Homomorphic Encryption for Multiple Users with Less Communications*

Jeongeun Park

imec-COSIC, Department of Electrical Engineering, KU Leuven, Belgium
jeongeun.park@esat.kuleuven.be

Abstract. Keeping privacy for every entity in outsourced computation is always a crucial issue. For efficient secure computation, homomorphic encryption (HE) can be one of nice solutions. Especially, multikey homomorphic encryption (MKHE) which allows homomorphic evaluation on encrypted data under different keys can be one of the simplest solutions for a secure computation which handles multiple users' data. However, the current main problem of MKHE is that the dimension of its evaluated ciphertext relies on the number of users. To solve this problem, there are several variants of multikey homomorphic encryption schemes to keep the size of ciphertext constant for a fixed number of users. However, users interact one another before computation to provide their inputs, which increases setup complexity. Moreover, all the existing MKHE schemes and their variants have unique benefits which cannot be easily achieved at the same time in one scheme. In other words, each type of scheme has a suitable computational scenario to put its best performance.
In this paper, we suggest more efficient evaluation key generation algorithms (relinearization key and bootstrapping key) for the existing variants of MKHE schemes which have no ciphertext expansion for a fixed number of users. Our method only requires a very simple and minor pre-processing; distributing public keys, which is not counted as a round at all in many other applications. Regarding bootstrapping, we firstly provide an efficient bootstrapping for multiple users which is the same as the base single-key scheme thanks to our simplified key generation method without a communication. As a result, participants have less communication, computation, and memory cost in online phase. Moreover, we provide a practical conversion algorithm between the two types of schemes in order to *efficiently* utilize both schemes' advantages together in more various applications. We also provide detailed comparison among similar results so that users can choose a suitable scheme for their homomorphic encryption based application scenarios.

Keywords: Compact MKHE, Homomorphic Encryption, MKHE, Multikey Homomorphic Encryption

1 Introduction

Homomorphic encryption (HE) supports an operation on encrypted data. Therefore, it can be applied to an outsourced computation where a server computes a function value of client's encrypted data. Likewise, HE seems to be a proper solution for a computation keeping data privacy. Hence, it has been studied actively to enhance the efficiency for a practical use. As a result, many of homomorphic encryption schemes [CGGI17,FV12, BGV12] and approximate HE [CKKS17] are now considered practical due to their continuous improvement and optimization.

However, HE is not always a proper solution for any outsourced computation, specifically, handling multiple users' data. It is because HE schemes are originally defined for single user setting where only one key is involved in ciphertexts (we call them single-key HE throughout this paper). Instead, multikey HE (MKHE) [LTV12] which allows a computation on ciphertexts encrypted under *different* keys can be a solution for such case. Since each user encrypts its input by its key, each input is protected from other users for free. MKHE ciphertexts can be correctly decrypted only when the associated secret keys are gathered for semantic security. Therefore, MKHE may provide more advanced security by its nature for a computation among

* This is the full paper version of [Par21] appeared in IEEE Access, vol. 9, pp. 135915 - 135926

multiple users than single-key HE; input privacy and output privacy. We call it multikey security. Indeed, MKHE seems much more suitable in real application scenarios than just single-key HE. In real worlds, many of companies have their own private data which might be confidential or involved in patenting issue, they do not want to reveal the data to others. At the same time, all they need is to compute a function value with their data such as predicted value of a machine learning model. On the other hand, in this case, the input security cannot be naturally achieved by using single-key HE. If all the companies share the same key (using single-key HE), dishonest companies can decrypt other honest parties' inputs. Then the input security may not be fully achieved. In order to prevent this, one may assume or build a specific computing environment where the transmit channel between a user and the server is private, hence no one can obtain other's information without permission. This kind of assumption is somewhat strong so that it seems quite expensive to be instantiated in practice.

Even though MKHE schemes have been implemented with practical results [CDKS19, CCS19], the ciphertext expansion during homomorphic operation is still the main issue to be solved. That is, the ciphertext size grows relying on the number of users, which results in increasing both computation and communication cost proportional to the number of users (say k). Even though the size of ciphertext in MKHE is significantly reduced (from quadratic to linear on k) [BP16], the computation time and memory costs still at least k times more than underlying single-key scheme. This structure seems inevitable to keep on-the-fly property which is a strong benefit of *homomorphism* between ciphertext and plaintext. The property lets a new user join an ongoing computation at any point, freely. So no information about participants are not needed to be known ahead of time. This is specifically called dynamic computation which is useful in general, but the current problem of the schemes having the property is that the ciphertext size, more precisely, the dimension of ciphertext vector, grows depending on the number of users. For a certain computation scenario with pre-defined number of users (i.e when there is no new dynamic computation), it is not fully recommended to use such scheme sacrificing efficiency. In the case, the short ciphertext size and multikey security are essential for an efficient and secure computation among pre-defined users.

As a partial solution to achieve constant size of ciphertext, there are variants of MKHE schemes which only work when participants are fixed, by creating a common public key among pre-defined multiple parties. We categorize those all as a compact MKHE scheme throughout this paper since they achieve multikey security, and have no ciphertext expansion relying on the number of users. All existing approaches [AH19, MTPBH21] have the same structure based on threshold FHE (ThFHE) [AJLA+12] where more than t out of N parties can decrypt an evaluated ciphertext encrypted under N different secret keys (if $t = N$, it is called multikey variant Threshold FHE [BGG+17]). In threshold FHE based schemes, users generate a common public key, therefore, the asymptotic complexity of the scheme becomes as same as its base single-key HE scheme with multikey security. However, each has been built upon a particular computing scenario and their evaluation key generation procedure is a protocol by taking some interactions among users. In other words, users need to be online to generate such keys. Then homomorphic operation can be performed with those keys. Even, key switching and bootstrapping steps also require an interaction among users. Hence, such schemes have lost on-the-fly property to some extent. Especially, dynamic computation is hardly achievable.

Overall, the suggested two types of schemes (MKHE and multikey variant Threshold FHE) are the most suitable solutions for computations handling multiple users' data based on homomorphic encryption. However, they have very different advantages. And one is not simply compatible with the other. Furthermore, the suggested multikey-Threshold FHE schemes are not optimized in the sense of communication round complexity. In practice, it is always better to minimize the communication among multiple users for efficiency in terms of storage, computation, and more. Therefore, it is important to provide an efficient solution to take both advantages for more application scenarios.

1.1 Our Contribution

In order to solve the aforementioned issues, we first present efficient evaluation key generation algorithms (for relinearization and bootstrapping) for a compact MKHE scheme where the size of ciphertext does not grow depending on the number of users. The main goal is to generate evaluation keys by taking no interaction among pre-defined users. The procedure is not a protocol anymore so that the scheme becomes

more user-friendly by reducing computation, storage cost per user than other works [AH19, MTPBH21]. Regarding bootstrapping, we firstly provide an efficient bootstrapping for multiple users which is the same as the base single-key scheme thanks to our simplified key generation method without a communication. To do this, we need a simple pre-processing step, distributing public keys to all, which is not a big deal in realistic scenarios such as in public key infrastructure (PKI). We use the latest multikey variant of Threshold FHE scheme [MTPBH21] as a base scheme to apply our evaluation key generation algorithms. Also, our method is always applicable to every such ciphertext format encrypted under the secret key $(s_1 + s_2 + \cdots + s_k, 1)$, where s_i is the i-th user's secret.

Moreover, thanks to the minimal communication, we could suggest an efficient method to connect existing compact MKHE schemes and other (non-compact) MKHE schemes to maximize the performance in a various computation environment. For example, compact schemes have no ciphertext expansion for a fixed number of users, but on-the-fly computation is not directly applicable for new users. On the other hand, the non-compact schemes have ciphertext expansion but easily allow to join on-going computation at any point. In order to compensate each negative point, it is the best way to make one to be compatible with the other *efficiently*. In particular, the conversion from non-compact to compact scheme is very meaningful to reduce the network cost. The conversion is efficient since a server only runs key switching algorithm without requiring any interaction among users. Our algorithms are instantiated based on BFV homomorphic encryption scheme [FV12], but the same idea can be applied to other FHE schemes such as BGV [BGV12] and CKKS [CKKS17].

Main idea behind the contribution Existing compact MKHE schemes [AH19, MTPBH21] employ the interactive evaluation key generation algorithm [AJLA+12] for relinearization. In fact, after the tensor product of two multikey ciphertexts, ct_1 and ct_2 encrypting m_1 and m_2, respectively, under $\mathsf{sk} = (s_1 + s_2 + \cdots + s_k, 1)$, the resulting ciphertext $\mathsf{ct}_\otimes$ satisfies $\langle \mathsf{ct}_\otimes, \mathsf{sk} \otimes \mathsf{sk} \rangle = \mathsf{ct}_\otimes[1](s_1 + \cdots + s_k)^2 + \mathsf{ct}_\otimes[2](s_1 + \cdots + s_k) + \mathsf{ct}_\otimes[3](s_1 + \cdots + s_k) + \mathsf{ct}_\otimes[4] \approx \Delta m_1 m_2$. So $\mathsf{ct}_\otimes$ is a degree 2 ciphertext which should be reduced to a degree 1 ciphertext as the original ciphertexts in the end. To do this, one can homomorphically relinearize the degree 2 ciphertext and this method requires some additional key $\mathbf{K} \in \mathcal{R}_q^{\ell \times 2}$ such that $\mathbf{K}[1](s_1 + \cdots + s_k) + \mathbf{K}[2] = (s_1 + \cdots + s_k)^2 \mathbf{g} + \mathbf{e}$ for some error $\mathbf{e}$ and a gadget vector $\mathbf{g} \in \mathcal{R}_q^\ell$. Then relinearization performs $\mathsf{ct}_\times[1] = \langle \mathbf{g}^{-1}(\mathsf{ct}_\otimes[1]), \mathbf{K}[1] \rangle + \mathsf{ct}_\otimes[2] + \mathsf{ct}_\otimes[3]$, $\mathsf{ct}_\times[2] = \langle \mathbf{g}^{-1}(\mathsf{ct}_\otimes[1]\mathbf{K}[2]) \rangle + \mathsf{ct}_\otimes[4]$ to generate the final ciphertext $\mathsf{ct}_\times$ of degree 1. As a result, $\langle \mathsf{ct}_\otimes, \mathsf{sk} \otimes \mathsf{sk} \rangle \approx \langle \mathsf{ct}_\times, \mathsf{sk} \rangle$.

What [AJLA+12] based schemes are trying to do is to generate a pseudo encryptions of $(s_1 + \ldots + s_k)$, $\mathbf{K}$, under $(s_1 + \ldots + s_k)$ by combining all *single-key encryptions of* s_i *encrypted under its own key* s_i for each user $i \in [k]$ at the end. The way to transform a single-key ciphertext into multikey one takes some interaction, resulting in 2 rounds. Then a server collects the final information and adds them to obtain $\mathbf{K}$.

However, it becomes much simpler if each user generates a multikey encryption of s_i under a common public key from the beginning, since the public keys are already known. As a result, the essential number of interaction is reduced to one from two; distributing public keys which can be saved in a certain computation scenarios such as PKI. Then no interaction is required.

This solution reduces computational burden such as time memory, etc. on the user's side when such schemes are applied to real computation scenarios. More importantly, it strikes a better balance between two different types of schemes which have distinct benefit. Therefore, the existing compact MKHE schemes become simpler to be compatible with original MKHE schemes, which may provide more convenient computing tool for users.

1.2 Related Works

Multikey homomorphic encryption scheme is firstly introduced by López-Alt et al. [LTV12] based on NTRU. Then a multikey version of GSW scheme [CM15] was introduced for the first time. It was developed to achieve round optimal multiparty computation [MW16]. Also there were plausible results to get rid of CRS [BHP17, KLP20] among multiple users based on GSW scheme. Other works allowed multi-hop (dynamic) property [PS16, BP16] in MKHE schemes. Brakerski and Perlman [BP16] proposed the shorter size of ciphertext which has linear growth on the number of users at first. Then other FHE primitives were naturally

transformed to multikey variants such as multikey BGV [CZW17], multikey TFHE [CCS19, LP19], and multikey BFV, and CKKS [CDKS19] with linear growth ciphertext length.

In order to have constant size of ciphertext size (to achieve a compact MKHE), there is a multikey variant of Threshold FHE [AH19, MTPBH21], where a joint public keys are computed by users employing the idea of [AJLA+12] taking two rounds, hence the ciphertext size remains constant for a fixed number of users. The protocol takes four rounds in total. Due to their round complexity, it is not fully practical even in their specific computation scenarios; specifically, the work [AH19] is recommended to an outsourced computation with multiple model providers which are usually fixed and clients who can be anyone. Then the model provider uses multikey ThFHE and the clients use the original MKHE schemes for dynamic property. Mouchet et al. [MTPBH21] suggests their scheme for multiparty computation where multiple parties jointly compute a function value and computing materials such as evaluation keys. Again, each is still not fully practical for its best case since users need some interactions for homomorphic evaluation. There is a detailed survey about the line of works [AHSL20].

1.3 A note on compact MKHE

We define a compact MKHE (CMKHE) if an MKHE scheme or its variant supports that the ciphertext size remains constant for a polynomially many number of users. Hence, the asymptotic complexity of computation and communication of CMKHE schemes remains as same as single-key homomorphic encryption. Therefore, CMKHE can be an efficient solution for secure computations handling multiple users' data.

The first MKHE scheme [LTV12] is based on NTRU problem and it is a compact MKHE without creating a joint key among users. Moreover, it allows dynamic property directly as well. More importantly, the participants are not need to be pre-defined. However, the scheme is too inefficient due to its large parameters. Furthermore, it was broken by subfield attacks [ABD16, CJL16]. Hence, so far, the only method to instantiate a secure and efficient lattice based compact MKHE is based on Threshold FHE [AH19, MTPBH21], where users jointly generate a common public key. Due to a common public key, the size of ciphertext does not increase depending on the number of joint users. It is still an open problem to build a secure and efficient compact MKHE scheme without any interaction among users before computation.

2 Preliminaries

Notation: We let λ denote the security parameter. Vectors and matrices are denoted in lowercase bold and uppercase bold, respectively. We denote by $\langle \mathbf{v}, \mathbf{w} \rangle$ the dot product of two vectors $\mathbf{v}, \mathbf{w}$. For a vector $\mathbf{x}$, $\mathbf{x}[i]$ denotes the i-th component scalar. Let $\mathcal{R}$ and $\mathcal{R}_q$ denote $\mathbb{Z}[X]/(X^N+1)$ and $\mathbb{Z}[X]/(X^N+1) \mod q$, respectively, for positive integers q, N. For a real $\alpha > 0$, $\mathcal{D}_\alpha$ denotes the Gaussian distribution of standard deviation α. The key distribution χ is the uniform distribution over the set of binary polynomials. $\log(\cdot)$ is binary logarithm. We also denote the set $\{1, \ldots, n\}$ by $[n]$. If an element $a \approx 0 \mod p$ for some $a \in \mathcal{R}$, $\lfloor a - 0 \rceil = 0 \mod p$ for some modulus p. $[b]_q$ for $b \in \mathcal{R}$ means $b \mod q$. $U(\mathcal{R}_q)$ denotes uniform distribution over $\mathcal{R}_q$. We denote $\mathsf{Var}(a)$ is the maximum variance of each coefficient of $a \in R$. Then the variance of the product of two polynomials in R is that $\mathsf{Var}(ab) = n\,\mathsf{Var}(a)\,\mathsf{Var}(b)$ for $a, b \in R$. $\mathsf{Var}(\mathbf{a})$ is defined as $\sum_{i=1}^{\ell} \frac{1}{\ell}\mathbf{a}[i]$ for a polynomial vector $\mathbf{a} \in \mathcal{R}^\ell$. The error contained in a ciphertext $\mathsf{ct} \in \mathcal{R}^2$ is denoted by $\mathsf{Err}(\mathsf{ct})$. Let X and Y be two distributions over a finite domain. X $\overset{comp}{\approx}$ Y if they are computationally indistinguishable

2.1 Ring Learning With Errors

We recall the decisional ring learning with error (RLWE) problem which is a simple ring-based version of the LWE problem [Reg09].

Definition 1. *For security parameter λ, let $\Phi_m(X)$ be a cyclotomic polynomial with degree $\phi(m)$ and set $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$. Let $q = q(\lambda) \geq 2$ be an integer. For a fixed secret random element $s \in \mathcal{R}_q$ and a distribution $\chi = \chi(\lambda)$ over $\mathcal{R}$, the decisional RLWE problem says the distribution outputting $(a, [a \cdot s + e])$,*

where a and e is chosen uniformly at random from $\mathcal{R}_q$ and χ, respectively, and the distribution outputting (a, u) chosen uniformly at random from $\mathcal{R}_q^2$ are computationally indistinguishable.

The sample $(a, [a \cdot s + e]) \in \mathcal{R}_q^2$ is called RLWE sample. We assume the above problem for our scheme. For a simple and convenient use like many schemes, we set $\Phi(X) = X^N + 1$, where N is a power of 2.

2.2 BFV Homomorphic Encryption Scheme

We briefly introduce BFV homomorphic encryption scheme [Bra12, FV12]. We call the scheme as single-key BFV scheme throughout the paper. BFV scheme is derived from LPR [LPR10] public key encryption scheme. Let $\mathcal{R}_p$ be plaintext space, where $p > 1$ is plaintext modulus. We define $\Delta = \lfloor \frac{q}{p} \rceil$, where q is ciphertext modulus such that $q \gg p$. Let χ be a secret key distribution. Note that $\mathsf{EvalKeyGen}$ generates (public) evaluation keys used in evaluation algorithm Eval.

Definition 2. *BFV homomorphic encryption scheme is a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{EvalKeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$*:*

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$*: On input security parameter λ, it generates a key pair* pk, sk*:*
  - *Set* sk*: $s \leftarrow \chi$ over $\mathcal{R}_q$*
  - *Set* pk*: $(a, b = (-as + e)) \mod q$, where $a \leftarrow \mathcal{R}_q$ uniformly at random, and $e \leftarrow \mathcal{D}_\alpha$.*
- $\mathsf{EvalKeyGen}(\mathsf{sk}, T) \to \mathsf{evk}$ *: Set* $\mathsf{sk} = s$*. For $i \in \{0, \ldots, \ell = \lfloor \log_T q \rfloor\}$,* $\mathsf{evk} = (a_i, -a_i s + e_i + T^i s^2) \mod q$
- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$*: For the message $m \in \mathcal{R}_t$, and public key* $\mathsf{pk} = (a, b)$*, it outputs a fresh ciphertext* $\mathsf{ct} = (ua + e_1, ub + \Delta m + e_2) \mod q \in \mathcal{R}_q^2$*, where $u \leftarrow \chi$ uniformly at random and $e_1, e_2 \leftarrow \mathcal{D}_\alpha$.*
- $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}) \to m$*: Let ciphertext* $\mathsf{ct} = (c_0, c_1)$ *and* $\mathsf{sk} = s$*, it outputs $m = \lfloor \frac{p}{q}[c_0 s + c_1]_q \rceil \mod p$.*
- $\mathsf{Eval}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk})$*:*
  - $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2) \to \mathsf{ct}_+$*: Output* $\mathsf{ct}_+ = (\mathsf{ct}_1[1] + \mathsf{ct}_2[1], \mathsf{ct}_1[2] + \mathsf{ct}_2[2]) \mod q \in \mathcal{R}_q^2$*.*
  - $\mathsf{Mult}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{evk}) \to \mathsf{ct}_\times$*:*
    (1) Compute $\lfloor \frac{t}{q} \mathsf{ct}_1 \otimes \mathsf{ct}_2 \rceil = (c_1, c_2, c_3, c_4)$*. Set* $\mathsf{ct}_\otimes = (c_1', c_2', c_3')$*, where* $c_1' = c_1, c_2' = c_2 + c_3, c_3' = c_4$*.*
    (2) Let $\mathsf{evk}_i[1] = a_i, \mathsf{evk}_i[2] = -a_i s + e_i + T^i s^2 \mod q$ *for $i \in [\ell]$, where T is the base of a gadget vector.*
    Compute $\mathsf{ct}_\times[1] = c_2' + \sum_{i=0}^{\ell} \mathsf{evk}_i[1] c_1'^{(i)}$*,*
    $\mathsf{ct}_\times[2] = c_3' + \sum_{i=0}^{\ell} \mathsf{evk}_i[2] c_1'^{(i)}$*, where* $c_1' = \sum_{i=0}^{\ell} T^i \cdot c_1'^{(i)}$*.*
    (3) Output $\mathsf{ct}_\times = (\mathsf{ct}_\times[1], \mathsf{ct}_\times[2])$*.*

A canonical RLWE based ciphertext like BFV ciphertext is a form of RLWE sample, which is, $(a, b) \in \mathcal{R}_q^2$. Homomorphic multiplication over this kinds of ciphertext contains computing tensor product of two ciphertexts, in general. So the size of ciphertext grows to square and there is non-linear element in the result. To fix the problems, a procedure called relinearization is performed after tensor product (step (2) of Mult), which requires additional evaluation keys.

2.3 Basic Scheme for Multikey variant of Threshold FHE based on BFV

We recall a multikey ThFHE scheme based on BFV [MTPBH21] since we improve the evaluation key generation algorithm of existing compact MKHE schemes. As we mentioned in Section 1, the existing such works are instantiated multikey-threshold homomorphic encryption. So we choose this scheme [MTPBH21] to apply our new technique. Let p and q be a plaintext modulus and a ciphertext modulus, respectively, and $q \gg p$. $\Delta = \lfloor q/p \rceil$ as defined in original BFV.

- $\mathsf{CMKHE.Setup}(1^\lambda) \to \mathsf{params}$: It takes security parameter λ and outputs public parameter $\mathsf{params} = \{N, \chi, p, q, \alpha, a\}$, where N is a polynomial degree, χ is a secret key distribution, α is a standard deviation of Gaussian distribution and $a \in \mathcal{R}_q$ is CRS.
- $\mathsf{CMKHE.KeyGen}(\mathsf{params}) \to (\mathsf{sk}, \mathsf{pk})$: Given params,
  - set a secret key $\mathsf{sk} = s \in \mathcal{R}_q$, where $s \leftarrow \chi$ over $\mathcal{R}_q$
  - set a public key $\mathsf{pk} = (a, b) \in \mathcal{R}_q^2$, where $b \leftarrow (-as + e)$ and $e \leftarrow D_\alpha$ over $\mathcal{R}_q$.

- Outputs a key pair $(\mathsf{sk}, \mathsf{pk})$.

After Key generation, all users share their public keys through public channel. Then parties generate a common public key by adding the second component of all given public keys.

- $\mathsf{CMKHE}.\mathsf{ComKey}(\mathsf{pk}_1, \ldots, \mathsf{pk}_k) \to \hat{\mathsf{pk}}$: Given all parties' public keys ($\mathsf{pk}_i = (a, b_i)$, it outputs a common public key $\hat{\mathsf{pk}} := (a, \sum_{i=1}^{k} b_i)$ such that $a(\sum_{i=1}^{k} s_i) + b \approx 0 \mod q$.
- $\mathsf{CMKHE}.\mathsf{Enc}(\hat{\mathsf{pk}}), m) \to \mathsf{ct}$: It takes a common public key $\hat{\mathsf{pk}}$ and a message $m \in \mathcal{R}_p$ on input, it outputs a fresh ciphertext ct.
  - parse $\hat{\mathsf{pk}} := (a', b')$ such that $a(\sum_{i=1}^{k} s_i) + b \approx 0 \mod q$.
  - samples $u \leftarrow \chi, e_1, e_2 \leftarrow D_\alpha$
  - computes a fresh ciphertext $\mathsf{ct}_i = (ua' + e_1, ub' + e_2 + \Delta m) \in \mathcal{R}_q^2$.
  - Outputs ct_i.
- $\mathsf{CMKHE}.\mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_k, \mathsf{ct}) \to m$: On input all users' secret keys $\{\mathsf{sk}_i = s_i\}_{i\in[k]}$ and a ciphertext ct,
  - Set $\hat{\mathsf{sk}} = (s_1 + \cdots + s_k, 1) \in \mathcal{R}_q^2$, where each s_i from sk_i.
  - Compute $\frac{1}{\Delta}\langle \mathsf{ct}, \hat{\mathsf{sk}} \rangle \approx m \mod p$

As we can see, a fresh ciphertext is encrypted under all involved users' keys. Therefore, the public keys of different users are simply added with a common randomness. More precisely, ct_i is decrypted with all associated secret key elements $s_1, \ldots, s_k$, that is, $\frac{1}{\Delta}\langle \mathsf{ct}_i, (s_1 + \cdots + s_k, 1) \rangle = \mathsf{ct}_i[1] \cdot (s_1 + \cdots + s_k) + \mathsf{ct}_i[2] = m + e_c$, where e_c is an error, s_i is a secret element of i-th user. The ciphertext space is just as same as the space of single-key BFV, working as a single-key homomorphic encryption.

2.4 Multikey Homomorphic Encryption based on BFV

The existing multikey BFV scheme [CDKS19] is a generalization of the above single-key BFV scheme. They let a server expand a single-key BFV ciphertext using other public materials given by all users. That is, each user generates key pair independently and encrypts each message only with its own public key, hence users do not need to know one another before computation. Therefore, its key generation and encryption algorithm are just as same as single-key BFV scheme. However, the size of evaluated ciphertext grows linearly in the number of users.

For example, A fresh ciphertext encrypting m_i of the i-th user is $\mathsf{ct}_i = (c_1, c_2) \in \mathcal{R}_q^2$ and it is decrypted with a corresponding secret key $\mathsf{sk}_i = (s_i, 1)$ as $\frac{p}{q}\langle \mathsf{ct}_i, \mathsf{sk}_i \rangle = c_1 s_i + c_2 \approx m_i \mod p$. Each operation (either addition or multiplication) with other user's fresh ciphertext adds one more component in a ciphertext i.e. $\mathsf{ct}' = (c_1, c_2, c_3) \in \mathcal{R}_q^3$, if there are two users. More precisely, if we add two fresh ciphertexts ct_1 and ct_2 of two parties, first rearrange the ciphertext entries by increasing the dimension of input ciphertexts such as $\mathsf{ct}_1' = (\mathsf{ct}_1[1], 0, \mathsf{ct}_1[2]), \mathsf{ct}_2' = (0, \mathsf{ct}_2[1], \mathsf{ct}_2[2])$ then their underlying secret key sk is $(s_1, s_2, 1)$. For addition, just add *the two expanded ciphertexts*, i.e. $\mathsf{ct}_+ := \mathsf{ct}_1' + \mathsf{ct}_2' \mod q \in \mathcal{R}_q^3$. For multiplication, tensor product is used between two expended ciphertexts as single-key BFV does. We first do tensor product which outputs $\mathsf{ct}_\otimes = \mathsf{ct}_1' \otimes \mathsf{ct}_2' \in \mathcal{R}_q^9$. Then ct satisfies $\langle \mathsf{ct}_\otimes, \mathsf{sk} \otimes \mathsf{sk} \rangle = \langle \mathsf{ct}_1', \mathsf{sk} \rangle \cdot \langle \mathsf{ct}_2', \mathsf{sk} \rangle$. We then relinearize it by using additional material (evaluation keys) since there is some non-linear parts such as $s_1 \cdot s_2$. Then the final evaluated ciphertext $\mathsf{ct}_\times \in \mathcal{R}_q^3$ satisfies $\frac{q}{p}\langle \mathsf{ct}_\times, \mathsf{sk} \rangle \approx m_1 m_2 \mod p$

After computation with k users, the evaluated ciphertext is $\hat{\mathsf{ct}} = (c_1, \ldots, c_{k+1}) \in \mathcal{R}_q^{k+1}$, which is decrypted as $\frac{p}{q}\langle \hat{\mathsf{ct}}, \hat{\mathsf{sk}} \rangle \approx f(m_1, \ldots, m_k) \mod p$, where $\hat{\mathsf{sk}} = (s_1, \ldots, s_k, 1)$ and $f(m_1, \ldots, m_k)$ is a desired function value of all users.

3 Main Technique

In this section, we present an efficient evaluation key (also bootstrapping key) generation algorithm for compact MKHE scheme to be round optimal. We assume that a common random string (CRS) is given to all users from a trusted generator in the following scheme.

3.1 Evaluation key generation algorithm

We recall a gadget vector which is $\mathbf{g} = (g_1, \ldots, g_\ell) \in \mathcal{R}_q^\ell$ (usually, $g_i = g^{i-1}$ for some $g \in \mathbb{Z}_q$). The gadget vector is widely used in homomorphic encryption schemes to control noise growth since the gadget decomposition (inverse of $\mathbf{g}$, $\mathbf{g}^{-1}$) keeps the size of output less than $\min\{g_i\}$. Furthermore, if $\mathbf{g}^{-1}$ is randomized (output follows subguassian distribution), it gives a tight bound for error analysis due to independent error term [AP14, GMP19, JLP21]. Note that $\mathbf{g}^{-1} : \mathcal{R}_q \to \mathcal{R}^\ell$, which outputs $\mathbf{g}^{-1}(a) = (u_1, \ldots, u_\ell)$ for $a \in \mathcal{R}_q$, such that $g_1u_1 + \ldots + g_\ell u_\ell = a \mod q$, $\ell = O(\log_g q)$.

We first define our simple evaluation key generation algorithm for relinearization. This process is simple and no much noise is contained in the output keys. Once joint parties' public keys are known, no interaction is required for generating a ciphertext and evaluation keys among participants, which makes our scheme more practical than other similar approaches [AJLA+12, AH19, MTPBH21].

– $\mathsf{CMKHE.EvalKeyGen}(\hat{\mathsf{pk}}, s_i) \to \mathbf{K}_i \in \mathcal{R}_q^{\ell \times 2}$:
  (1) $\mathbf{u}_i \leftarrow \chi^\ell$ over $\mathcal{R}_q$, $\mathbf{e}_i, \mathbf{e}_i' \leftarrow D_\alpha^\ell$ over $\mathcal{R}_q$.
  (2) Parse $\hat{\mathsf{pk}} = (a, b)$ such that $a(\sum_{i=1}^k s_i) + b \approx 0 \mod q$.
  (3) $\mathbf{K}_i[1] = a \cdot \mathbf{u}_i + s_i \cdot \mathbf{g} + \mathbf{e}_i \mod q$,
  $\mathbf{K}_i[2] = b \cdot \mathbf{u}_i + \mathbf{e}_i' = -a(\sum_{t=1}^k s_t) \cdot \mathbf{u}_i + \tilde{\mathbf{e}}_i \in \mathcal{R}_q^\ell \mod q$.

Let $\mathbf{K}_i$ be an evaluation key of i-th user. Then

$$\mathbf{K}_i[1] \cdot \sum_{t=1}^k s_t + \mathbf{K}_i[2] = s_i(\sum_{t=1}^k s_t) \cdot \mathbf{g} + \hat{\mathbf{e}}_i \mod q, \tag{1}$$

for some error vector $\hat{\mathbf{e}}_i \in \mathcal{R}_q^\ell$. Note that a of the common public key $\hat{\mathsf{pk}}$ is the same common random parameter given by a trusted setup in this section.

In fact, the keys are pseudo-encryptions of secret key. Hence we assume circular security as general homomorphic encryption schemes do. The keys are securely encrypted by multiple users' secret keys, which we will discuss in A by proving this pseudo-encryptions of secrets are computationally indistinguishable to uniform random elements by RLWE assumption. Therefore, any secret key information is not leaked at all unless all of the users agree on it.

Now we define our evaluation algorithm consisting of addition and multiplication over ciphertexts. We note that a server can use the master evaluation key by adding all given individual evaluation keys from k users to save memory and computation cost. Then the evaluation algorithm is just as same as the one of the underlying single-key FHE scheme. Let $\hat{\mathbf{K}} := \sum_{t=1}^k \mathbf{K}_t$ be the master evaluation key. Let $\mathsf{ct}_i = (c_{i,1}, c_{i,2}) \in \mathcal{R}_q^2$, $\mathsf{ct}_j = (c_{j,1}, c_{j,2}) \in \mathcal{R}_q^2$ be two multikey ciphertexts encrypted under k different keys, and m_i, m_j be a corresponding plaintext respectively, for $i, j \in [k]$ (i and j are possibly same).

– $\mathsf{CMKHE.Eval}(C, \mathsf{ct}_i, \mathsf{ct}_j, \hat{\mathbf{K}}) \in \{\mathsf{ct}_+, \mathsf{ct}_\times\}$, C is a circuit.
  • [**Addition**] $\mathsf{CMKHE.Add}(\mathsf{ct}_i, \mathsf{ct}_j) \to \mathsf{ct}_+ \in \mathcal{R}_q^2$:
    (1) $\mathsf{ct}_+ = \mathsf{ct}_i + \mathsf{ct}_j \mod q = (c_{i,1} + c_{j,1}, c_{i,2} + c_{j,2}) \mod q$.
  • [**Multiplication**] $\mathsf{CMKHE.Mult}(\mathsf{ct}_i, \mathsf{ct}_j, \hat{\mathbf{K}}) \to \mathsf{ct}_\times \in \mathcal{R}_q^2$:
    (1) $\mathsf{ct}' = \mathsf{ct}_i \otimes \mathsf{ct}_j \in \mathcal{R}^4$.
    (2) Compute $c_v = [\lfloor \frac{1}{\Delta} \mathsf{ct}'[v] \rceil]_q$ for $v \in [4]$.
    (3) $\mathsf{ct}_\times[1] = \langle \mathbf{g}^{-1}(c_1), \hat{\mathbf{K}}[1] \rangle + (c_2 + c_3) \mod q$,
    $\mathsf{ct}_\times[2] = \langle \mathbf{g}^{-1}(c_1), \hat{\mathbf{K}}[2] \rangle + c_4 \mod q$.

After the first step (tensor product), the output ct' satisfies $\langle \mathsf{ct}', \mathsf{sk} \otimes \mathsf{sk} \rangle = \langle \mathsf{ct}_i, \mathsf{sk} \rangle \cdot \langle \mathsf{ct}_j, \mathsf{sk} \rangle$, where $\mathsf{sk} = (s_1 + \cdots + s_k, 1)$. Then $\frac{1}{\Delta}[\mathsf{ct}'[1](s_1 + \cdots + s_k)^2 + (\mathsf{ct}[2] + \mathsf{ct}[3])(s_1 + \cdots + s_k) + \mathsf{ct}'[4]] \approx m_i m_j$. Therefore, the dimension of ct' is increased and the ciphertext contains some non-linear parts such as $s_i \cdot s_j$. So we use relinearization algorithm to reduce the dimension as the original one. The third step of the multiplication is relinearization. The computation complexity is as same as single-key BFV since there is no ciphertext expansion.

Correctness of evaluation: Let $\hat{\mathsf{sk}} = (s_1 + \cdots + s_k, 1) \in \mathcal{R}_q^2$ be a corresponding secret key of a multikey ciphertext. For addition, we can easily check that

$$\begin{aligned}[\langle \mathsf{ct}_+, \hat{\mathsf{sk}} \rangle]_q &= [\mathsf{ct}_+[1](s_1 + \cdots + s_k) + \mathsf{ct}_+[2]]_q \\ &\approx \Delta(m_i + m_j) \mod q.\end{aligned}$$

For multiplication, we first note that ct' satisfies the following equation with a secret key $\hat{\mathsf{sk}} \otimes \hat{\mathsf{sk}}$:

$$\frac{1}{\Delta}\left[c_1 \cdot (\sum_{i=1}^{k} s_i)^2 + (c_2 + c_3) \cdot \sum_{i=1}^{k} s_i + c_4\right]_q \approx m_i m_j \mod p$$

After relinearization (step (3)) with evaluation keys, it satisfies that

$$\begin{aligned}&\langle \mathsf{ct}_\times, \hat{\mathsf{sk}} \rangle \mod q = \mathsf{ct}_\times[1](s_1 + \cdots + s_k) + \mathsf{ct}_\times[2] \mod q \\ &= \langle \mathbf{g}^{-1}(c_1), \hat{\mathbf{K}}[1] \rangle \cdot (\sum_{i=1}^{k} s_i) + (c_2 + c_3) \cdot (\sum_{i=1}^{k} s_i) \\ &+ \langle \mathbf{g}^{-1}(c_1), \hat{\mathbf{K}}[2] \rangle + c_4 \mod q \\ &= \left[\langle \mathbf{g}^{-1}(c_1), \hat{\mathbf{K}}[1] \cdot (\sum_{i=1}^{k} s_i) + \hat{\mathbf{K}}[2] \rangle + (c_2 + c_3) \cdot \sum_{i=1}^{k} s_i + c_4\right]_q \\ &= \left[\langle \mathbf{g}^{-1}(c_1), (s_1 + \cdots + s_k)^2 \cdot \mathbf{g} + \mathbf{e} \rangle + (c_2 + c_3) \cdot \sum_{i=1}^{k} s_i + c_4\right]_q \\ &= \left[(s_1 + \cdots + s_k)^2 \langle \mathbf{g}^{-1}(c_1), \mathbf{g} \rangle + (c_2 + c_3) \cdot \sum_{i=1}^{k} s_i + c_4 + \tilde{e}\right]_q \\ &\approx \left[c_1 \cdot (s_1 + \cdots + s_k)^2 + (c_2 + c_3) \cdot (s_1 + \cdots + s_k) + c_4\right]_q \\ &\approx \Delta m_i m_j \mod q,\end{aligned}$$

where $\mathbf{e} \in \mathcal{R}_q^\ell$ and $\tilde{e} \in \mathcal{R}_q$ are some errors.

3.2 Bootstrapping for CMKHE

Bootstrapping of standard FHE is not straightforward to do and even requires large size of additional key materials (encryptions of *secret key*). Moreover, a server needs much larger storage and takes more computation time for MKHE scheme (at least k times more than its single-key scheme). Even though it seems inevitable to take all key elements from each user, we can reduce the computation time of a server at least since we use compact MKHE.

We follow the bootstrapping of [CDKS19], but ours is much simpler since the size is reduced to single-key version. Moreover, there is no interaction among users for bootstrapping, even creating a bootstrapping key unlike [MTPBH21]. Everything we need to modify is evaluation of Galois automorphisms which is a linear transformation requiring rotations on encrypted elements since the rest is as same as an improved single-key BFV bootstrapping procedure [CH18]. It is mainly based on key switching such as using a map $\tau_t : P(X) \mapsto P(X^t)$, where $P(X)$ is a polynomial in $\mathcal{R}_q$. We denote $\tau_t(\mathbf{a}) = (\tau_t(a_1), \ldots, \tau_t(a_d))$ for a d-dimensional vector $\mathbf{a} = (a_1, \ldots, a_d)$. It is easy to see that $\tau_t(\mathsf{ct})$ is encrypting $\tau_t(m)$ under a secret key $\tau_t(\mathsf{sk})$, where $\mathsf{CMKHE.Dec}(\mathsf{ct}, \mathsf{sk}) = m, \mathsf{sk} = (s, 1)$. To bootstrap an evaluated multikey ciphertext, it requires secret key elements, say, $\mu_s \in \mathcal{R}_q$, which is generated by additional algorithm only for bootstrapping key in [CDKS19]. We first generate an key generation algorithm which takes a secret key element on input. Let $\hat{\mathsf{pk}}$ be a common public key of k users.

– $\mathsf{CMKHE.EncSKGen}(\hat{\mathsf{pk}}, \mu_s) \to \mathsf{key} \in \mathcal{R}_q^{\ell \times 2}$
  (1) $\mathbf{u} \leftarrow \chi^\ell$ over $\mathcal{R}_q$, $\mathbf{e}, \mathbf{e}' \leftarrow D_\alpha^\ell$ over $\mathcal{R}_q$.
  (2) Parse $\hat{\mathsf{pk}} = (a', b')$ such that $a'(\sum_{i=1}^k s_i) + b' \approx 0 \mod q$
  (3) $\mathsf{key}[1] = a' \cdot \mathbf{u} + \mathbf{e} \mod q$,
  $\mathsf{key}[2] = b' \cdot \mathbf{u} + \mu_s \cdot \mathbf{g} + \mathbf{e}' = -a(\sum_{i=1}^k s_i) \cdot \mathbf{u} + \mu_s \cdot \mathbf{g} + \tilde{\mathbf{e}} \in \mathcal{R}_q^\ell \mod q$, for some error $\tilde{\mathbf{e}} \in \mathcal{R}_q^\ell$.

We can easily check key satisfies the following equation:

$$\mathsf{key}[1] \cdot (\sum_{i=1}^k s_i) + \mathsf{key}[2] \approx \mu_s \cdot \mathbf{g} \mod q \in \mathcal{R}_q^{\ell \times 2}$$

Then one may transform a ciphertext encrypted under s to an encryption of the same plaintext under different secret key s'. Then we use key switching algorithm similarly defined in [CDKS19] but more efficient than that since we only require one 2ℓ ring multiplication rather than $2k\ell$. Let ct be a ciphertext encrypted under s and the key is encrypting s' under the secret s.

– $\mathsf{CMKHE.KeySwitch}(\mathsf{ct}, \mathsf{key}) \to \mathsf{ct}' \in \mathcal{R}_q^2$
  • Compute $\mathsf{ct}' = (\mathsf{ct}'[1], \mathsf{ct}'[2])$ as
    ∗ $\mathsf{ct}'[1] = \langle \mathbf{g}^{-1}(\mathsf{ct}[1]), \mathsf{key}[1] \rangle \mod q$
    ∗ $\mathsf{ct}'[2] = \langle \mathbf{g}^{-1}(\mathsf{ct}[1]), \mathsf{key}[2] \rangle + \mathsf{ct}[2] \mod q$

Then, in our case, bootstrapping key (also called Galois key) is generated by running $\mathsf{CMKHE.EncSKGen}$ algorithm as [CDKS19]. We define Galois key generation algorithm below.

– $\mathsf{CMKHE.GkGen}(\hat{\mathsf{pk}}, \tau_t(s_i)) \to \mathbf{GK}_{i,t}$, for $i \in [k], t \in \mathbb{Z}_{2N}^*$
  (1) Runs $\mathsf{CMKHE.EncSKGen}(\hat{\mathsf{pk}}, \tau_t(s_i)) \to \mathsf{key}_{i,t}$
  (2) Outputs $\mathbf{GK}_{i,t} := \mathsf{key}_{i,t}$

In fact, like the master evaluation key, a server can save the master bootstrapping key by adding all bootstrapping keys of users. It can save memory consumption of a server definitely. An evaluation of Galois automorphism on a compact MKHE ciphertext is easily done by running key switching algorithm $\mathsf{CMKHE.KeySwitch}$ taking bootstrapping key on input. For simplicity and efficiency, we use $\hat{\mathbf{GK}}_t := \sum_{i=1}^k \mathbf{GK}_{i,t}$ as the master bootstrapping key. Now we check the correctness of key switching with the master bootstrapping key.
Correctness: We show that $\langle \mathsf{ct}', \hat{\mathsf{sk}} \rangle = \tau_t(\langle \mathsf{ct}, \hat{\mathsf{sk}} \rangle) \mod q$, where $\hat{\mathsf{sk}} = (s_1 + \cdots + s_k, 1)$.

$$\begin{aligned}
\langle \mathsf{ct}', \hat{\mathsf{sk}} \rangle &= \mathsf{ct}'[1](s_1 + \cdots + s_k) + \mathsf{ct}'[2] \\
&= \langle \mathbf{g}^{-1}(\tau_t(\mathsf{ct}[1])), \hat{\mathbf{GK}}_t[1] \cdot \sum_{i=1}^k s_i \rangle + \langle \mathbf{g}^{-1}(\tau_t(\mathsf{ct}[1])), \hat{\mathbf{GK}}_t[2] \rangle \\
&+ \tau_t(\mathsf{ct}[2]) \mod q \\
&= \left[\langle \mathbf{g}^{-1}(\tau_t(\mathsf{ct}[1])), \hat{\mathbf{GK}}_t[1] \cdot \sum_{i=1}^k s_i + \hat{\mathbf{GK}}_t[2] \rangle + \tau_t(\mathsf{ct}[2]) \right]_q \\
&\approx \left[\langle \mathbf{g}^{-1}(\tau_t(\mathsf{ct}[1])), (\tau_t(s_1) + \cdots + \tau_t(s_k)) \cdot \mathbf{g} + e \rangle + \tau_t(\mathsf{ct}[2]) \right]_q \\
&\approx \tau_t(\mathsf{ct}[1]) \cdot (\tau_t(s_1) + \cdots + \tau_t(s_k)) + \tau_t(\mathsf{ct}[2]) \mod q \\
&= \langle \tau_t(\mathsf{ct}), \tau_t(\hat{\mathsf{sk}}) \rangle \mod q \\
&= \tau_t(\langle \mathsf{ct}, \hat{\mathsf{sk}} \rangle) \mod q,
\end{aligned}$$

where $e \in \mathcal{R}_q$ is an error.

3.3 Security

We can prove that the distribution of our evaluation keys are computationally indistinguishable to uniform distribution, since the evaluation keys are only different to existing secure constructions [AH19, MTPBH21]. We define the $\mathsf{CMKHE.EvalKeyGen}$ (resp. $\mathsf{CMKHE.EncSKGen}$)is an encryption algorithm of message m under a public key generated from $\mathsf{CMKHE.KeyGen}$. We prove that the encryption algorithm is semantically secure, then equivalently we conclude that the evaluation keys do not leak any information about underlying message (secret key) against passive adversaries by assuming circular security.

For MKHE security with multiple decryptor, it is recommended to prove that the scheme is secure against both internal adversary and external adversaries [LP19]. However, we only care of the internal adversary which is one of participants for evaluation keys, since the keys are not used for decryption process. We show the entire security proof of the proposed scheme in Appendix C with the brief explanation about proper adversaries.

Theorem 1. *The distribution of evaluation keys are computationally indistinguishable to uniform distribution over $\mathcal{R}_q^{\ell\times 2}$ by assuming underlying RLWE assumption with the parameter (n, q, χ, D_α) is hard and circular security.*

Proof. Let k be the number of users, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{CMKHE.KeyGen}\,(\mathsf{params})$, where $\mathsf{params} \leftarrow \mathsf{CMKHE.Setup}$ for $i \in [k]$, and we set $\mathsf{pk}_i = (a_i, b_i), \mathsf{sk}_i = s_i$. Let $\mathbf{K}_b \leftarrow \mathsf{CMKHE.EvalKeyGen}(\hat{\mathsf{pk}}, b)$ be a relinearization key encrypting a bit b under k users' secrets. Let $\mathsf{key}_b \leftarrow \mathsf{CMKHE.EncSKGen}(\hat{\mathsf{pk}}, b)$ be a bootstrapping key of encryption a bit b under k user's keys. Without loss of generality, we assume $k = 2$. Here, the challenge ciphertext is an evaluation key $\mathbf{K}_b$ (resp. key_b). We denote an internal adversary of CMKHE by $\mathcal{A}$ and an RLWE distinguisher by $\mathcal{B}$. In the security game for internal adversary [KLP20], $\mathcal{A}$ performs what he can do such as accessing certain oracle after receiving a challenge ciphertext $\mathbf{K}_b$ (resp. key_b) from a challenger of a security game, sends the value to $\mathcal{B}$ by letting him guess which b was chosen by the challenger. Then $\mathcal{A}$ forwards the answer of $\mathcal{B}$ to the challenger.

It is clear to see that the challenge ciphertext $\mathbf{K}_b$ (resp. key_b) encrypting b under pk_1 and pk_2. Then the internal adversary who holds sk_2 (assume he is the second user), can partially decrypt $\mathbf{K}_b$ (resp. key_b) with its key by performing $\langle(\mathbf{K}_b[1, j], \mathbf{K}_{2,j}), \mathsf{sk}_2\rangle$ for some $j \in [ell]$, resulting in an element $au_j s_1 + b \cdot s_2 \cdot g_j + e_j \in \mathcal{R}_q$ denoted by b_j. Now we can see that (a, b_j) is an RLWE sample under the given parameter. Then the adversary sends b_1 with a which is a uniform random element and pk_1 to $\mathcal{B}$. In other words, the advantage of $\mathcal{A}$ becomes the advantage of the adversary $\mathcal{B}$. which is negligible by RLWE assumption with the given parameter.

Therefore, we conclude that the encryption scheme for evaluation keys are IND-CPA secure assuming circular security and that RLWE assumption is hard with such parameter set. Equivalently, the keys are computationally indistinguishable to $\mathcal{R}_q^{\ell\times 2}$, hence do not leak any information about secret key through public channel against passive adversary. □

4 Compatibility with Existing Schemes

Table 1. Pros and Cons of existing CMKHE and original MKHE

	Existing CMKHE	MKHE
`Pros`	No ciphertext expansion	Dynamic computation for free No restriction for the number of joint users before computation
`Cons`	Dynamic computation takes some rounds Users should be pre-defined	Ciphertext size grows linearly in the number of users

Compact MKHE schemes we present above are as efficient as its base single-key HE schemes. However, they are not always better than other existing schemes. In order to show the different ability of MKHE

schemes and their variants depending on computing environment, we denote our scheme by CMKHE and a multikey BFV scheme [CDKS19] by MKHE in this section,.

We let the key generation of MKHE be same as CMKHE's in the presence of CRS. We summarize the properties of each scheme which may affect on a computation in a positive or negative way in Table 1. It is interesting to see that the benefit that one scheme posses is what the other scheme cannot achieve directly. That is to say, in all existing CMKHE schemes, there is no ciphertext expansion for pre-defined users, but dynamic computation is hardly achievable directly. On the other hand, MKHE allows dynamic computation for free but the ciphertext size grows linearly in the number of users.

In fact, it would be the best if the dynamic computation is possible between different schemes to achieve the best performance using each benefit. Therefore, we suggest practical methods to convert one scheme into the other to compensate the negative point of a scheme by the positive point of the other to some extent.

- The conversion from MKHE to CMKHE is meaningful if a server has lack of its computation power at some point or no more new users join the computation. Moreover, it can be used to reduce communication cost by a server without any interaction with users after evaluation a function value.
- We recommend the opposite direction (from CMKHE to MKHE) for a case that new users often take part in a computation after computing a joint function value among a fixed number of users.

We note that these conversion algorithms are also applicable to existing compact MKHE schemes [AH19, MTPBH21] since they have basically the same structure but it is preferable to use ours due to lower round complexity.

We only consider the issue on adding new users in the paper since we can remove users by using multikey proxy re-encryption [YKHK18] regardless of type of schemes. If a user wants to stop continuing the computation, we have to make sure that anything about security issue can not happen. To do this, there is no other efficient solution than multikey proxy re-encryption so far. With all users' proxy re-encryption keys, the server partially decrypt the output then sends to the user. And user can finally decrypt the output with its own key, while the server can continue further computation with the rest of users' information.

4.1 Conversion CMKHE into MKHE

The conversion from CMKHE to MKHE is pretty straightforward. Just add a new component for a new user by extending ciphertext dimension. To do this, however, users' evaluation keys of MKHE should be previously generated and sent to a server(or a computing party). For example, let $\mathsf{ct} = (a, b) \in \mathcal{R}^2$ be an CMKHE ciphertext encrypting m under secret $s_1 + s_2$ and let $\mathsf{ct}_3 = (a_3, b_3)$ be a MKHE ciphertext encrypting m_3 under s_3. The server expands ct to $\mathsf{ct}' = (a, 0, b)$ and ct_3 to $\mathsf{ct}'_3 = (0, a_3, b_3)$, of which underlying secret key is $(s_1 + s_2, s_3, 1)$. Then a server computes a function f over ct' and ct'_3 by using $\mathsf{MKHE.Eval}$ algorithm with evaluation keys of MKHE, i.e., addition and multiplication are performed as MKHE's evaluation procedure, then the output is $\mathsf{ct}' = (a_1, a_2, b) \in \mathcal{R}^3$ such that $b + a_1(s_1 + s_2) + a_2 s_3 \approx \Delta f(m, m_3)$.

We note that this conversion is actually used in [AH19] to collaborate both schemes by giving a particular example (between a model owners who use a compact MKHE scheme and users who are dynamically join the computation by using MKHE). We say that this conversion can be extended to more general cases with less communication among users.

If a group of new users are joining at the same time, it would be better to extend only one component for the new group of which information is encrypted by using CMKHE than adding every component according to the number of new users. Then run $\mathsf{MKHE.Eval}$ algorithm between two groups (the old and the new). If new users unexpectedly join the computation, it is better to use MKHE is used from then.

4.2 Conversion MKHE into CMKHE

Before going to the conversion, we note that it is the most efficient way to use our scheme among any other CMKHE schemes such as [AH19, MTPBH21] for the conversion because it takes no round for that. Server can do everything with previously given information.

We assume a CRS for large number of k and that public keys are known to one another. Let k users have jointly computed a function value $f(m_1, \ldots, m_k)$ based on MKHE, where the resulting evaluated MKHE ciphertext $\mathsf{ct} \in \mathcal{R}_q^{k+1}$ which satisfies $\frac{1}{\Delta}[\langle \mathsf{ct}, (s_1, \ldots, s_k, 1)\rangle]_q \approx f(m_1, \ldots, m_k) \mod p$. A server converts it to CMKHE ciphertext for a further computation with a function g with the same joint users. Let MKHE pubic key of each user be $(a, b_i) \in \mathcal{R}_q^2$, and secret key be s_i for $i \in [k]$, where a is CRS. Then the protocol between k users and a server is following:

- **Setup**: Before computation, users share their public keys to create a common public key $\hat{\mathsf{pk}}$ then generate conversion keys $\mathbf{CK}_i$ (for $i \in [k]$) by running $\mathsf{ConvKeyGen}$ which can be precomputed. Then sends them to a server.
- **Server**:
  - runs $\mathsf{MKHEtoCMKHE}(\mathsf{ct}, \{\mathbf{CK}_i\}_{i\in[k]},)$

The setup phase can be considered as pre-processing in many of applications. Now, we define the conversion key generation algorithm $\mathsf{ConvKeyGen}$ and the conversion algorithm $\mathsf{MKHEtoCMKHE}$. It is basically, running similar process in key switching $\mathsf{CMKHE.KeySwitch}$ algorithm with conversion keys of k users.

- $\mathsf{ConvKeyGen}(\hat{\mathsf{pk}}, s_i) \to \mathbf{CK}_i \in \mathcal{R}_q^{\ell\times 2}$
  - Run $\mathsf{CMKHE.EncSKGen}(\hat{\mathsf{pk}}, s_i) \to \mathsf{key}$
  - Output $\mathbf{CK}_i := \mathsf{key}$

- $\mathsf{MKHEtoCMKHE}(\mathsf{ct}, \{\mathbf{CK}_i\}_{i\in[k]}) \to \mathsf{ct}' \in \mathcal{R}_q^2$
  - Parse $\mathsf{ct} = (a_1, a_2, \ldots, a_k, b)$.
  - $\mathsf{ct}'[1] = \sum_{i=1}^k \langle \mathbf{g}^{-1}(a_i), \mathbf{CK}_i[1]\rangle \mod q$
  - $\mathsf{ct}'[2] = \sum_{i=1}^k \langle \mathbf{g}^{-1}(a_i), \mathbf{CK}_i[2]\rangle + b \mod q$

Correctness: We only check that $\frac{1}{\Delta}[\langle \mathsf{ct}', (\sum_{i=1}^k s_i, 1)\rangle]_q \approx f(m_1 \ldots, m_k)$

$$
\begin{aligned}
[\langle \mathsf{ct}', (\sum_{i=1}^k s_i, 1)\rangle]_q &= [\sum_{i\in[k]} \langle \mathbf{g}^{-1}(a_i), \mathbf{CK}_i[1]\rangle (\sum_{i=1}^k s_i) \\
&+ \sum_{i\in[k]} \langle \mathbf{g}^{-1}(a_i), \mathbf{CK}_i[2]\rangle + b]_q \\
&= [\sum_{i\in[k]} \langle \mathbf{g}^{-1}(a_i), \mathbf{CK}_i[1](\sum_{i=1}^k s_i)\rangle + \sum_{i\in[k]} \langle \mathbf{g}^{-1}(a_i), \mathbf{CK}_i[2]\rangle + b]_q \\
&= [\sum_{i\in[k]} \langle \mathbf{g}^{-1}(a_i), \mathbf{CK}_i[1](\sum_{i=1}^k s_i) + \mathbf{CK}_i[2]\rangle + b]_q \\
&\approx [\sum_{i\in[k]} (\langle \mathbf{g}^{-1}(a_i), s_i\mathbf{g}\rangle + b]_q \\
&\approx [\sum_{i\in[k]} a_i s_i + b]_q \\
&\approx \Delta f(m_1, \ldots, m_k) \mod q
\end{aligned}
$$

We note that the conversion key's security is also guaranteed as we discussed of the security of the evaluation keys in Section 3.

Table 2. Complexity comparison between our work and other similar works. We let k be the number of users, n be a degree of ciphertext polynomial, ℓ be evaluation key parameter, σ_g be the variance of the gadget decomposition algorithm.

		Relin noise	EvalKeyGen p user	$\lvert$evk$\rvert$	Pre-processing	# interaction	# interaction in PKI	Setup
MKHE [CDKS19]	w/o pre	$\tilde{O}(k^2n^2\ell\sigma_g)$	$\tilde{O}(\ell)$	$\tilde{O}(k\ell)$	–	0	0	NO
	w pre	$\tilde{O}(k^2n^2\ell^2\sigma_g^2)$	$\tilde{O}(\ell)$	$\tilde{O}(k^2\ell)$	$\tilde{O}(k^2\ell^2)$	0	0	YES
CMKHE	[MTPBH21]	$\tilde{O}(k^2n^2\ell\sigma_g)$	$\tilde{O}(k\ell)$	$\tilde{O}(\ell)$	–	2	1	YES
	Ours	$\tilde{O}(k^2n^2\ell\sigma_g)$	$\tilde{O}(\ell)$	$\tilde{O}(\ell)$	–	1	0	YES

4.3 Complexity comparison with similar works

We show the complexity comparison between multikey [CDKS19] scheme and compact MKHE schemes ([MTPBH21] and ours) in Table 2 based on what we mentioned in this section. Since both [MTPBH21] and [AH19] (as model owners side) schemes basically have same structure but [MTPBH21] improved the noise growth, so we compare [MTPBH21] as a representative previous compact MKHE scheme. We also show two kinds of MKHE [CDKS19] in the presence of pre-processing. They suggest pre-processing model to improve the time cost of relinearization by generating a common evaluation key as a setup process. Therefore, the number of users should be pre-defined in this case. As a trade-off, their storage for keys increased quadratically in k, compared to $O(k)$ in others.

Relin noise denotes the complexity of noise growth after relinearization with evaluation keys. EvalKeyGen p user denotes the complexity of evaluation key generation algorithm per user. $\lvert$evk$\rvert$ denotes the evaluation key (relinearization) key size which a server stores for homomorphic multiplication. Pre-processing denotes the server's computation complexity for combining evaluation keys to generate a common relinearization key before computation starts. And we show how many interactions among users for homomorphic operation are needed in general and in PKI model where public keys are stored in advance, respectively. The last column shows that if the users are required to know each other to use each scheme. We let k be the number of users.

We first look into the noise after relinearization. The compact MKHE schemes have the same k^2 factor in noise complexity as original non-compact MKHE scheme [CDKS19]. In compact MKHE schemes, the sum of k independent noise of distinct public keys are contained in a common public keys, hence, this factor influences in each evaluation key. Since the common evaluation key is the sum of k users' evaluation keys, total k^2 factors are inevitable in this construction. Regarding the noise, MKHE with pre-processing has the large complexity since they deal with large size of key sets. We show our noise analysis in Appendix A.

Then we compare the complexity of the evaluation key generation algorithm for each user in every scheme. Interactive protocols take k evaluation key materials in each round, whereas each user in MKHE scheme and ours scheme only generates its evaluation key without interaction. More precisely, in [MTPBH21] (also in [AH19]), every user generates an evaluation key material in the first round and publishes all to every user. After that, users can generate the final material by combining previously given materials and publish them to all. After gathering all final information, a common evaluation key is then generated. Therefore, they have k factor in the computation complexity since they are dealing with k information in every round, whereas users in our scheme only generate a common evaluation key on their own without interaction if public keys are known. As a result, previous CMKHE schemes [MTPBH21, AH19] takes 2 rounds for homomorphic operations and our work needs one round; distributing public keys. If we assume a public key infrastructure (PKI) as a pre-processing model, we have no round at all, whereas [MTPBH21, AH19] still require one round to generate a common evaluation key by combining additional materials.

Users in MKHE with pre-processing generate its public key and evaluation key on their own and send them to a server (or a computing party). Then the server computes a shared evaluation key by taking non-negligible time due to its large key size comparing to relatively negligible time in other CMKHE schemes. Also the server uses large storage due to large size of keys with k^2 factor. MKHE without pre-processing does not generate a shared relinearization key so a server stores all k evaluation keys of users. However, a server only stores a common evaluation key of CMKHE schemes, which occupies the smallest storage.

Consequently, we present the most efficient compact MKHE scheme for arithmetic operation until now. Moreover, in practice, our scheme may outperform than MKHE with pre-processing in terms of time, noise, memory, and network cost since it anyway requires pre-defined users, even though ours needs one round interaction among users. If a computation keeps adding new users' information, it is recommended to use practical MKHE schemes [CCS19, CDKS19] since the conversion from CMKHE to MKHE is a non-negligible time algorithm.

5 Conclusion

We propose an efficient evaluation key generation method for compact MKHE schemes where the size of ciphertext does not depend on the number of users. Hence, no further interaction is needed for homomorphic evaluation. Moreover, we suggest a practical conversion algorithm to efficiently convert non-compact one to compact one and the inverse direction in order to utilize unique advantages of both schemes together. In other words, compact MKHE scheme is as efficient as single-key preserving individual input privacy for free, but dynamic computation is not easily achievable since it only works for pre-defined number of users. On the other hand, original non-compact scheme allows new users to join on-going computation freely, but having ciphertext expansion. As a result, we provide multiple choices to users for their practical secure computation.

Acknowledgement. This work was supported by CyberSecurity Research Flanders with reference number VR20192203

References

ABD16. Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 153–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

AH19. Asma Aloufi and Peizhao Hu. Collaborative homomorphic computation on data encrypted under multiple keys. arXiv, 2019.

AHSL20. Asma Aloufi, Peizhao Hu, Yongsoo Song, and Kristin Lauter. Computing blindfolded on data homomorphically encrypted under multiple keys: An extended survey, 2020.

AJLA+12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 483–501. Springer, 2012.

AP14. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

BGG+17. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. Cryptology ePrint Archive, Report 2017/956, 2017. https://eprint.iacr.org/2017/956.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS '12*, 2012.

BHP17. Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *Theory of Cryptography Conference*, pages 645–677. Springer, 2017.

BP16. Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 190–213, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from tfhe. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 446–472, Cham, 2019. Springer International Publishing.

CDKS19. Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 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, page 395–412, New York, NY, USA, 2019. Association for Computing Machinery.

CGGI17. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 377–408, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

CH18. Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved FHE bootstrapping. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 315–337, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

CJL16. Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero. *LMS Journal of Computation and Mathematics*, 19(A):255–266, 2016.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

CM15. Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled fhe from learning with errors. In *Annual Cryptology Conference*, pages 630–656. Springer, 2015.

CZW17. Long Chen, Zhenfeng Zhang, and Xueqing Wang. Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 597–627, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

GMP19. Nicholas Genise, Daniele Micciancio, and Yuriy Polyakov. Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 655–684, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

JLP21. Sohyun Jeon, Hyang-Sook Lee, and Jeongeun Park. Efficient lattice gadget decomposition algorithm with bounded uniform distribution. *IEEE Access*, 9:17429–17437, 2021.

KLP20. Eunkyung Kim, Hyang-Sook Lee, and Jeongeun Park. Towards round-optimal secure multiparty computations: Multikey fhe without a crs. *International Journal of Foundations of Computer Science*, 31(02):157–174, 2020.

LP19. Hyang-Sook Lee and Jeongeun Park. On the security of multikey homomorphic encryption. In Martin Albrecht, editor, *Cryptography and Coding*, pages 236–251, Cham, 2019. Springer International Publishing.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

LTV12. Adriana LópezAlt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

MTPBH21. Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies*, 2021(4):291–311, 2021.

MW16. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 735–763, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

Par21. Jeongeun Park. Homomorphic encryption for multiple users with less communications. *IEEE Access*, 9:135915–135926, 2021.

PS16. Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 217–238, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

YKHK18. Satoshi Yasuda, Yoshihiro Koseki, Ryo Hiromasa, and Yutaka Kawai. Multi-key homomorphic proxy re-encryption. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *Information Security*, pages 328–346, Cham, 2018. Springer International Publishing.

A Noise analysis on evaluation

We can provide an average-case noise analysis on variance of the noise instead of the use of other measures such as infinity norm if we use a heuristic assumption that all the coefficients of the noise *independently* follows subgaussian distribution or we use a real subgaussian sampling [JLP21, GMP19] for $\mathbf{g}^{-1}(\cdot)$. We define the secret key is sampled the uniform distribution over the set of binary polynomials.

Since other algorithms are as same as previous works, we only look into the noise contained in evaluation key first. As denoted in this section, the noise contained in $\hat{\mathbf{K}}_i$ is $\hat{\mathbf{e}}_i$. $\hat{\mathbf{e}}_i = (\sum_{t=1}^{k} s_t)\mathbf{e}_i + e_{\mathsf{pk}}\mathbf{u}_i + \mathbf{e}'_i$, where e_{pk} is an error contained in a public key. Therefore,

$$\mathsf{Var}(\mathsf{Err}(\mathbf{K}_i)) = \mathsf{Var}(\hat{\mathbf{e}}_i) \approx kn\alpha^2 + n\alpha^2 + \alpha^2 = (kn + n + 1)\alpha^2.$$

Hence, the variance of the noise in the common evaluation key is $\mathsf{Var}(\mathsf{Err}(\mathbf{K})) \approx k(nk + n + 1)\alpha^2$. Comparing to other work [MTPBH21] which significantly reduces the output noise in evaluation key as opposed to the technique in [AJLA+12] and its extension [AH19], we have similar (a bit smaller) noise in evaluation keys.

Now we observe the error after relinearization. As we can see that, $\langle \mathsf{ct}_\times, \mathsf{sk}\rangle = \langle \mathbf{g}^{-1}(c_1), \hat{\mathbf{K}}[1](\sum_{i=1}^{k} s_i) + \hat{\mathbf{K}}[2]\rangle + (c_2 + c_3)(\sum_{i=1}^{k} s_i) + c_4 = \langle \mathsf{ct}', \mathsf{sk} \otimes \mathsf{sk}\rangle + \langle \mathbf{g}^{-1}(c_1), \mathsf{Err}(\hat{\mathbf{K}})\rangle$, where $\mathsf{sk} = (\sum_{i=1}^{k} s_i, 1)$. The relinearization error is $\langle \mathbf{g}^{-1}(c_1), \mathsf{Err}(\hat{\mathbf{K}})\rangle$ The variance of this noise is following:

$$\mathsf{Var}_{\mathsf{Relin}}(\mathsf{ct}_\times) = n\ell\, \mathsf{Var}(\mathbf{g}^{-1}(c_1)) \cdot \mathsf{Var}(\mathsf{Err}(\hat{\mathbf{K}}))$$

As a result, compact MKHE schemes have the same k^2 factor in noise complexity as original non-compact MKHE scheme [CDKS19]. In compact MKHE schemes, the sum of k independent noise of distinct public keys are contained in a common public keys, hence, this factor influences in each evaluation key. Since the common evaluation key is the sum of k users' evaluation keys, total k^2 factors are inevitable in this construction.

B Different Types of Decryption

We can use several proposed decryption algorithms depending on processing environment. In single decryptor setting, where the trusted party holds all the involved secret keys, the one can simply run $\mathsf{CMKHE.Dec}$ algorithm as [LTV12].

In the multi decryptor setting, where all involved users jointly decrypt a common ciphertext, the decryption is a protocol, more precisely, distributed decryption protocol taking one round. The protocol consists of two algorithms $\mathsf{PartDec}$, FinDec run by each user. The FinDec is originally proposed as a public algorithm of which all inputs are public elements, but it may leak output information to the one who watch the public channel and run the public algorithm on its own, hence, it is modified to an algorithm requiring a valid secret key as an input to achieve output privacy of multikey ciphertext [LP19].

We can use all the existing solutions for different cases, hence, introduce the methods to decrypt evaluated ciphertexts of our scheme:

(1) If the output privacy is not concerned such as standard MPC protocol or users employ their own secure channel, they can use the one round distributed decryption protocol [CCS19] with noise flooding technique.

(2) To achieve the output privacy through public channels, users use a decryption protocol suggested in [LP19], with re-encrypting the output of PartDec. It requires more communication cost than (1).
(3) As proposed in [YKHK18], a server can run partial decryption algorithm on behalf of each user using re-encryption key given in advance, hence, there is no communication among users at decryption phase, requiring large storage and computation time of a server instead.

C Security of proposed Scheme

We first recall static adversaries of multikey homomorphic encryption schemes [LP19]. There are two types of adversaries; the internal adversary who is a member of joint users, and the external adversary who does not take part in a computation but can see what is transmitted through a public channel.

The input security is achieved by the multikey IND-CPA security [KLP20] which models the same adversary as IND-CPA security except that the adversary (internal adversary) can perform the decryption by its own secret key before and after challenge phase (also with challenge message). The output security models adversaries who is an adversary of IND-CPA security except that he (external adversary) can additionally access the partial decryption oracle for each party' secret key before and after challenge step (also with challenge message) [LP19].

she can obtain the output but wishes to learn other users' inputs. The external adversary is not taking part in a computation but able to see users' transmit channel hoping to learn both input and output.

Therefore, if a scheme considers a single decryptor, it only focus on internal adversaries for security. For multi decryptor setting, a scheme should be secure against both internal and external adversaries. We show that our scheme is semantically secure in the multi decryptor setting to consider both adversaries. We follow the security game and the proof of [LP19, KLP20]

Theorem 2. *The scheme* CMKHE *with multi decryptor is semantically secure if underlying single-key BFV is semantically secure under RLWE assumption.*

Proof. Let k be the number of users, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{CMKHE.KeyGen}(\mathsf{params})$, where $\mathsf{params} \leftarrow \mathsf{CMKHE.Setup}$ for $i \in [k]$, and we set $\mathsf{pk}_i = (a_i, b_i), \mathsf{sk}_i = s_i$. ct be a multikey ciphertext under the common public key pk. Let $\{\mathbf{K}_i \leftarrow \mathsf{CMKHE.EvalKeyGen}(\hat{\mathsf{pk}}, s_i)\}$ be evaluation keys of i-th user. Let $\mathsf{key}_i \leftarrow \mathsf{CMKHE.EncSKGen}(\hat{\mathsf{pk}}, s_i)$ be bootstrapping key of each user. Without loss of generality, we assume $k = 2$. We denote any kind of adversary of CMKHE by $\mathcal{A}$ and the adversary which breaks underlying BFV scheme by $\mathcal{B}$. In each security game per each type of adversary [KLP20, LP19], $\mathcal{A}$ performs what he can do such as accessing certain oracle after receiving a challenge ciphertext ct_b from a challenger of a security game, sends the value to $\mathcal{B}$ by letting him guess which b was chosen by the challenger. Then $\mathcal{A}$ forwards the answer of $\mathcal{B}$ to the challenger. We note that $\mathcal{A}$ and $\mathcal{B}$ cannot learn anything from $\{\mathbf{K}_i\}$ and $\{\mathsf{key}_i\}$ by RLWE assumption as we proved in 3.

1. For input security against an internal adversary, it is clear to see that the challenge ciphertext ct_b encrypting m_b under pk_1 and pk_2. Then the internal adversary who holds sk_2 (assume he is the second user), can partially decrypt ct_b with its key by performing $\langle \mathsf{ct}_b, \mathsf{sk}_2 \rangle$, resulting in a ciphertext ct'_b which is a BFV ciphertext encrypted under sk_1. Then the adversary sends ct'_b and pk_1 to an adversary who breaks BFV scheme. In other words, the advantage of $\mathcal{A}$ becomes the advantage of BFV adversary $\mathcal{B}$ which is negligible by RLWE assumption.
2. For output security against the external adversary $\mathcal{A}$ which does not have any associated secret keys, we consider the decryption of [LP19] consisting of two algorithms PartDec and FinDec. For more detail, we focus on other transmitted information, partial decryption which is the output of PartDec. Our PartDec partially decrypts a multikey ciphertext with its own key then re-encrypt the result with BFV scheme by the receiver's public key. Recall the security game of [LP19], the challenge ciphertext ct_b is encrypted under $\mathsf{sk}_1, \mathsf{sk}_2$, encrypting m_b. The adversary can access to partial decryption oracle to obtain partial decryption information d_1 and d_2, which is a BFV encryption under sk_1 and sk_2, respectively. Then $\mathcal{A}$ forwards d_1, d_2 and $\mathsf{pk}_1, \mathsf{pk}_2$ to the BFV adversary $\mathcal{B}$. Since the advantage of $\mathcal{B}$ is negligible due to RLWE assumption, so is $\mathcal{A}$'s.

CMKHE is semantically secure since it is CPA-secure against both internal adversaries and external adversaries, if single-key BFV is CPA-secure under RLWE assumption.

□