\right) (\mathrm{mod} P) \ (\mathrm{mod} q) \\
&\approx \ c_{0,0} + \textstyle\sum_{i=1}^{k} (c_{i,0} + c_{0,i}) s_i + \sum_{i,j=1}^{k} c_{i,j} \cdot s_i s_j \\
&= < \overline{\mathbf{ct}}, \mathbf{s} \otimes \mathbf{s} > (\mathrm{mod}\ q)
\end{aligned}
$$

### 3.3 Analysis of the Relinearization

The previous analysis showed that the security of our optimized algorithm is the same as CDKS. Here we compare the memory space and time complexity between our improved relinearization algorithm and that of CDKS19 to analyze the advantages of our optimized algorithm. See Table 1 for details.

It can be seen from Table 1 that compared to the CDKS19 scheme, the memory space of the evaluation key and relinearization key is reduced about times. Because the public keys can be generated off-line, the slight changes of the public key in Table 1 have

**Table 1.** Memory space and time complexity of the relinearization algorithm

| Relinearization types | | Space | | | Time complexity |
|---|---|---|---|---|---|
| | | Public key | Evaluation key | Relinearization key | |
| CDKS19 | | $2kdn\lceil\log q\rceil$ | $k(d\times 3)n\lceil\log q\rceil$ | $3dkn\lceil\log q\rceil$ | $O(d^3n)$ |
| Our method | Method 1 | $(kd+2k)n\lceil\log q\rceil$ | $k(d+1)n\lceil\log q\rceil$ | $3kn\lceil\log q\rceil$ | $O(d^2n)$ |
| | Method 2 | $kdn\lceil\log Pq\rceil+2kdn\lceil\log q\rceil$ | $k(d+1)n\lceil\log Pq\rceil$ | $3kn\lceil\log Pq\rceil$ | $O(dn)$ |

neglected impact on the computation efficiency. Moreover, the time complexity cost in the relinearization is smaller than that in CDKS19. So our relinearization algorithm is more efficient than CDKS19.

## 4 New Construction of BGV-Type MKFHE Scheme

Our relinearization algorithm can be used as well as CDKS. It can also be designed as a leveled MKFHE scheme. Firstly, the ciphertext of different levels is expanded. Then the optimized relinearization algorithm is used to re-linearize the tensor product of ciphertexts. Finally the modulus-switching is carried out to convert the result ciphertext into the next level.

### 4.1 The Ciphertext Extension

In this subsection, we detail the ciphertext extension algorithm MKFHE.CTExt in Refs. [16, 17], which converts a BGV-type ciphertext to a larger dimensional ciphertext corresponding to a common larger dimensional joint secret key. In fact, the joint secret key is a concatenation of secret keys from a party set.

For the security parameter $\lambda$, given a bound $K$ on the number of parties, a bound $L$ on the circuit depth with $L$ decreasing modulus $q_L \gg q_{L-1} \gg \cdots \gg q_0$ for each level and a small integer $p$ coprime with all $q_l$ (or an integer $P = P(\lambda, q)$). For the $i$-th party ($l = 0, \ldots, L$), output his key pairs $(sk_i, pk_{l,i}, ek_{l,i})$ and the ciphertext $\mathbf{c}_{l,i} \in R_q^2$.

Let $K$ be an ordered set containing all indexes of parties that the ciphertexts corresponding to, and we assume that the indexes are arranged from small to large and $K$ has no duplicate elements, thus we can describe a ciphertext as a tuple $ct = \{\mathbf{c}_l, K, l\}$, where $\mathbf{c}_l = (c_{l,i_0} | c_{l,i_1} | \cdots | c_{l,i_k}) \in R_{q_l}^{k+1}$ corresponding joint secret keys $\mathbf{s}_K = (1, s_{i_1}, ..., s_{i_k}) \in R_3^{k+1}$. Note that $\mathbf{s}_K = (1, s_{i_1}, ..., s_{i_k}) \in R_3^{k+1}$ satisfies all levels of ciphertext decryption. That is $\mu \leftarrow\!< \mathbf{c}_l, \mathbf{s}_K >\! (\mathrm{mod} q_l \bmod p)$. So when the party set updates, the ciphertext of party $i$ also changes.

• MKFHE.CTExt$(\mathbf{c}_l, K')$: On input a ciphertext tuple $ct = \{\mathbf{c}_l \in R_{q_l}^{k+1}, K = \{i_1, ..., i_k\}, l\}$ corresponding to $k$ parties and another party set $K' = \{j_1, ..., j_{k'}\}$ for $K \in K'$, output an extended tuple $ct' = \{\overline{\mathbf{c}}_l \in R_{q_l}^{k'+1}, K' = \{i_1, ..., i_{k'}\}, l\}$. The extending algorithm is as follows:

1. Divide the ciphertext $\mathbf{c}_l$ into $k+1$ sequential sub-vectors indexed by $K = \{i_1, ..., i_k\}$ (except for the first sub-vector), i.e.,

$\mathbf{c}_l = (c_{l,i_0}|c_{l,i_1}|\cdots|c_{l,i_k}) \in R_{q_l}^{k+1}$, where the corresponding secret key is $\mathbf{s}_K = (1, s_{i_1}, ..., s_{i_k}) \in R_3^{k+1}$.

2. The extended ciphertext $\bar{\mathbf{c}}_l$ consists of $k' + 1$ sequential sub-vectors, which can be indexed by $K' = \{j_1, ..., j_{k'}\}$, i.e., $\bar{\mathbf{c}} = (c'_{j_0}|c'_{j_1}|\cdots|c'_{j_{k'}}) \in R_{q_l}^{k'+1}$.

If index $j$ in $K'$ is also included in $K$, we set $c'_j = c_j$,otherwise we set $c'_j = 0$ (except for the first sub-vector). The corresponding secret key for decryption is $\mathbf{s}_{K'}=(1, s_{j_1}, ..., s_{j_{k'}}) \in R_3^{k'+1}$.

It's easy to verify that $\langle \mathbf{c}, \mathbf{s}_{K,l} \rangle = \langle \bar{\mathbf{c}}, \mathbf{s}_{K'} \rangle \bmod q_l$.

## 4.2 Detail Construction

1. **Structure of Our Scheme**

The specific structure of our leveled MKFHE scheme is shown in Fig. 1. The scheme architecture is mainly divided into two parts: key generation and homomorphic computation of ciphertexts. Take two parties as an example. For the key generation part: they select public parameters from the cloud server to generate their key pairs, and then upload them. The cloud server uses the parameters and the uploaded keys to generate the relinearization key. For the homomorphic computation part: all parties upload their ciphertexts to the cloud server. Firstly, the ciphertexts are expanded by the ciphertext extension algorithm. Then, homomorphic multiplication is performed on the ciphertexts, and use the relinearization key to re-linearize the tensor product of ciphertexts. Finally, output the computed ciphertext which can be decrypted as the resulting ciphertext or be switched as a new input ciphertext in the next level.

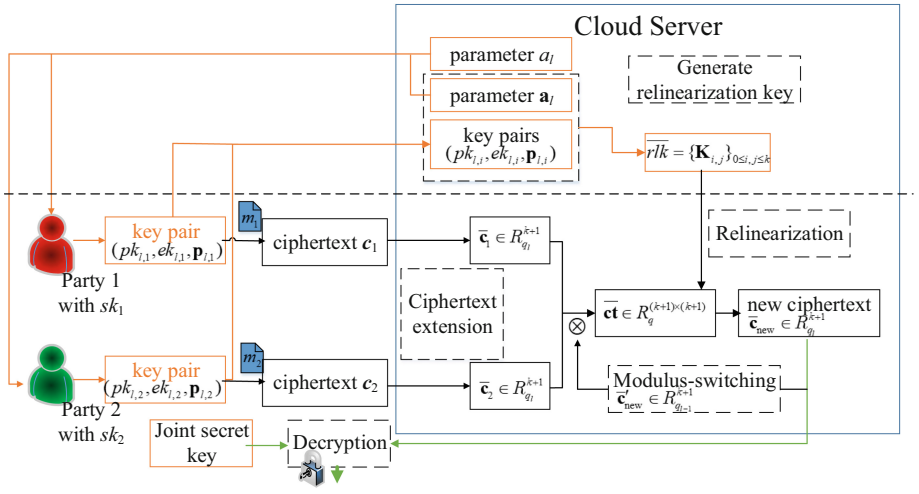


**Fig. 1.** The homomorphic multiplication structure of our leveled MKFHE

## 2. **Detail Construction**

According to the optimized relinearization, the general leveled BGV-type MKFHE is designed as follows.

• MKFHE.Setup($1^\lambda$, $1^K$, $1^L$): For the security parameter $\lambda$, given a bound $K$ on the number of keys, a bound $L$ on the circuit depth with decreasing modulus $q_L \gg q_{L-1} \gg \cdots \gg q_0$ (where $q_0 \gg 3$) for each level and a small integer $p$ coprime with all $q_l$ (or an integer $P = P(\lambda, q_l)$). We work over rings $R = \mathbb{Z}[X]/\Phi_m$ and $R_{q_l} = R/q_l R$ defined above. Let $\psi = \psi(\lambda)$ be a $B$-bound error distribution over $R$ whose coefficients are in the range $[-B, B]$. Let $\chi = \chi(\lambda)$ be a distribution over $R$ whose coefficients are in the range $(-B, B)$. Choose $L+1$ random public vectors $\mathbf{a}_l \leftarrow U(R_{q_l}^d)$ and polynomials $a_l \leftarrow U(R_{q_l})$ for $l \in \{0, \ldots, L\}$. All the following algorithms implicitly take the public parameter $pp = (R, B, \chi, \psi, \{q_l, \mathbf{a}_l, a_l\}_{l\in\{0,\ldots,L\}}, p, P)$ as input.

• MKFHE.KeyGen($pp$): Input the public parameter $pp$, and generate the keys of circuit depth $l$ for the $i$-th party.

1. Sample the polynomial $s_i$ at random with the coefficients $\{-1, 0, 1\}$ in $R$, and set the secret key $sk_{l,i} := (1, s_i) \in R_3^2$.

2. Sample $e_{l,i} \leftarrow \psi$, and set the public key $pk_{l,i} := (b_{l,i}, a_l) \in R_{q_l}^2$, where $b_{l,i} = -a_l s_i + p e_{l,i} \bmod q_l$.

3. Sample $\mathbf{e}_{l,i} \leftarrow \psi^d$, and generate the auxiliary key $\mathbf{p}_{l,i} = s_i \mathbf{a}_l + \mathbf{e}_{l,i} \in R_{q_l}^d$.

4. Sample $r_{l,i} \leftarrow \chi$, generate the evaluation key $ek_{l,i} := \mathbf{D}_{l,i}$.

Output the key pairs of party $i$ $(sk_i, pk_{l,i}, \mathbf{p}_{l,i})$.

• MKFHE.Enc($pk_{L,i}, \mu_i$): Input the plaintext $\mu_j \in R_p$ and the public key $pk_{L,i}$. Choose $r_i, e, e' \leftarrow \chi$, compute the ciphertext $\mathbf{c}_{L,i} = (c_{i,0}^L, c_{i,1}^L) = (r_i b_{L,i} + pe + \mu_i, r a_{L,i} + pe') \in R_{q_L}^2$.

• MKFHE.CTExt($\mathbf{c}_{L,i}, K$): Input the $i$-th party's ciphertext $\mathbf{c}_{L,i} \in R_{q_L}^2$, and output the ciphertext tuple $ct = \{\mathbf{c}_{L,i}, \{i\}, L\} \in R_{q_L}^{k+1}$.

• MKFHE.Dec($\mathbf{sk}_K, ct = \{\mathbf{c}_{L,i}, \{i\}, L\}$): Input the tuple $ct = (\mathbf{c}_{L,i}, K, l)$ and the joint secret keys $\bar{\mathbf{s}} = (1, s_1, ..., s_k) \in R_3^{k+1}$, output $\mu \leftarrow \langle ct, \bar{\mathbf{s}} \rangle \bmod q_l \bmod p$.

• MKFHE.Eval($(pk_{l,i}, ek_{l,i}, ct_i)_K, C$): Let $S$ be a subset of $K$, where $1 \le |S| \le k$. For another party $j$, performing the MKFHE.CTExt($\mathbf{c}_{l,i}, K$) and relinearization key generation program $\overline{rlk}_{l,S} \leftarrow \text{Convert}(\mathbf{D}_S, b_{l,j})$. The homomorphism operation is as follows:

1. MKFHE.EvalAdd($\overline{rlk}_{l,S}, \bar{\mathbf{c}}_{l,i}, \bar{\mathbf{c}}_{l,j}$): Input the ciphertexts $\bar{\mathbf{c}}_{l,i}, \bar{\mathbf{c}}_{l,j} \in R_{q_l}^{k+1}$ at the same level-$l$.

1) Compute $\bar{\mathbf{c}}_{l,\text{add}} := \bar{\mathbf{c}}_{l,i} + \bar{\mathbf{c}}_{l,j} \bmod q_l$, corresponding the joint secret key $\bar{\mathbf{s}} \in R_3^{k+1}$.

2) Compute $\bar{\mathbf{c}}'_{l-1,\text{add}} := \text{ModulusSwitch}(\bar{\mathbf{c}}_{l,\text{add}}, q_l, q_{l-1})$.

2. MKFHE.EvalMult($\overline{rlk}_{l,S}, \bar{\mathbf{c}}_{l,i}, \bar{\mathbf{c}}_{l,j}$): Input the ciphertexts $\bar{\mathbf{c}}_{l,i}, \bar{\mathbf{c}}_{l,j} \in R_{q_l}^{k+1}$ at the same level-$l$.

1) Compute $\bar{\mathbf{c}}_{l,\text{mult}} := \bar{\mathbf{c}}_{l,i} \otimes \bar{\mathbf{c}}_{l,j} \bmod q_l$, corresponding the joint secret key $\hat{\mathbf{s}} = \bar{\mathbf{s}} \otimes \bar{\mathbf{s}} \in R_3^{(k+1)^2}$.

2) Perform $\bar{\mathbf{c}}'_{l,\text{mult}} = \text{Relin}(\bar{\mathbf{c}}_{l,\text{mult}}; (\mathbf{D}_{l,i}, b_{l,j}))$, corresponding the joint secret key $\bar{\mathbf{s}} \in R_3^{k+1}$.

3) Compute $\bar{\mathbf{c}}'_{l-1,\text{mult}} \triangleq \text{ModulusSwitch}(\bar{\mathbf{c}}'_{l,\text{mult}}, q_l, q_{l-1})$.

### 4.3 Analysis of the Scheme

1. **Security Analysis**

Our optimized relinearization algorithm uses the single-key BGV-based encryption algorithm like CDKS19, as mentioned earlier, the security of optimized relinearization algorithm is based on RLWE assumption. Like most BGV-type MKFHE schemes [CZW17, LZY+19 et al.], our scheme security also rely on cyclic security assumption. And the modulus switching does not affect security. Therefore, our MKFHE is IND-CPA secure under the RLWE assumption of parameter $pp^{\mathrm{RLWE}}$.

2. **Correctness**

In our scheme, the joint secret key $\bar{\mathbf{s}} \in R_3^{k+1}$ synthesized by all parties' secret key can satisfy all $L$ levels ciphertext decryption. We take two ciphertexts' homomorphic multiplication as an example to analyze the correctness of our MKFHE scheme in the most complex case. That is, when the ciphertexts of parties $i$ and $j$ are homomorphiclly multiplied in level $l$, and there is a new party $t$ participating in homomorphic multiplication in level $l-1$. For parties $i$ and $j$, we get the product ciphertext $\bar{\mathbf{c}}'_{l-1,\mathrm{mult}} = \mathrm{ModulusSwitch}(\bar{\mathbf{c}}'_{l,\mathrm{mult}}, q_l, q_{l-1}) \in R_{q_{l-1}}^{k+1}$ in level $(l-1)$. Set the new party set $K' \supset K$. The ciphertext of new party $t$ is $\mathbf{c}_{l-1,t} \in R_{q_{l-1}}^2$. So the multiplication in level $(l-1)$ can be described as follows.

(1) Re-perform the ciphertext expansion algorithm $\bar{\mathbf{c}}'_{l-1,\mathrm{M}} \in R_{q_{l-1}}^{k'+1} \leftarrow \mathrm{MKFHE.CTExt}(\bar{\mathbf{c}}'_{l-1,\mathrm{mult}}, K')$ and $\bar{\mathbf{c}}_{l-1,t} \in R_{q_{l-1}}^{k'+1} \leftarrow \mathrm{MKFHE.CTExt}(\mathbf{c}_{l,t}, K')$, which corresponds to the new joint secret key $\bar{\mathbf{s}}' \in R_3^{k'+1}$.

(2) Use Method 1 to generate a new relinearization key of subset $Y$, where $Y$ contains the parties $i$ and $j$. According to the auxiliary keys of parties $i$ and $j$, cloud server carries out a new auxiliary key $\mathbf{pb}_Y$ of the subset $Y$, where $\mathbf{pb}_Y = s_Y\mathbf{a}_{l-1} + \mathbf{e}_{l-1} \in R_{q_{l-1}}^d$ such that $s_Y = s_i s_j$. Perform the program $\mathrm{UniEnc}(s_Y; \mathbf{pb}_Y) \rightarrow \mathbf{D}_Y = [\mathbf{h}_0|h_1]$:

1) Sample $r_{l-1} \leftarrow \chi$.

2) Sample $\bar{\mathbf{e}}_{l-1,1} \leftarrow \psi^d$, and set $\mathbf{h}_0 = -\mathbf{pb}_Y + \bar{\mathbf{e}}_{l-1,1} + r_{l-1} \cdot \mathbf{g} (\mathrm{mod} q_{l-1})$.

3) Sample $\bar{e}_{l-1,2} \leftarrow \psi$, and set $h_1 = -r \cdot a_{l-1} + \bar{e}_{l-1,2} + s_Y (\mathrm{mod} q_{l-1})$.

Then, we get the relinearization key of subset $Y$ is $\mathbf{K}_{Y,t} = [k_{Y,t,0}|k_{Y,t,1}|k_{Y,t,2}] \in R_{q_{l-1}}^3$, where $k_{Y,t,0} = \langle \mathbf{g}^{-1}(b_{l-1,t}), \mathbf{h}_0 \rangle$, $k_{Y,t,1} = \langle \mathbf{g}^{-1}(b_{l-1,t}), \mathbf{a}_{l-1} \rangle$ and $k_{Y,t,2} = h_1$.

(3) For the tensor product $\mathbf{ct}'' = \bar{\mathbf{c}}'_{l-1,\mathrm{M}} \otimes \bar{\mathbf{c}}_{l-1,t}$, we can get the following result after relinearization.

$$
\begin{aligned}
&< \mathbf{ct}'', \bar{\mathbf{s}}' > (\mathrm{mod} q_{l-1}) = c''_0 + \textstyle\sum_{\tau=1}^{k} c''_\tau \cdot s_\tau \\
&= c'_{0,0} + \textstyle\sum_{\tau=1}^{k'-1} (c'_{\tau,0} + c_{0,\tau}) s_\tau \\
&+ \textstyle\sum_{|Y|,t=1}^{k'-1} \sum_{\zeta=0}^{d-1} \left(c'_{|Y|,t}\right)_\zeta \cdot \mathbf{K}_{Y,t} \cdot (1, s_Y, s_t) \ (\mathrm{mod} q_{l-1}) \\
&\approx c'_{0,0} + \textstyle\sum_{\tau=1}^{k'-1} (c'_{\tau,0} + c_{0,\tau}) s_\tau + \sum_{|Y|,t=1}^{k'-1} c'_{|Y|,t} s_Y s_t \\
&= < \bar{\mathbf{c}}'_{l-1,\mathrm{M}} \otimes \bar{\mathbf{c}}_{l-1,t}, \bar{\mathbf{s}}' \otimes \bar{\mathbf{s}}' > (\mathrm{mod} q_{l-1}) \\
&= \left( < \bar{\mathbf{c}}'_{l,\mathrm{mult}}, \bar{\mathbf{s}} > (\mathrm{mod} q_l) \right) \cdot \left( < \bar{\mathbf{c}}_{l-1,t}, \bar{\mathbf{s}}' > (\mathrm{mod} q_{l-1}) \right) \\
&\approx \mu_i \mu_j \cdot \mu_t
\end{aligned}
$$

Therefore, in the homomorphic operation process, when the party set updates, all the parties do not need to regenerate their keys and ciphertexts, so our scheme satisfies multi-hop.

**Table 2.** The time complexity and error size between our scheme and CDKS19 in cloud server.

| Relinearization types | | Time complexity | | Error size |
|---|---|---|---|---|
| | | Relinearization key generation | Homomorphic multiplication decryption | |
| CDKS19 | | $O(kd^2n)$ | $O(k^2d^3n)$ | $O\left(k^2n^2d^2\lceil\log q_l\rceil B^2\right)$ |
| Our scheme | Method 1 | $O(kdn)$ | $O(k^2d^2n)$ | $O\left(k^2n^2d^2\lceil\log q_l\rceil B^2\right)$ |
| | Method 2 | $O(kdn)$ | $O(k^2dn)$ | $O\left(((q_l\lceil\log Pq_l\rceil)/2P)k^2n^2dB^2\right)$ |

**Note.** Here, we only analyze the result of once homomorphic multiplication decryption by two parties. See Appendix A and B for a specific computation.

3. **Efficiency Analysis**
Our MKFHE scheme realized the leveled fully homomorphic computation without using the key-switching technology. As mentioned in Subsect. 3.3, our optimized relinearization algorithms are more efficient than that of CDKS19. So, we analyze the time complexity of our scheme in the cloud server here to show the advantages. See Table 2 for details.

As shown in Table 2, compared with the CDKS19 scheme, our MKFHE scheme takes less time to generate the relinearization key and decrypt the multiplication of ciphertexts, and the error size is also decreased. Therefore, our construction is more efficient.

## 5 Conclusion

In this paper, two optimized relinearization algorithms are proposed and applied to the design of the leveled BGV-type MKFHE schemes. Compared to the prior MKFHE schemes, our scheme uses the relinearization algorithm instead of the key-switching technology to re-linearize the ciphertext more efficiently. In the next research content, we will focus on how to break through the restriction of the number of parties on the homomorphic computation efficiency.

**Acknowledgments.** This work was supported by National Key R&D Program of China (Grant No. 2017YFB080 2000), National Natural Science Foundation of China (Grant Nos. U1636114, 61872384, 61872289), National Cryptography Development Fund of China (Grant No. MMJJ20170112).

## Appendix

### A. Time Complexity

In this appendix, we calculate the time complexity in Tables 1 and 2.

We define the time complexity as the scalar operation (addition or multiplication) of a polynomial, denoted as $\Delta$. For example, the time complexity of the product of two $n$-dimensional polynomials $a, b \in R_q$ is defined as $\Delta n$. Also, our optimized relinearization algorithm allows parties to generate their evaluation key $\mathbf{D}_i$ offline, so we do not calculate the time complexity of $\mathbf{D}_i$ any more.

**A1. Time Complexity Calculation in Table 1**

(1) For the CDKS19 scheme, the cloud server needs to perform $\left(2d^3 + d + con\right)$ (where *con* is a constant term) times polynomial multiplications to complete once relinearization, so the time complexity of once relinearization is about $O(d^3 n)$.

In the same way, we can calculate the time complexity of our optimized relinearization algorithms.

(2) For the Method 1, the cloud server needs to perform $\left(2d^2 + d + con\right)$ times polynomial multiplications to complete once relinearization, so the time complexity of once relinearization is about $O(d^2 n)$.

(3) For the Method 2, the cloud server needs to perform $(2d + con)$ times polynomial multiplications to complete once relinearization, so the time complexity of once relinearization is about $O(dn)$.

**A2. Time Complexity Calculation in Table 2**

(1) For the CDKS19 scheme

Every two parties need to perform $\left(2d^2 + con\right)$ times polynomial multiplications to generate a relinearization key. For $k$ parties, at least $\lfloor k/2 \rfloor$ relinearization keys need to be generated. So, the time complexity of generating all the relinearization keys is about $O(d^2 kn)$.

If the two parties decrypt the homomorphic multiplication of their ciphertexts successfully, the cloud server needs to perform $\left(2k^2 d^3 + k^2 d + k + con\right)$ times polynomial multiplications, so the time complexity is about $O(k^2 d^3 n)$.

(2) For the Method 1

Every two parties need to perform $(2d + con)$ times polynomial multiplications to generate a relinearization key. So, the time complexity of generating all the relinearization keys is about $O(kdn)$.

If the two parties decrypt the homomorphic multiplication of their ciphertexts successfully, the cloud server needs to perform $\left(2k^2 d^2 + k^2 d + k + con\right)$ times polynomial multiplications, so the time complexity is about $O(k^2 d^2 n)$.

(3) For the Method 2

Every two parties need to perform $(2d + con)$ times polynomial multiplications to generate a relinearization key. So, the time complexity of generating all the relinearization keys is about $O(kdn)$.

If the two parties decrypt the homomorphic multiplication of their ciphertexts successfully, the cloud server needs to perform $\left(2k^2 d + 2k^2 d + k + con\right)$ times polynomial multiplications, so the time complexity is about $O(k^2 d^2 n)$.

**B. Error Analysis**

Set the bound of $\psi$ and $\chi$ is $B$, so for the $a, b \in \psi$, such that $||a \cdot b||_\infty \leq nB^2$.

**1. For the Method 1**

As shown in Method 1 of Subsect.3.1, the error size of $\mathbf{K}_{i,j} \cdot \left(1, s_i, s_j\right)$ is as follows.

$$||e_{\text{small}}||_\infty = ||\mathbf{g}^{-1}(b_j)\mathbf{e}_1 + s_j e_2 + r_i e_j||_\infty \leq n\lceil \log q_l \rceil (d+1)B + n\lceil \log q_l \rceil B^2;$$

After relinearization and decryption, we can get the following results.

$$\begin{aligned} &< \overline{\mathbf{ct}'}, \bar{\mathbf{s}} >= c'_0 + \textstyle\sum_{i=1}^{k} c'_i \cdot s_i \\ &= c_{0,0} + \textstyle\sum_{i=1}^{k} (c_{i,0} + c_{0,i}) s_i + \sum_{i,j=1}^{k} \sum_{\varsigma=0}^{d-1} \left(c_{i,j}\right)_\varsigma \cdot \mathbf{K}_{i,j} \cdot \left(1, s_i, s_j\right) \pmod{q_l} \\ &= c_{0,0} + \textstyle\sum_{i=1}^{k} (c_{i,0} + c_{0,i}) s_i + \sum_{i,j=1}^{k} c_{i,j} \cdot s_i s_j + e_{\text{mult}} \end{aligned}$$

Therefore, after successful decryption, the final error size is as follows.

$$\begin{aligned} &||e_{\text{mult}}||_\infty = || \textstyle\sum_{i,j=1}^{k} \sum_{\varsigma=0}^{d-1} \left(c_{i,j}\right)_\varsigma \cdot e_{\text{small}}||_\infty \\ &= k^2 nd\left(n\lceil \log q_l \rceil (d+1)B + n\lceil \log q_l \rceil B^2\right) \leq O\left(k^2 n^2 d^2 \lceil \log q_l \rceil B^2\right) \end{aligned};$$

**2. For the Method 2**

As shown in Method 2 of Subsect.3.1, we have

$$||e'_{\text{small}}||_\infty = ||\left(\mathbf{g}^{-1}(b_j)\mathbf{e}_1 + r_i e_j + s_j e_2\right)||_\infty = n\lceil \log P \cdot q_l \rceil (dB + B^2 + B)$$

After relinearization and decryption, we can get the following results.

$$\begin{aligned} &||e'_{\text{mult}}||_\infty = || \textstyle\sum_{i,j=1}^{k} P^{-1} \cdot c_{i,j} e'_{\text{small}}||_\infty \\ &= ((q_l \lceil \log P \cdot q_l \rceil)/2P) k^2 n^2 (dB + B^2 + B) \leq O\left(((q_l \lceil \log P \cdot q_l \rceil)/2P) k^2 n^2 dB^2\right) \end{aligned}$$

Note. we usually choose the $P \succ q_l$, that is $P/q_l \approx 1$. So

$$O\left(((q_l \lceil \log P \cdot q_l \rceil)/2P) k^2 n^2 dB^2\right) \approx O\left(k^2 n^2 d \lceil \log q_l \rceil B^2\right).$$

**3. For the CDKS19**

As shown in Subsect. 2.5, the CDKS19 scheme completes once relinearization and decryption generating the error as follows.

$$||e_{\text{CDKS}}||_\infty = k^2 nd\left(n\lceil \log q_l \rceil (d+1)B + n\lceil \log q_l \rceil B^2\right) \leq O\left(k^2 n^2 d^2 \lceil \log q_l \rceil B^2\right).$$

## References


1. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multi-key fully homomorphic encryption. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, pp. 1219–1234. ACM (2012)
2. Qaosar, M., Zaman, A., Ssique, M.A., et al.: Privacy-preserving secure computation of skyline query in distributed multi-party databases. Information **10**(3), 119–135 (2019)
3. Xiong, J.B., Zhao, M.F., Bhuiyan, M., et al.: An AI-enabled three-party game framework for guaranteed data privacy in mobile edge crowdsensing of IoT. IEEE Trans. Industr. Inf. **17**(2), 922–933 (2021)
4. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ringbased public key cryptosystem. In: International Symposium on AlgorithmicNumber Theory, pp. 267–288 (1998)
5. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic AES evaluation using the modified LTV scheme. Des. Codes Crypt. **80**(2), 333–358 (2016)
6. Chongchitmate, W., Ostrovsky, R.: Circuit-private multi-key FHE. In: Fehr, S. (ed.) Public-Key Cryptography – PKC 2017. Lecture Notes in Computer Science, vol. 10175. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54388-7_9
7. Che, X.L., Zhou, T.P., Li, N.B., et al.: Modified multi-key fully homomorphic encryption based on NTRU cryptosystem without key-switching. Tsinghua Sci. Technol. **25**(5), 564–578 (2020)
8. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled FHE from learning with errors. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology – CRYPTO 2015. Lecture Notes in Computer Science, vol. 9216. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_31
9. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_26
10. Peikert, C., Shiehian, S.: Multi-key FHE from LWE, revisited. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 217–238. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_9
11. Brakerski, Z., Perlman, R.: Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. Lecture Notes in Computer Science, vol. 9814. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_8.
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_1
13. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. Lecture Notes in Computer Science, vol. 8042. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5
14. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 377–408. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_14
15. Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from TFHE. In: Galbraith, S., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. ASIACRYPT 2019. Lecture Notes in Computer Science, vol. 11922. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_16

16. Chen, L., Zhang, Z., Wang, X.: Batched Multi-hop multi-key FHE from ring-LWE with compact ciphertext extension. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography. TCC 2017. Lecture Notes in Computer Science, vol. 10678. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_20
17. Li, N.B., Zhou, T.P., Yang, X.Y., et al.: Efficient multi-key FHE with short extended ciphertexts and directed decryption protocol. IEEE Access **7**, 56724–56732 (2019)
18. Chen, H., Dai, W., Kim, M., et al.: 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, pp. 395–412. ACM, London (2019)
19. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, Cambridge, MA, USA, pp. 309–325 (2012)
20. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. CRYPTO 2014. Lecture Notes in Computer Science, vol. 8616. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_17
21. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
22. Li, B.Y., Micciancio, D.: On the security of homomorphic encryption on approximate. Cryptology ePrint Archive, Report 2020/1533 (2020). https://eprint.iacr.org/2020/1533
23. Cheon, J.H., Hong, S., Kim, D.: Remark on the security of CKKS scheme in practice. Cryptology ePrint Archive: Search Results 2020/1581 (2020). https://eprint.iacr.org/2020/1581
24. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM (JACM) **60**(6), 43 (2013)
25. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_3
26. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_50
27. Halevi, S., Polyakov, Y., Shoup, V.: An improved RNS variant of the BFV homomorphic encryption scheme. In: Matsui, M. (ed.) Topics in Cryptology – CT-RSA 2019. CT-RSA 2019. Lecture Notes in Computer Science, vol. 11405. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_5
28. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Proceedings of Annual Symposium on Foundations of Computer Science, Los Alamitos, CA, USA, pp. 97–106 (2011)