# A Unified Framework of Homomorphic Encryption for Multiple Parties with Non-Interactive Setup

Hyesun Kwak[1], Dongwon Lee[1], Yongsoo Song[1], and Sameer Wagh[2]

Seoul National University
{hskwak, dongwonlee95, y.song}@snu.ac.kr
Devron Corporation
sameer@devron.ai

**Abstract.** The standard Homomorphic Encryption (HE) poses an authority issue when multiple parties are involved, as the authority is concentrated solely to whom that owns the (single) secret key. To solve this issue, variants of HE have emerged in the context of multiple parties, resulting in the development of two different lines of HE schemes – Multi-Party HE (MPHE) and Multi-Key HE (MKHE). MPHE schemes tend to be much more efficient; but require the interaction between parties in the key generation and the set of parties is fixed throughout the entire evaluation. On the other hand, MKHE schemes have poor scaling with the number of parties but allow us to add new parties to the joint computation anytime.

In this work, we construct the first MPHE scheme that features a non-interactive key generation. We refactor the evaluation key to be nearly linear, allowing it to be computed by simple summation. As a result, our MPHE allows each party to independently and asynchronously broadcasts its key.

In addition, we propose a new HE primitive, called Multi-Group HE (MGHE). Stated informally, an MGHE scheme provides seamless integration between MPHE and MKHE, and combines the best of both these primitives. In an MGHE scheme, a group of parties generates a public key jointly which results in compact ciphertexts and efficient homomorphic operations, similar to MPHE. However, unlike MPHE, it also supports computations on encrypted data under different keys, a property enjoyed by MKHE schemes. We present a construction of MGHE from the BFV scheme and provide a proof-of-concept implementation to demonstrate its concrete performance.

Finally, we describe a general approach to construct a multi-party protocol from MGHE. We provide a proof-of-concept implementation of a logistic regression model where our MGHE interpolates between MPHE (where the training is performed) and MKHE (where the inference is performed).

**Keywords:** Multi-Key Homomorphic encryption, Multi-Party Homomorphic Encryption

## 1 Introduction

Homomorphic Encryption (HE) enables computation over encrypted data without decryption. It prevents the leakage of private information while evaluating data within an untrusted environment. However, HE typically requires a large resource even when it computes a simple arithmetic operation such as multiplication. As a result, HE is particularly well-suited for implementation in cloud systems that can supply large computing power for evaluation.

A typical HE only supports computations between data encrypted *by the same key.* Consequently, when multiple data owners are involved, it relies on a trusted third party who possesses a key distributed to each party for encryption. Still, this merely transfers the trust problem from the cloud service provider to the new third party and thus does not provide an acceptable solution to this problem. To overcome this challenge, extensive research has explored the use of distributed trust in designing HE schemes involving multiple parties.

In the context of multiple parties, two important lines of HE schemes have emerged: Threshold HE and Multi-Key HE (MKHE). In Threshold HE [5, 9, 38, 40, 37], multiple parties collaborate to generate a joint public key, and encryption is performed under this joint key. Threshold HE has a $t$-out-of-$n$ ($t \leq n$) access structure where any $t$ parties can reconstruct the secret key to decrypt the ciphertext. Studies on Threshold HE are again diverged into two different directions: the case where $t < n$ and the case

where $t = n$. In our work, we focus on the case when $t = n$, which we is referred to as Multi-Party HE (MPHE). Like any other Threhold HE schemes, MPHE is comparable to that of the single-key HE schemes since encryption and homomorphic computation are performed in a similar manner. However, the set of participants should be determined beforehand and fixed in the preparation phase and no other parties can join the computation in the middle. Moreover, the existing MPHE schemes are based on a multi-round key generation protocol in which the involved parties should interact with each other.

On the other hand, MKHE [35, 22, 39, 41, 14, 15] features a distributed setup phase where each party independently generates its own key pair, without requiring any information about other participants. The encryption can be done by a single key, and it allows to perform arithmetic operations on ciphertexts that do not necessarily have to be encrypted under the same key. The main advantage of MKHE lies in its flexibility: it is not necessary to pre-determine the list of participants or the computational task. From the performance perspective, however, the size of ciphertexts increases with the number of involved parties, and so does the complexity of homomorphic operations.

### 1.1 Our Contributions

**Construction of MPHE with Non-Interactive Setup.** We present the first construction of an MPHE scheme with fully non-interactive setup based on the BFV [10, 25] and CKKS [19] schemes. To the best of our knowledge, all known MPHE schemes [5, 38, 40] require parties to interact with each other in order to generate a common evaluation key with non-linear structure. We present a *nearly linear* key generation algorithm where the joint encryption and evaluation keys are obtained from independently generated individual keys by simply summing them.

**Formalization of Multi-Group HE.** We propose a novel variant of HE designed for multiple parties, called Multi-Group HE (MGHE), and define its security notion. An MGHE scheme can be viewed as a generalization of both MPHE and MKHE, which enjoys the best of both primitives. In MGHE, a group of parties collaboratively generates a public key that is commonly used among the parties for encryption. Hence, MGHE behaves like an MPHE scheme in a single group. Moreover, an MGHE scheme has the capability to perform arbitrary computations on encrypted data, regardless of whether the input ciphertexts are encrypted under the same group key or not, a crucial property of MKHE.

**Construction of MGHE.** We extend our MPHE scheme to construct an MGHE scheme and provide a rigorous proof of its semantic security. Our MGHE scheme regards an MPHE ciphertext as a single-key encryption under the joint secret key so that ciphertexts corresponding to different group keys can be operated in a multi-key manner. Consequently, our MGHE scheme has a hierarchical structure where a ciphertext is decryptable by the joint secret keys of the associated groups, each of which is additively shared among the group members.

**Building Multi-Party Computation Protocol from MGHE.** We build a round-optimal Multi-Party Computation (MPC) protocol on top of our MGHE scheme, which is naturally derived from the non-interactive key generation (setup). We show that the protocol is secure against semi-malicious adversaries in the dishonest majority setting, relying on the semantic security of MGHE.

**Experimental results.** We implement our MGHE scheme based on both BFV and CKKS and present its concrete performance. We provide basic benchmark and demonstrate a logistic regression model as a proof-of-concept.

### 1.2 Technical Overview

At the heart of our construction lies a non-interactive key generation algorithm. This allows the joint key of a group to be constructed non-interactively from independently generated keys of the group members. The key generation follows a hybrid construction between MPHE (the encryption key aspects) and MKHE (the relinearization mechanisms). We begin by giving a high-level overview of our MPHE construction.

We assume that each party is identified as a unique index $i$ and let $I$ be a group of parties. The homomorphic property of LWE makes the summation of public and secret key pairs be a valid key

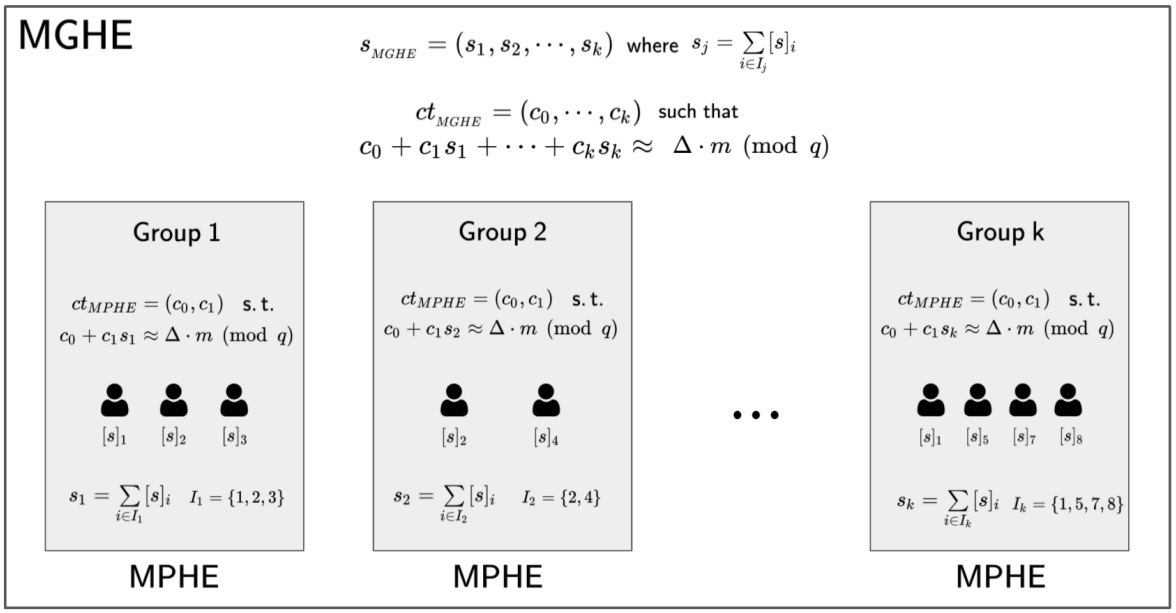


**Fig. 1.** A schematic presenting the overall structure of MGHE schemes. Each boxed group of participants acts as an MPHE scheme. The secret keys and ciphertext equations for each group and the entire set of participants (including between groups) are described above.

pair. To be precise, an MPHE scheme behaves like a single-key HE scheme where the joint secret key $s = \sum_{i \in I} [s]_i$ is additively shared among the members of $I$. We make the Common Random String (CRS) assumption to construct a joint public (encryption) key: given a random polynomial $a \in R_q$, each party $i \in I$ generates $[b]_i = a \cdot [s]_i + [e]_i \pmod q$ for some error $[e]_i$, then the joint public key is obtained as $b = \sum_{i \in I} [b]_i \approx a \cdot s \pmod q$. However, it is more challenging to generate a joint evaluation key because, roughly speaking, the evaluation key is usually supposed to be an 'encryption' of $s^2$ which has quadratic structure with respect to the individual secrets $[s]_i$. In the previous constructions [5, 38], the key generation procedure involves a multi-round protocol among the parties: (1) parties publish individual encryption keys to build a joint encryption key, then (2) use it to generate 'encryptions' of $[s]_i \cdot s$ and broadcast them to construct a joint evaluation key. We propose a new key generation algorithm which is *nearly linear* with respect to the secret key. This property enables the non-interactive key generation in that each party independently generates and broadcasts its public key $[\mathsf{pk}]_i$ once, which adds up to the joint public (encryption and evaluation) key $\mathsf{jpk} = \sum_{i \in I} [\mathsf{pk}]_i$ corresponding to the joint secret $s = \sum_{i \in I} [s]_i$.

Finally, to construct our MGHE scheme, we extend the functionality of our MPHE scheme to support homomorphic computation between ciphertexts under different keys. For example, if we perform homomorphic computation on MPHE ciphertexts $\mathsf{ct}_j$ under the joint secret keys $s_j = \sum_{i \in I_j} [s]_i$ of groups $I_j$ for $1 \leq j \leq k$, then it outputs a 'multi-group' ciphertext under the secret $(s_1, \ldots, s_k)$. In particular, the joint public keys of the involved groups themselves are used in the relinearization process of multi-group ciphertexts so that no further interaction is required among the parties. The technical details of our MPHE and MGHE constructions are described in Sections 3 and 4.

Thus, our MGHE scheme behaves as if it is an MKHE scheme in which each key is jointly generated by a group of parties (akin to MPHE). This makes MGHE an ideal generalization of both these HE variants and the hierarchical key structure allows an MGHE scheme to take advantage of strengths of both MPHE and MKHE.

### 1.3 Related Work

We first remark that the terminology for HE-like primitive has not been agreed upon yet in the literature. We use the terms 'MPHE' and 'MKHE' to classify the related works.

Asharov et al. [5] designed the first MPHE scheme from BGV [11]. Mouchet et al. [38] proposed a simplified construction from BFV [10, 25] and presented some experimental results. Park [40] recently modified the key generation protocol to reduce the interaction and also suggested a conversion between MPHE and MKHE. To the best of our knowledge, all known MPHE schemes require a multi-round protocol among the parties to generate a shared key pair.

On the other hand, there have been several attempts to construct an MKHE scheme by generalizing single-key HE schemes. López-Alt et al. [35] designed the first MKHE from NTRU [32], and [22, 39, 41] studied multi-key variants of GSW [28]. Then, Brakerski and Perlman [12] presented an LWE-based MKHE [12], followed by Chen et al. [14] who presented a multi-key variant of TFHE [20]. Another line of work [17, 15] studied MKHE schemes from batched HEs such as BGV [11], BFV [10, 25] and CKKS [19]. Our MGHE construction is inspired by [15], but we make an additional CRS assumption to possess the linearity property. Recently, Ananth et al. [4] proposed a general methodology to design an MKHE scheme in the plain model. The construction is done by combining an oblivious transfer protocol and MKHE schemes with limited functionality or trusted setup.

We remark that some MKHE schemes can be converted into MGHE: if the key generation algorithm of an MKHE scheme has the homomorphic property, then we can simply operate on the public keys of multiple parties to build a shared key for the group. For example, multi-key GSW schemes [22, 39, 41] hold the condition since GSW does not require an evaluation key for multiplication.

Aloufi et al. [3] combined MPHE and MKHE to perform computation on ciphertexts under two different keys: a joint key of model owners and the other of a client. This can be viewed as a special case of MGHE in which there are two groups consisting of model owners and a client, respectively. However, its key generation procedure also involves an interactive protocol to obtain a evaluation key.

Boneh et al. [9] suggested the notion of threshold FHE that has $t$-out-of-$n$ access structure protocol by splitting the secret key into shares. Its key generation is based on a Shamir secret sharing scheme where each party receives a share of the secret key. Badrinarayanan et al. [6] also presented a threshold FHE scheme but in the distributed setting.

## 2 Background

### 2.1 Notation

We assume all logarithms are in base two unless otherwise indicated. Vectors are denoted in bold, e.g. $\mathbf{a}$, and matrices in upper-case bold, e.g. $\mathbf{A}$. We denote the inner product of two vectors $\mathbf{u}$, $\mathbf{v}$ as $\langle \mathbf{u}, \mathbf{v} \rangle$. For a finite set $S$, $U(S)$ denotes the uniform distribution over $S$.

Let $N$ be a power of two. We denote by $R = \mathbb{Z}[X]/(X^N + 1)$ the ring of integers of the $(2N)$-th cyclotomic field and $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ the residue ring of $R$ modulo an integer $q$. An element of $R$ (or $R_q$) is uniquely represented as a polynomial of degree less than $N$ with coefficients in $\mathbb{Z}$ (or $\mathbb{Z}_q$). We identify $a = \sum_{0 \le i < N} a_i \cdot X^i \in R$ with the vector of its coefficients $(a_0, \ldots, a_{N-1}) \in \mathbb{Z}^N$. For $\sigma > 0$, we denote by $D_\sigma$ a distribution over $R$ which samples $N$ coefficients independently from the discrete Gaussian distribution of variance $\sigma^2$ and $\chi$ as a key distribution.

For $a, b \in R_q$, we informally write $a \approx b \pmod q$ if $b = a + e \pmod q$ for some small $e \in R$.

### 2.2 Ring Learning with Errors

Given the parameters $(N, q, \chi, \sigma)$, consider the samples of the form $b_i = s \cdot a_i + e_i \pmod q$ for polynomial number of $i$'s where $a_i \leftarrow U(R_q)$ and $e_i \leftarrow D_\sigma$ for a fixed $s \leftarrow \chi$. The Ring Learning with Errors (RLWE) assumption states that the RLWE samples $(b_i, a_i)$'s are computationally indistinguishable from uniformly random elements of $U(R_q^2)$.

### 2.3 Gadget Decomposition and External Product

The gadget decomposition is a widely used technique in HE schemes to manage the noise growth of homomorphic operations. For a *gadget vector* $\mathbf{g} = (g_i) \in \mathbb{Z}_q^d$ and an integer $q$, the gadget decomposition is a map $\mathbf{g}^{-1} : R_q \to R^d$ such that $a = \langle \mathbf{g}^{-1}(a), \mathbf{g} \rangle \pmod q$ for all $a \in R_q$. Typical examples are bit decomposition [10, 11], digit decomposition [20], and Residue Number System (RNS) based decompositions [7, 30]. Our implementation is based on an RNS-friendly decomposition for efficiency.

For $\mu \in R$, we call $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1) \in R_q^{d\times 2}$ a *gadget encryption* of $\mu$ under secret $s$ if $\mathbf{u}_0 + s \cdot \mathbf{u}_1 \approx \mu \cdot \mathbf{g} \pmod q$. Chillotti et al. [20] formalized an operation between RLWE and RGSW ciphertexts and named it the *external product*. We adopt and generalize this concept as follows: For $c \in R_q$ and $\mathbf{v} \in R_q^d$, we define the external product as $c \boxdot \mathbf{v} := \langle \mathbf{g}^{-1}(c), \mathbf{v} \rangle \pmod q$. We also write $c \boxdot \mathbf{U} = (c \boxdot \mathbf{u}_0, c \boxdot \mathbf{u}_1)$ for $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1) \in R_q^{d\times 2}$. We note that $\langle c \boxdot \mathbf{U}, (1, s) \rangle = c \boxdot (\mathbf{u}_0 + s \cdot \mathbf{u}_1) \approx c \cdot \mu \pmod q$ if $\mathbf{U}$ is a gadget encryption of $\mu$.

The special modulus technique [27] is a well-known technique to reduce the noise growth of homomorphic operations. Although the special modulus technique is applied to the external product in our implementation, we do not describe it in our scheme description for simplicity.

### 2.4 Variants of HE for Multiple Parties

The standard HE schemes support computation on ciphertexts encrypted under the same secret key. In the context of multiple parties, however, the template raises an issue where authority is concentrated on whom possesses the key. To extend HE to the case of multiple parties, research has been undertaking in two directions: threshold HE and multi-key HE (MKHE). Threshold HE [5, 9, 38, 40, 36] has $t$-out-of-$n$ access structure, that is, any set of $t$ parties can reconstruct the secret key. Several studies on threshold HE are dedicated to the case of $t = n$ (which we call MPHE), while there have been a few number of studies for $t < n$ with some limitations such as huge parameter requirement or intricate assumptions for decryption. The other line of work, MKHE [35, 22, 39, 41, 14, 15], gets the parties who are associated with the output ciphertext involved in the decryption phase. The key difference is that, unlike MPHE where the participating parties should be pre-determined in the key generation phase, MKHE is free to involve new parties in computation. In this paper, we focus on MPHE and MKHE.

An MPHE scheme over a plaintext space $\mathcal{M}$ is a tuple of algorithms $\mathtt{MPHE} = (\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$:

- **Setup:** $\mathsf{pp} \leftarrow \mathtt{MPHE.Setup}(1^\lambda, 1^d)$. On input the security parameter $\lambda$ and a depth bound $d$, the setup algorithm outputs a public parameter set $\mathsf{pp}$.
- **Key Generation Protocol:** $(\mathsf{jpk}, \{[\mathsf{sk}]_i\}_{i\in I}) \leftarrow \mathtt{MPHE.KeyGen}(I)$. A set of parties $I$ runs the key-generation protocol to generate a joint public key $\mathsf{jpk}$. Each party $i \in I$ also obtains a private share $[\mathsf{sk}]_i$ of the (implicitly defined) secret key $\mathsf{sk}$.
- **Encryption:** $\mathsf{ct} \leftarrow \mathtt{MPHE.Enc}(\mathsf{jpk}; m)$. Given a joint public key $\mathsf{jpk}$ and a message $m \in \mathcal{M}$, output a ciphertext $\mathsf{ct}$.
- **Evaluation:** $\mathsf{ct} \leftarrow \mathtt{MPHE.Eval}(\mathsf{jpk}; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_L)$. Given input a circuit $C : \mathcal{M}^L \to \mathcal{M}$ and ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_L$, the evaluation algorithm outputs a ciphertext $\mathsf{ct}$.
- **Decryption:** $m \leftarrow \mathtt{MPHE.Dec}(\{[\mathsf{sk}]_i\}_{i\in I}; \mathsf{ct})$. Given a ciphertext $\mathsf{ct}$ and the secret key shares $\{[\mathsf{sk}]_i\}_{i\in I}$, the decryption algorithm outputs a message $m \in \mathcal{M}$.

In MPHE, the same public key is used among all parties for encryption and homomorphic evaluation. We note that the key generation protocol of existing MPHE schemes (e.g. [5, 38, 40]) required multiple rounds to construct a joint evaluation (relinearization) key. The security and correctness of MPHE are defined as follows.

**Definition 1 (Correctness of MPHE).** *Let* $\mathsf{pp} \leftarrow \mathtt{MPHE.Setup}(1^\lambda, 1^d)$ *and* $(\mathsf{jpk}, \{[\mathsf{sk}]_i\}_{i\in I}) \leftarrow \mathtt{MPHE.KeyGen}(I)$ *for a set of parties* $I$*. Then for any* $m_1, \ldots, m_L \in \mathcal{M}$ *and for any circuit* $C : \mathcal{M}^L \to \mathcal{M}$ *of depth* $\le d$*, the probability that*

$$\mathtt{MPHE.Dec}(\{[\mathsf{sk}]_i\}_{i\in I}; \mathsf{ct}) \neq C(m_1, \ldots, m_L)$$

*is negligible with* $\lambda$ *where* $\mathsf{ct}_j \leftarrow \mathtt{MPHE.Enc}(\mathsf{jpk}; m_j)$ *for* $j = 1, \ldots, L$ *and* $\mathsf{ct} \leftarrow \mathtt{MPHE.Eval}(\mathsf{jpk}; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_L)$*.*

**Definition 2 (Security of MPHE).** *Let $I$ be a set of parties. An MPHE scheme is said to be secure if, for any set $A \subsetneq I$ and $H = I \backslash A$, the advantage of $\mathcal{A}$ in the following game is negligible for any PPT adversary $\mathcal{A}$:*

1. *The challenger generates a public parameter* $\mathsf{pp} \leftarrow \mathtt{MPHE.Setup}(1^\lambda, 1^d)$.
2. *The adversary plays with an honest challenger the key generation protocols* $\mathtt{MPHE.KeyGen}(I)$. *At the end of the protocol, the adversary gets the joint public key* $\mathsf{jpk}$ *and secret shares* $[\mathsf{sk}]_i$ *for* $i \in A$ *while the challenger obtains* $\mathsf{jpk}$ *and* $[\mathsf{sk}]_i$ *for* $i \in H$.
3. *The adversary chooses messages* $m_0, m_1 \in \mathcal{M}$ *and sends them to the challenger. The challenger samples a random bit* $b \in \{0,1\}$ *and sends* $\mathtt{MPHE.Enc}(\mathsf{jpk}; m_b)$ *back to the adversary.*
4. *The adversary $\mathcal{A}$ outputs $b'$. The advantage of the adversary is defined as* $2\left|\Pr[b = b'] - \frac{1}{2}\right|$.

MKHE is yet another variant of HE with different key structure and functionality from MPHE. In the MKHE setting, each party can generate its key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ independently and messages can be encrypted using different public keys. A fresh ciphertext corresponds to a single party $i$ who generated the public key $\mathsf{pk}_i$ used in encryption, but a multi-key ciphertext is in general associated with an (ordered) set $I$ of parties (we will assume that it is implicit in the ciphertext). More specifically, if we perform some operations on multi-key ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$ where each $\mathsf{ct}_i$ corresponds to $I_i$, the resulting ciphertext is associated to the parties in $I_1 \cup \cdots \cup I_k$.

An MKHE scheme for a plaintext space $\mathcal{M}$ is a tuple of algorithms $\mathtt{MKHE} = (\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$:

- **Setup:** $\mathsf{pp} \leftarrow \mathtt{MKHE.Setup}(1^\lambda, 1^d)$. On input the security parameter $\lambda$ and a depth bound $d$, the setup algorithm outputs a public parameter set $\mathsf{pp}$.
- **Key Generation:** $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathtt{MKHE.KeyGen}(i)$. Each party $i$ generates its own key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$.
- **Encryption:** $\mathsf{ct} \leftarrow \mathtt{MKHE.Enc}(\mathsf{pk}_i; m)$. Given a public key $\mathsf{pk}_i$ and a message $m \in \mathcal{M}$, output a ciphertext $\mathsf{ct}$.
- **Evaluation:** $\mathsf{ct} \leftarrow \mathtt{MKHE.Eval}(\{\mathsf{pk}_i\}_{i \in I}; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_L)$. Given input a circuit $C : \mathcal{M}^L \to \mathcal{M}$, ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_L$, and the collection of public keys $\mathsf{pk}_i$ which are associated to at least one input ciphertext, output a ciphertext $\mathsf{ct}$ (corresponding to $I$).
- **Decryption:** $m \leftarrow \mathtt{MKHE.Dec}(\{\mathsf{sk}_i\}_{i \in I}; \mathsf{ct})$. Given a ciphertext $\mathsf{ct}$ and the corresponding secret keys $\mathsf{sk}_i$ for $i \in I$, output a message $m \in \mathcal{M}$.

MKHE is more flexible and dynamic in that any new party can join the computation in the middle, contrasting with MPHE where parties must be pre-determined in the key generation phase.

**Definition 3 (Correctness of MKHE).** *Let* $\mathsf{pp} \leftarrow \mathtt{MKHE.Setup}(1^\lambda, 1^d)$ *and* $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathtt{MKHE.KeyGen}(i)$ *for $i \in I$. For party $i \in I$, $\mathsf{pk}_i$ be a public key and $\mathsf{sk}_i$ be the corresponding secret key generated by party $i$. For any $m_1, \ldots, m_L \in \mathcal{M}$ and $k_1, \ldots, k_L \in I$, let* $\mathsf{ct}_j \leftarrow \mathtt{MKHE.Enc}(\mathsf{pk}_{k_j}; m_j)$ *for* $1 \le j \le L$. *For any circuit $C : \mathcal{M}^L \to \mathcal{M}$ of depth $\le d$, it holds that*

$$\mathtt{MKHE.Dec}(\{\mathsf{sk}_i\}_{i \in I}; \mathtt{MKHE.Eval}(\{\mathsf{pk}_i\}_{i \in I}; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_L)) = C(m_1, \ldots, m_L)$$

*with an overwhelming probability in $\lambda$.*

**Definition 4 (Security of MKHE).** *An MKHE scheme is said to be secure if it is semantically secure. In other words, for the security parameter* $\lambda$, $\mathsf{pp} \leftarrow \mathtt{MKHE.Setup}(1^\lambda, 1^d)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathtt{MKHE.KeyGen}(i)$ *and for any $m_0, m_1 \in \mathcal{M}$, the distributions* $\mathtt{MKHE.Enc}(\mathsf{pk}_i; m_0)$ *and* $\mathtt{MKHE.Enc}(\mathsf{pk}_i; m_1)$ *are computationally indistinguishable.*

Both MPHE and MKHE are useful generalizations of single-key HE with different pros and cons, but it may not be straightforward to build a secure multi-party protocol from these HE primitives. We will discuss relevant issues and provide a general strategy in Sec. 5.

## 3 MPHE with Non-Interactive Setup

In this section, we present a new MPHE scheme with non-interactive key generation. To the best of our knowledge, all known MPHE constructions [5, 38, 40] have a common limitation in that the geratation of a joint pulbic and evaluation keys require a multi-round protocol among the parties since the relinearization key has a non-linear structure with respect to the joint secret key. For instance, in the latest work on MPHE [40], the joint key generation consists of two steps:

1. Each party $i$ samples a secret $[s]_i$, then uses a CRS $a \in R_q$ to generate and broadcast $[b]_i \approx -a \cdot [s]_i \pmod q$. At the end, all parties share a joint encryption key $\mathsf{jek} = (b = \sum_i b_i, a) \in R_q^2$ such that $b + as \approx 0 \pmod q$ corresponding to the joint secret $s = \sum_i [s]_i$.
2. Each party $i$ uses $\mathsf{jek}$ to generate a gadget encryption of $[s]_i \cdot s$ under $s$. This is done by first generating a gadget encryption of zero (whose rows are independently sampled as an RLWE encryption of zero using $\mathsf{jek}$) then adding $[\mathbf{0}|[s]_i \cdot \mathbf{g}]$ to it. At the end, all parties broadcast gadget encryptions of $[s]_i \cdot s$ and aggregate them to build a gadget encryption of $\sum_i [s]_i \cdot s = s^2$, which will be used as a joint relinearization key.

As shown above, the main challenge was to generate a joint relinearization key, which is a gadget encryption of $s^2$ under $s$, due to its quadratic structure. In this work, we solve this issue by modifying the public key to have a *nearly linear* structure. Consequently, our MPHE scheme allows each party to independently generate an individual public key at once (even without any information about other parties), and the joint public key can be built on the server by simply adding individual public keys. We describe a multi-party variant of BFV to present our idea, but also provide a CKKS-based construction in Appendix A.1.

In Section 3.1, we first provide the basic algorithms including setup, key generation, encryption, and decryption. In Section 3.2, we provide the algorithms of arithmetic operations and automorphism, together with their correctness. In Section 3.3, we provide the security analysis of our scheme.

### 3.1 Basic Scheme

Our scheme is based on the Common Reference String (CRS) model, *i.e.,* all parties have access to the same random string. A parameter set also includes the RLWE dimensions, ciphertext modulus, the key distribution, as well as the error parameter.

• $\underline{\texttt{MPHE.Setup}(1^\lambda)}$: Set the RLWE dimension $N$, the plaintext modulus $t$, the ciphertext modulus $q$, the key distribution $\chi$ over $R$, and the error parameter $\sigma$. Sample random vectors $\mathbf{a}, \mathbf{u}, \mathbf{k} \leftarrow U(R_q^d)$ and choose a gadget decomposition $\mathbf{g}^{-1} : R_q \to R^d$ with a gadget vector $\mathbf{g} \in R_q^d$. Return the parameter set $\mathsf{pp} = (N, t, q, \chi, \sigma, \mathbf{a}, \mathbf{u}, \mathbf{k}, \mathbf{g}^{-1}, \mathbf{g})$. We write $\Delta = \lfloor q/t \rceil$.

We note that the parameter set in our scheme include multiple CRSs denoted as $\mathbf{a}, \mathbf{u}$, and $\mathbf{k}$. However, this should not be considered as a strong assumption. From a practical point of view, we can implement a CRS of arbitrary size using a keyed pseudorandom function (PRF). This allows us to rely on the CRS assumption for a fixed-size seed, regardless of the number of common random polynomials used for public and automorphism keys.

• $\underline{\texttt{MPHE.IndKeyGen}(i)}$: Each party $i$ samples $[s]_i, [r]_i \leftarrow \chi$ and $[\mathbf{e}_0]_i, [\mathbf{e}_1]_i, [\mathbf{e}_2]_i, [\mathbf{e}_3]_i \leftarrow D_\sigma^d$. Set $[\mathbf{b}]_i = -[s]_i \cdot \mathbf{a} + [\mathbf{e}_0]_i \pmod q$, $[\mathbf{d}]_i = -[r]_i \cdot \mathbf{a} + [s]_i \cdot \mathbf{g} + [\mathbf{e}_1]_i \pmod q$, $[\mathbf{v}]_i = -[s]_i \cdot \mathbf{u} - [r]_i \cdot \mathbf{g} + [\mathbf{e}_2]_i \pmod q$, and $[\mathbf{h}]_i = -[s]_i \cdot \mathbf{k} + \psi([s]_i) \cdot \mathbf{g} + [\mathbf{e}]_i \pmod q$. Return the secret key $[\mathsf{sk}]_i = [s]_i$, the public key $[\mathsf{pk}]_i = ([\mathbf{b}]_i, [\mathbf{d}]_i, [\mathbf{v}]_i, [\mathbf{h}]_i)$.

• $\underline{\texttt{MPHE.JointKeyGen}(\{[\mathsf{pk}]_i : i \in I\})}$: Let $I$ be a group of parties. Given a set of public keys $[\mathsf{pk}]_i$ of parties $i$ in $I$, return the joint public key $\mathsf{jpk} = (\mathbf{b}, \mathbf{d}, \mathbf{v}, \mathbf{h}) \in R_q^{d \times 4}$ where $\mathbf{b} = \sum_{i \in I} [\mathbf{b}]_i$, $\mathbf{d} = \sum_{i \in I} [\mathbf{d}]_i$, $\mathbf{v} = \sum_{i \in I} [\mathbf{v}]_i$, and $\mathbf{h} = \sum_{i \in I} [\mathbf{h}]_i \pmod q$. We denote the joint encryption key as $\mathsf{jek} = (\mathbf{b}[0], \mathbf{a}[0]) \in R_q^2$, the joint relinearization key as $\mathsf{jrlk} = (\mathbf{b}, \mathbf{d}, \mathbf{v})$, and the joint automorphism key as $\mathsf{jak} = \mathbf{h}$.

The key generation procedure consists of two steps; each party $i$ first generates (locally) a key pair and publishes a public key $[\mathsf{pk}]_i$, and then a joint public key of a group of parties is built as $\mathsf{pk} = \sum_i [\mathsf{pk}]_i$ from the collection of public keys from the associated parties. The generation of the individual keys can be done locally without knowing any information about other parties, and building the joint public key can be done publicly by the cloud without further interaction with the involved parties. This 'non-interactive' property not only has advantages in communication, but also provide better flexibility. For instance, if a party $i$ would like to participate as a member of several groups, it is not necessary to run the key generation algorithm multiple times but allowed to reuse the same key $[\mathsf{pk}]_i$ for all groups. More discussions on this feature will be given in Section 5.1.

Each component of the public key $[\mathsf{pk}]_i$ forms a gadget encryption with a CRS under the secrets $[s]_i$ or $[r]_i$. We call $s = \sum_{i \in I} [s]_i$ the (implicitly defined) joint secret key of the group $I$ of parties. The individual secrets $[s]_i$ of parties $i \in I$ can be viewed as additive shares of $s$. Furthermore, the public key $[\mathsf{pk}]_i$ is *nearly linear* with respect to $[s]_i$ and $[r]_i$ so that the joint public key $\mathsf{jpk} = (\mathbf{b}, \mathbf{d}, \mathbf{v}, \mathbf{h})$ satisfies the same properties as the individual keys:

$$\begin{aligned}
\mathbf{b} &\approx -s \cdot \mathbf{a} && \pmod q \\
\mathbf{d} &\approx -r \cdot \mathbf{a} + s \cdot \mathbf{g} && \pmod q \\
\mathbf{v} &\approx -s \cdot \mathbf{u} - r \cdot \mathbf{g} && \pmod q \\
\mathbf{h} &\approx -s \cdot \mathbf{k} + \psi(s) \cdot \mathbf{g} && \pmod q
\end{aligned}$$

Note that the encryption key $\mathsf{jek}$ can be viewed as an RLWE instance with secret $s$. The usual BFV encryption and decryption algorithms are used in our scheme as follows.

• $\underline{\texttt{MPHE.Enc}(\mathsf{jek}; m)}$: Given a message $m \in R_p$ and the joint encryption key $\mathsf{jek}$, sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Return the ciphertext $\mathsf{ct} = w \cdot \mathsf{jek} + (\Delta \cdot m + e_0, e_1) \pmod q$.

• $\underline{\texttt{MPHE.Dec}(\mathsf{sk}; \mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1)$ and a secret key $\mathsf{sk} = s$, output $m = \lfloor (t/q) \cdot (c_0 + c_1 \cdot s) \rceil \pmod t$.

**Correctness of encryption.** Our encryption algorithm returns a valid BFV ciphertext under the secret $s$. The only difference is that our encryption key has a larger noise variance depending on the number of parties which also affects the final encryption noise.

Ideally, an MPHE ciphertext is decryptable by the joint secret key $s$, however, the basic decryption algorithm is not generally useful in practice since the joint secret is shared between the parties in $I$. On the other hand, the parties can perform a simple multi-party protocol to decrypt an MPHE ciphertext in a distributed manner. As an example, we present a well-known method based on the noise smudging technique [5, 38]. In this protocol, each party $i \in I$ partially decrypts the input ciphertext using $[s]_i$ and publishes its approximate value by adding auxiliary noise, then the plaintext can be recovered from the sum of partial decryptions. We discuss how to choose appropriate $\sigma'$, an error parameter for noise smudging, for the partial decryption in Section 5.2.

• $\underline{\texttt{MPHE.DistDec}(\{[\mathsf{sk}]_i : i \in I\}, \sigma'; \mathsf{ct})}$: Let $\mathsf{ct} = (c_0, c_1)$ be a multi-party ciphertext, $\sigma' > 0$ an error parameter, and $[\mathsf{sk}]_i = [s]_i$ the secret key of party $i \in I$. The distributed decryption protocol consists of the following procedures:

- Partial decryption: Each party $i \in I$ samples $[e']_i \leftarrow D_{\sigma'}$, then computes and publishes $[\mu]_i = c_1 \cdot [s]_i + [e']_i \pmod q$.
- Merge: Compute $\mu = c_0 + \sum_{i \in I} [\mu]_i \pmod q$ and return $m = \lfloor (t/q) \cdot \mu \rceil$.

### 3.2 Basic Operations

In the following, we present basic operations which are homomorphic addition, multiplication, and automorphism algorithms. The major difference between our scheme and the standard BFV scheme is in their multiplication: our relinearization algorithm during the multiplication is more expensive due to the linear structure of a joint public key, but it provides the same functionality as shown below.

**Algorithm 1** Relinearization procedure of MPHE

**Input:** $\mathsf{jrlk} = (\mathbf{b}, \mathbf{d}, \mathbf{v})$, $\mathsf{ct}_{\mathsf{mul}} = (c_0'', c_1'', c_2'')$
**Output:** $\mathsf{ct}_{\mathsf{relin}} = (c_0^*, c_1^*) \in R_q^2$

1: $c_2^* \leftarrow c_2'' \boxdot \mathbf{b}$
2: $(c_0^*, c_1^*) \leftarrow (c_0'', c_1'' + c_2'' \boxdot \mathbf{d}) + c_2^* \boxdot (\mathbf{v}, \mathbf{u}) \pmod q$

• $\underline{\texttt{MPHE.Add}(\mathsf{ct}, \mathsf{ct}')}$: Given two ciphertexts $\mathsf{ct}, \mathsf{ct}' \in R_q^2$, output $\mathsf{ct}_{\mathsf{add}} = \mathsf{ct} + \mathsf{ct}' \pmod q$.

• $\underline{\texttt{MPHE.Mult}(\mathsf{jrlk}; \mathsf{ct}, \mathsf{ct}')}$: Given two ciphertexts $\mathsf{ct} = (c_0, c_1)$, $\mathsf{ct}' = (c_0', c_1')$ and a joint relinearization key $\mathsf{jrlk}$, let $\mathsf{ct}_{\mathsf{mul}} = (c_0'', c_1'', c_2'')$ where $c_0'' = \lfloor (t/q) \cdot (c_0 c_0') \rceil \pmod q$, $c_1'' = \lfloor (t/q) \cdot (c_0 c_1' + c_1 c_0') \rceil \pmod q$ and $c_2'' = \lfloor (t/q) \cdot (c_1 c_1') \rceil \pmod q$. Return the ciphertext $\mathsf{ct}_{\mathsf{relin}} \leftarrow \texttt{MPHE.Relin}(\mathsf{jrlk}; \mathsf{ct}_{\mathsf{mul}})$ where $\texttt{MPHE.Relin}(\cdot;\cdot)$ is the relinearization procedure described in Alg. 1.

**Correctness of homomorphic multiplication.** Let $[s]_i, [r]_i$ be the polynomials sampled from $\chi$ during the generation of a key pair $[\mathsf{sk}]_i = [s]_i$ and $[\mathsf{rlk}]_i = ([\mathbf{b}]_i, [\mathbf{d}]_i, [\mathbf{v}]_i)$ of the $i$-th party and let $s = \sum_{i \in I} [s]_i$ and $r = \sum_{i \in I} [r]_i$. First of all, we remark that the first step of homomorphic multiplication computing $\mathsf{ct}_{\mathsf{mul}}$ is identical to the usual BFV scheme. If $\mathsf{ct}$ and $\mathsf{ct}'$ are valid BFV encryptions of $m$ and $m'$, respectively, then $\mathsf{ct}_{\mathsf{mul}} = (c_0'', c_1'', c_2'')$ is an encryption of $mm'$ under $(1, s, s^2)$, that is, $c_0'' + c_1'' \cdot s + c_2'' \cdot s^2 \approx \Delta \cdot mm' \pmod q$.

Now suppose that $(c_0^*, c_1^*) \leftarrow \texttt{MPHE.Relin}(\mathsf{jrlk}; (c_0'', c_1'', c_2''))$ is the output of our relinearization algorithm. We claim that $c_0^* + c_1^* \cdot s \approx c_0'' + c_1'' \cdot s + c_2'' \cdot s^2 \pmod q$. Recall that the joint public key satisfies $\mathbf{b} + s \cdot \mathbf{a} \approx 0 \pmod q$, $\mathbf{d} + r \cdot \mathbf{a} \approx s \cdot \mathbf{g} \pmod q$ and $\mathbf{v} + s \cdot \mathbf{u} \approx -r \cdot \mathbf{g} \pmod q$. Then, we have

$$\begin{aligned}
c_0^* + c_1^* \cdot s &= c_0'' + c_1'' \cdot s + (c_2'' \boxdot \mathbf{d}) \cdot s + c_2^* \boxdot (\mathbf{v} + s \cdot \mathbf{u}) && \pmod q \\
&\approx c_0'' + c_1'' \cdot s + c_2'' \boxdot (-rs \cdot \mathbf{a} + s^2 \cdot \mathbf{g}) - c_2^* \boxdot (r \cdot \mathbf{g}) && \pmod q \\
&\approx c_0'' + c_1'' \cdot s + r \cdot (c_2'' \boxdot \mathbf{b}) + c_2'' \cdot s^2 - r \cdot c_2^* && \pmod q \\
&\approx c_0'' + c_1'' \cdot s + c_2'' \cdot s^2 && \pmod q
\end{aligned}$$

as desired.

The packing technique of the BFV scheme enables us to encode multiple values in a finite field into a single plaintext polynomial for better efficiency [11]. The (un)packing algorithm has a similar algebraic structure with the canonical embedding map over the cyclotomic field $K = \mathbb{Q}[X]/(X^N + 1)$, and the automorphisms in the Galois group $\mathcal{G}al(K/\mathbb{Q})$ provide special functionality on the plaintext slots such as rotation. Below, we introduce the homomorphic operation of the automorphism $\psi$.

• $\underline{\texttt{MPHE.Auto}(\mathsf{jak}; \mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1)$ and the joint automorphism key $\mathsf{jak} = \mathbf{h}$, compute and return the ciphertext $\mathsf{ct}_{\mathsf{aut}} = (\psi(c_0), 0) + \psi(c_1) \boxdot (\mathbf{h}, \mathbf{k}) \pmod q$.

**Correctness of homomorphic automorphism.** Let $\mathsf{ct}_{\mathsf{aut}} = (c_0', c_1') \leftarrow \texttt{Auto}(\mathsf{jak}; \mathsf{ct})$ for a ciphertext $\mathsf{ct} = (c_0, c_1)$. As mentioned above, the joint automorphism key $\mathsf{jak} = \mathbf{h}$ satisfies that $\mathbf{h} + s \cdot \mathbf{k} \approx \psi(s) \cdot \mathbf{g} \pmod q$. Therefore, we have

$$\begin{aligned}
c_0' + c_1' \cdot s &= \psi(c_0) + \psi(c_1) \boxdot (\mathbf{h} + s \cdot \mathbf{k}) && \pmod q \\
&\approx \psi(c_0) + \psi(c_1) \cdot \psi(s) && \pmod q \\
&= \psi(c_0 + c_1 \cdot s) && \pmod q
\end{aligned}$$

### 3.3 Security

In this section, we show that our MPHE scheme achieves a semantic security under the RLWE assumption.

**Lemma 1 (Security of MPHE).** *The MPHE scheme described above is semantically secure under the RLWE assumption with parameter $(n, q, \chi, \sigma)$.*

*Proof.* Let $I$ be a group of parties, $A \subsetneq I$ be a set, and $H = I \backslash A$. We define some hybrid games as follows:

- **Game 0:** This is a real world execution of the security game defined in Definition 2.
- **Game 1:** It is similar to **Game 0**, but the challenger samples $[\mathsf{pk}]_i$ uniformly at random from $R_q^{d\times 4}$ for $i \in H$.
- **Game 2:** It is similar to **Game 1**, but the challenger encrypts 0 instead of $m_b$.

Let $[\mathsf{pk}]_i = ([\mathbf{b}]_i, [\mathbf{d}]_i, [\mathbf{v}]_i, [\mathbf{h}]_i)$ be the public key of party $i \in H$. Since $([\mathbf{b}]_i, \mathbf{a})$ and $([\mathbf{v}]_i, \mathbf{u})$ follow the RLWE distribution of secret $[s]_i$, a pair $([\mathbf{b}]_i, [\mathbf{v}]_i)$ is indistinguishable from a uniform distribution over $R_q^{d\times 2}$. In addition, $([\mathbf{d}]_i, \mathbf{a})$ follows the RLWE distribution of secret $[r]_i$, $[\mathbf{d}]_i$ is also indistinguishable from a uniform distribution over $R_q^d$. Meanwhile, $([\mathbf{d}]_i, \mathbf{a})$ and $([\mathbf{v}]_i, \mathbf{u})$ can be viewed as a 'chain' of two gadget encryptions of $[s]_i$ and $-[r]_i$ under secrets $[r]_i$ and $[s]_i$, respectively. Here we make an additional *circular security* assumption which guarantees that our scheme remains secure even if $[\mathbf{d}]_i$, $[\mathbf{v}]_i$, and $[\mathbf{h}]_i$ are public. On the other hand, $[\mathbf{h}]_i$ is an gadget encryption of $\psi([s]_i)$ under $[s]_i$ with a random vector $\mathbf{k}$. Therefore, **Game 0** and **Game 1** are computationally indistinguishable.

In both **Game 1** and **Game 2**, the encryption key $\mathbf{b}[0] = \sum_{i\in A}[\mathbf{b}]_i[0] + \sum_{i\in H}[\mathbf{b}]_i[0]$ is computationally indistinguishable from a uniform random variable over $R_q$. This is because $H$ is non-empty, and each $[\mathbf{b}]_i$ is sampled uniformly from $R_q^d$ for all $i \in H$. Furthermore, under the RLWE assumption, encryptions of 0 and $m_b$ are computationally indistinguishable. As a result, the difference in advantage between **Game 1** and **Game 2** is negligible.

Based on the above reasoning, we can conclude that the adversary's advantage in **Game 0** is negligible. Therefore, our MPHE scheme achieves semantic security against semi-malicious corruptions in a real-world execution of the game. □

## 4 Construction of MGHE

In this section, we present a Multi-Group Homomorphic Encryption (MGHE) scheme, which improves the functionality of our existing MPHE scheme by supporting homomorphic operations between multi-party ciphertexts that do not necessarily have to be encrypted under the same key.

Following the previous section, we built an MGHE scheme from the BFV scheme, but our idea is easily applicable to design multi-group variants of other HE schemes such as BGV [11] and CKKS [19]. In particular, we implement MGHE schemes from both BFV and CKKS and present experimental results in Section 6. We provide a formal description of multi-group CKKS in Appendix A.2.

In Section 4.1, we provide a formal description of the MGHE scheme, and discuss its connection to MPHE and MKHE. In Section 4.3, we outline the overall scheme of MGHE together with its correctness proof and we provide the security analysis of MGHE in Section 4.4.

### 4.1 Formalizing MGHE

An MGHE scheme over a plaintext space $\mathcal{M}$ is a tuple of algorithms $\mathtt{MGHE} = (\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$:

- **Setup:** $\mathsf{pp} \leftarrow \mathtt{MGHE.Setup}(1^\lambda, 1^d)$. On input the security parameter $\lambda$ and a depth bound $d$, the setup algorithm outputs a public parameter set $\mathsf{pp}$.
- **Key Generation Protocol:** $\mathsf{jpk} \leftarrow \mathtt{MGHE.KeyGen}(I)$. For a group $I$, the parties $i \in I$ run the key-generation protocol and output a joint public key $\mathsf{jpk}$ corresponding to $I$. Each party $i \in I$ also obtains a private share $[\mathsf{sk}]_i$ of the (implicitly defined) secret key $\mathsf{sk}$.
- **Encryption:** $\mathsf{ct} \leftarrow \mathtt{MGHE.Enc}(\mathsf{jpk}; m)$. Given a joint public key $\mathsf{jpk}$ and a message $m \in \mathcal{M}$, output a ciphertext $\mathsf{ct}$.
  For convenience, we assume that every ciphertext includes references to the associated public keys even if it is not described explicitly. For example, a fresh ciphertext contains a reference to the public key that is used in its encryption.

- **Evaluation:** $\mathsf{ct} \leftarrow \mathtt{MGHE.Eval}(\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_L)$. Given input a circuit $C : \mathcal{M}^L \to \mathcal{M}$, $L$ ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_L$, let $\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k$ be the joint public keys which are associated to at least one of input ciphertexts. The evaluation algorithm outputs a ciphertext $\mathsf{ct}$ which contains $L$ references to $\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k$.
- **Decryption:** $m \leftarrow \mathtt{MGHE.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_k; \mathsf{ct})$. Given a ciphertext $\mathsf{ct}$ and the associated secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_k$, the decryption algorithm outputs a message $m \in \mathcal{M}$.

**Definition 5 (Correctness of MGHE).** *An MGHE scheme is said to be correct if the following holds: Let* $\mathsf{pp} \leftarrow \mathtt{MGHE.Setup}(1^\lambda, 1^d)$. *Let* $\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k$ *be public keys each of which is generated by parties in* $I_1, \ldots, I_k$ *and* $\mathsf{sk}_1, \ldots, \mathsf{sk}_k$ *the corresponding secret keys, respectively. For any* $m_1, \ldots, m_L \in \mathcal{M}$ *and indices* $1 \le k_1, \ldots, k_L \le k$, *let* $\mathsf{ct}_i \leftarrow \mathtt{MGHE.Enc}(\mathsf{jpk}_{k_i}; m_i)$. *For any circuit* $C : \mathcal{M}^L \to \mathcal{M}$ *of depth* $\le d$, *it holds that*

$$\mathtt{MGHE.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_k; \mathtt{MGHE.Eval}(\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_L)) = C(m_1, \ldots, m_L)$$

*with an overwhelming probability in* $\lambda$.

**Definition 6 (Security of MGHE).** *Let* $I_1, I_2, \ldots, I_k$ *be groups and let* $I = \cup_{1 \le j \le k} I_j$. *For any set* $A \subsetneq I$, *let* $H = I \backslash A$. *An MGHE scheme is said to be semantically secure if the advantage of* $\mathcal{A}$ *in the following game is negligible for any PPT adversary* $\mathcal{A}$*:*

- **Setup:** The challenger generates a public parameter $\mathsf{pp} \leftarrow \mathtt{MGHE.Setup}(1^\lambda, 1^d)$.
- **Key Generation:** The adversary plays with an honest challenger the key generation protocols $\mathtt{MGHE.KeyGen}(\mathsf{pp}, I_j)$ for all $1 \le j \le k$. At the end of the protocol, the adversary gets the secret shares $\{[\mathsf{sk}_j]_i : i \in A, 1 \le j \le k\}$ and the challenger receives $\{[\mathsf{sk}_j]_i : i \in H, 1 \le j \le k\}$.
- **Challenge:** The adversary chooses messages $m_0, m_1 \in \mathcal{M}$ and an index $j$ such that $I_j \nsubseteq A$, and sends them to the challenger. The challenger samples a random bit $b \in \{0, 1\}$ and sends $\mathtt{MGHE.Enc}(\mathsf{pk}_j; m_b)$ back to the adversary.
- **Output:** The adversary $\mathcal{A}$ outputs $b'$. The advantage of the adversary is defined as $\left|\Pr[b = b'] - \frac{1}{2}\right|$.

**Connections to MPHE and MKHE.** MPHE and MKHE are generalizations of HE with distributed authority with different characteristics. Our proposal, the MGHE primitive, can be considered a generalization of both primitives. In other words, MPHE and MKHE are specific instantiations of MGHE which are obtained by restricting the number of groups or parties. An MGHE scheme works on groups of parties, where the evaluation within a group is done in MPHE sense while it is done in MKHE sense across the groups. Hence, MGHE over a single group can be viewed as an MPHE scheme, on the other hand, it can also be viewed as an MKHE scheme when each group consists of only one party.

In addition, the semantic security of MGHE is also a natural extension of security definitions of MPHE and MKHE. In the single-group case, there is only one index to be chosen in the challenge phase, so the security game is the same as that of MPHE [34]. On the other hand, if each group $I_j$ contains only one party, then the encryption is done by a single key which corresponds to the ordinary semantic security of (MK)HE.

### 4.2 Basic Scheme

As we shall see, the setup, key generation, and encryption procedures happen to be identical to our MPHE scheme and thus is non-interactive. However, it also supports homomorphic operations between ciphertexts under different keys, and the dimension of ciphertext dimension may increase as the homomorphic computation progresses when we add or multiply two multi-group ciphertexts corresponding to different sets of groups.

• $\underline{\mathtt{MGHE.Setup}(1^\lambda)}$: Run $\mathtt{MPHE.Setup}(1^\lambda)$ and return the public parameter $\mathsf{pp} = (N, t, q, \chi, \sigma, \mathbf{a}, \mathbf{u}, \mathbf{k}, \mathbf{g}^{-1}, \mathbf{g})$.

• $\underline{\mathtt{MGHE.IndKeyGen}(i)}$: Each party $i$ runs $\mathtt{MPHE.IndKeyGen}(i)$ and outputs secret keys, and public keys $[\mathsf{sk}]_i = [s]_i$, $[\mathsf{pk}]_i = ([\mathbf{b}]_i, [\mathbf{d}]_i, [\mathbf{v}]_i, [\mathbf{h}]_i)$, respectively.

• $\underline{\texttt{MGHE.JointKeyGen}(\{[\mathsf{pk}]_i : i \in I\})}$: Given a set of public keys of $i \in I$, output the joint public key $\mathsf{jpk} = (\mathbf{b}, \mathbf{d}, \mathbf{v}, \mathbf{h}) \leftarrow \texttt{MPHE.JointKeyGen}(\{[\mathsf{pk}]_i : i \in I\})$. We write the joint encryption key as $\mathsf{jek} = (\mathbf{b}[0], \mathbf{a}[0])$, the joint relinearization key as $\mathsf{jrlk} = (\mathbf{b}, \mathbf{d}, \mathbf{v})$, and the joint automorphism key as $\mathsf{jak} = \mathbf{h}$.

• $\underline{\texttt{MGHE.Enc}(\mathsf{jek}; m)}$: Given a joint encryption key $\mathsf{jek}$ and a message $m$, return $\mathsf{ct} \leftarrow \texttt{MPHE.Enc}(\mathsf{jek}; m)$.

As we discussed in Section 4.1, an MGHE ciphertext holds the references to the associated public keys. In our scheme, each ciphertext stores an ordered set of the involved groups. For example, a fresh ciphertext encrypted by a joint public key $\mathsf{jpk} = \sum_{i \in I} [\mathsf{pk}]_i$ is linked to the set containing a single element $I$. More generally, a multi-group encryption of $m$ corresponding an ordered set of $k$ groups $\{I_1, \ldots, I_k\}$ is an $(k+1)$ tuple $\overline{\mathsf{ct}} = (c_0, c_1, \ldots, c_k) \in R_q^{k+1}$ satisfying $c_0 + c_1 \cdot s_1 + \cdots + c_k \cdot s_k = \Delta \cdot m + e \pmod q$ for some error $e$ where $s_j = \sum_{i \in I_j} [s]_i$ is the joint secret key of $I_j$ for $1 \le j \le k$.

Finally, we present a basic (ideal) decryption algorithm and a distributed decryption protocol. For given a ciphertext $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$ which is linked to $k$ groups $I_1, \ldots, I_k$, the basic algorithm takes as input the joint secret keys $s_i$ of the associated groups $I_i$ and recovers the plaintext message while the distributed decryption protocol let the parties in $\bigcup_{1 \le j \le k} I_j$ perform the same computation securely in a distributed manner. As we mentioned before, we describe how to set concrete $\sigma'$ for the distributed decryption decryption in Section 5.2.

• $\underline{\texttt{MGHE.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_k; \overline{\mathsf{ct}})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1, \ldots, c_k)$ and joint secret keys $\mathsf{sk}_j = s_j$ for $1 \le j \le k$, return $m = \left\lfloor (t/q) \cdot (c_0 + \sum_{1 \le j \le k} c_j \cdot s_j) \right\rceil \pmod t$.

• $\underline{\texttt{MGHE.DistDec}(\{[\mathsf{sk}]_i : i \in \cup_{1 \le j \le k} I_j\}, \sigma'; \overline{\mathsf{ct}})}$: Let $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$ be a multi-group ciphertext corresponding to an ordered set of groups $(I_1, \ldots, I_k)$. The distributed decryption protocol consists of the following procedures:

- Partial decryption: Let $I = \cup_{1 \le j \le k} I_j$. Each party $i \in I$ samples $[e']_i \leftarrow D_{\sigma'}$, then broadcasts $[\mu]_i = \left(\sum_{1 \le j \le k,\ i \in I_j} c_j\right) \cdot [s]_i + [e']_i \pmod q$.
- Merge: Compute $m = \left\lfloor (t/q) \cdot \left(c_0 + \sum_{i \in I} [\mu]_i\right) \right\rceil \pmod t$.

### 4.3 Basic Operations

Homomorphic operations include a pre-processing step that aligns the components of input ciphertexts as follows. For given two multi-group ciphertexts, we consider the corresponding ordered sets and compute their union, say $\{I_1, \ldots, I_k\}$. Then, we extend the input ciphertexts by padding some zeros and rearranging their components so that both ciphertexts are decryptable with respect to the same secret $\overline{\mathsf{sk}} = (s_1, \ldots, s_k)$ where $s_j$ is the joint secret of group $I_j$, $1 \le j \le k$. We assume that this pre-processing is always performed on the input ciphertext and the output ciphertext is linked to the union $\{I_1, \ldots, I_k\}$ of ordered sets even if it is not explicitly mentioned in the algorithm description.

• $\underline{\texttt{MGHE.Add}(\overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two ciphertexts $\overline{\mathsf{ct}}$ and $\overline{\mathsf{ct}}'$, return the ciphertext $\overline{\mathsf{ct}}_{add} = \overline{\mathsf{ct}} + \overline{\mathsf{ct}}' \pmod q$.

• $\underline{\texttt{MGHE.Mult}(\mathsf{jrlk}_1, \ldots, \mathsf{jrlk}_k; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two multi-group ciphertexts $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$, $\overline{\mathsf{ct}}' = (c'_0, \ldots, c'_k)$ and $k$ joint relinearization keys $\mathsf{jrlk}_1, \ldots, \mathsf{jrlk}_k$, compute $\overline{\mathsf{ct}}_{\mathsf{mul}} = (c_{i,j})_{0 \le i,j \le k}$ where $c_{i,j} = \left\lfloor (t/q) \cdot c_i c'_j \right\rceil \pmod q$ for $0 \le i, j \le k$. Return the ciphertext $\overline{\mathsf{ct}}_{\mathsf{relin}} \leftarrow \texttt{MGHE.Relin}(\mathsf{jrlk}_1, \ldots, \mathsf{jrlk}_k; \overline{\mathsf{ct}}_{\mathsf{mul}})$ where $\texttt{MGHE.Relin}(\cdot)$ is the relinearization procedure described in Alg. 2.

We remark that the relinearization algorithm can be shared between our MGHE scheme and the previous MKHE scheme [15] as they have the same ciphertext structure. Our relinearization algorithm is an improvement of the previous method which reduces the number of external products by almost a factor of 2. More formally, the prior algorithm computes lines 8-11 of Alg. 2 by repeating the following computation iteratively over $1 \le i, j \le k$:

$$(c_0^*, c_i^*) \leftarrow (c_0^*, c_i^*) + (c_{i,j} \boxdot \mathbf{b}_j) \boxdot (\mathbf{v}_i, \mathbf{u}) \pmod q.$$

**Algorithm 2** Relinearization procedure of MGHE

**Input:** $\overline{\mathsf{ct}}_{\mathsf{mul}} = (c_{i,j})_{0\le i,j\le k}$, $\mathsf{jrlk}_j = (\mathbf{b}_j, \mathbf{d}_j, \mathbf{v}_j)$ for $1 \le j \le k$.
**Output:** $\overline{\mathsf{ct}}_{\mathsf{relin}} = (c_j^*)_{0\le j\le k} \in R_q^{k+1}$.

$c_0^* \leftarrow c_{0,0}$
**for** $1 \le j \le k$ **do**
  $c_j^* \leftarrow c_{0,j} + c_{j,0} \pmod q$
**end for**
**for** $1 \le j \le k$ **do**
  $c_j^* \leftarrow c_j^* + \sum_{1\le i\le k} c_{i,j} \boxdot \mathbf{d}_i \pmod q$
**end for**
**for** $1 \le i \le k$ **do**
  $c_i'' \leftarrow \sum_{1\le j\le k} c_{i,j} \boxdot \mathbf{b}_j$
  $(c_0^*, c_i^*) \leftarrow (c_0^*, c_i^*) + c_i'' \boxdot (\mathbf{v}_i, \mathbf{u}) \pmod q$
**end for**

We observe that $\sum_{1\le j\le k} c_{i,j} \boxdot \mathbf{b}_j$ is pre-computable and reusable for the relinearization of multiple ciphertext components. This idea consequently reduces the number of external products down to $2k^2+2k$ in total, compared to the former method which requires $4k^2$ external products.

We provide a high-level sketch of the correctness proof. We refer the reader to Appendix B for details about the noise analysis.

**Correctness of homomorphic multiplication.** Suppose that $\overline{\mathsf{ct}}$ and $\overline{\mathsf{ct}}'$ are encryptions of $m$ and $m'$ under secret $\overline{\mathsf{sk}} = (s_1, \dots, s_k)$, respectively, and let $\overline{\mathsf{ct}}_{\mathsf{mul}} = (c_{i,j})_{0\le i,j\le k} = \lfloor (t/q) \cdot \overline{\mathsf{ct}} \otimes \overline{\mathsf{ct}}' \rceil \pmod q$. Then, it satisfies following relation: $\langle \overline{\mathsf{ct}}_{\mathsf{mul}}, (1, \overline{\mathsf{sk}}) \otimes (1, \overline{\mathsf{sk}}) \rangle \approx \Delta \cdot mm' \pmod q$. We claim that if $\overline{\mathsf{ct}}_{\mathsf{relin}} \leftarrow \mathtt{MGHE.Relin}(\{\mathsf{jrlk}_j\}_{1\le j\le k}; \overline{\mathsf{ct}}_{\mathsf{mul}})$, then the output ciphertext $\overline{\mathsf{ct}}_{\mathsf{relin}} = (c_0^*, \dots, c_k^*)$ satisfies $c_0^* + \sum_{1\le j\le k} c_j^* \cdot s_j \approx \sum_{0\le i,j\le k} c_{i,j} \cdot s_i s_j$ and thereby is a valid encryption of $mm'$.

First, we have

$$c_0^* + \sum_{1\le j\le k} c_j^* \cdot s_j = c_{0,0} + \sum_{1\le j\le k} (c_{0,j} + c_{j,0}) \cdot s_j + \sum_{1\le i,j\le k} (c_{i,j} \boxdot \mathbf{d}_i) \cdot s_j + \sum_{1\le i\le k} c_i'' \boxdot (\mathbf{v}_i + s_i \cdot \mathbf{u})$$

where $c_i'' = \sum_{1\le j\le k} c_{i,j} \boxdot \mathbf{b}_j$ from the definition of Alg. 2.

We also consider the properties $s_j \cdot \mathbf{d}_i \approx -r_i s_i \cdot \mathbf{a} + s_i s_j \cdot \mathbf{g} \approx r_i \cdot \mathbf{b}_j + s_i s_j \cdot \mathbf{g} \pmod q$ and $\mathbf{v}_i + s_i \cdot \mathbf{u} \approx -r_i \cdot \mathbf{g} \pmod q$ of the joint public keys and deduce the following equations:

$$\sum_{1\le i,j\le k} (c_{i,j} \boxdot \mathbf{d}_i) \cdot s_j \approx \sum_{1\le i,j\le k} r_i \cdot (c_{i,j} \boxdot \mathbf{b}_j) + \sum_{1\le i,j\le k} c_{i,j} \cdot s_i s_j \pmod q,$$
$$\sum_{1\le i\le k} c_i'' \boxdot (\mathbf{v}_i + s_i \cdot \mathbf{u}) \approx -\sum_{1\le i\le k} r_i \cdot c_i'' = -\sum_{1\le i,j\le k} r_i \cdot (c_{i,j} \boxdot \mathbf{b}_j) \pmod q.$$

Putting them all together, we obtain

$$c_0^* + \sum_{1\le j\le k} c_j^* \cdot s_j \approx c_{0,0} + \sum_{1\le j\le k} (c_{0,i} + c_{i,0}) \cdot s_j + \sum_{1\le i,j\le k} c_{i,j} \cdot s_i s_j = \sum_{0\le i,j\le k} c_{i,j} \cdot s_i s_j \pmod q$$

which completes the correctness proof of the relinearization algorithm.

The idea of homomorphic automorphism for MPHE can be also extended to the multi-group case. Given a multi-group ciphertext $\overline{\mathsf{ct}} = (c_0, \dots, c_k)$ linked to $k$ groups $I_1, \dots, I_k$, the joint automorphism key of $I_j$ is used to perform the key-switching procedure of the $j$-th entry $\psi(c_j)$ during the homomorphic evaluation of $\psi \in \mathcal{Gal}(K/\mathbb{Q})$.

• $\mathtt{MGHE.Auto}(\mathsf{jak}_1, \dots, \mathsf{jak}_k; \overline{\mathsf{ct}})$: Given a ciphertext $\overline{\mathsf{ct}} = (c_0, c_1, \dots, c_k)$ and the joint automorphism keys $\mathsf{jak}_j = \mathbf{h}_j$ for $1 \le j \le k$, compute and return the ciphertext $\overline{\mathsf{ct}}_{\mathsf{aut}} = (c_0', c_1', \dots, c_k')$ where $c_0' = \psi(c_0) + \sum_{1\le j\le k} (\psi(c_j) \boxdot \mathbf{h}_j) \pmod q$ and $c_j' = \psi(c_j) \boxdot \mathbf{k} \pmod q$ for $1 \le j \le k$.

**Algorithm 3** Multiplication procedure of MGHE with homomorphic gadget decomposition

**Input:** $\overline{\mathsf{ct}} = (c_i)_{0\le i\le k}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le k}$, $\mathsf{jrlk}_j = (\mathbf{b}_j, \mathbf{d}_j, \mathbf{v}_j)$ for $1 \le j \le k$.
**Output:** $\overline{\mathsf{ct}}_{\mathsf{relin}} = (c^*_j)_{0\le j\le k} \in R_q^{k+1}$.

$c^*_0 \leftarrow \lfloor (t/q) \cdot c_0 c'_0 \rceil \pmod q$
**for** $1 \le j \le k$ **do**
  $c^*_j \leftarrow \lfloor (t/q) \cdot c_0 c'_j + c_j c'_0 \rceil \pmod q$
**end for**
$\mathbf{z} \leftarrow \sum_{1\le i\le k} \tilde{\mathbf{g}}^{-1}(c_i) \odot \mathbf{d}_i \pmod q$
$\mathbf{w} \leftarrow \sum_{1\le j\le k} \tilde{\mathbf{g}}^{-1}(c'_j) \odot \mathbf{b}_j \pmod q$
**for** $1 \le j \le k$ **do**
  $c^*_j \leftarrow c^*_j + c'_j \tilde{\boxdot} \mathbf{z} \pmod q$
**end for**
**for** $1 \le i \le k$ **do**
  $(c^*_0, c^*_i) \leftarrow (c^*_0, c^*_i) + (c_i \tilde{\boxdot} \mathbf{w}) \boxdot (\mathbf{v}_i, \mathbf{u}) \pmod q$
**end for**

**Correctness of homomorphic automorphism.** We show below the correctness of multi-group homomorphic automorphism algorithm:

$$\begin{aligned} c'_0 + \sum_{1\le j\le k} c'_j \cdot s_j &= \psi(c_0) + \sum_{1\le j\le k} \psi(c_j) \boxdot (\mathbf{h}_j + s_j \cdot \mathbf{k}) \\ &\approx \psi(c_0) + \sum_{1\le j\le k} \psi(c_j) \cdot \psi(s_j) = \psi(c_0 + \sum_{1\le j\le k} c_j \cdot s_j) \pmod q \end{aligned}$$

where $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$ and $\overline{\mathsf{ct}}_{\mathsf{aut}} = (c'_0, \ldots, c'_k) \leftarrow \mathtt{MGHE.Auto}(\mathbf{h}_1, \ldots, \mathbf{h}_k; \overline{\mathsf{ct}})$.

**Bootstrapping.** For a fixed parameter set, the BFV and CKKS schemes can support the evaluation of circuits with a limited depth due to the reduction of ciphertext modulus or the noise growth induced from homomorphic operations. Bootstrapping is a method to refresh a ciphertext and recover its computational capability. From the technical point of view, bootstrapping is done by homomorphically evaluating the decryption circuit of HE. The known bootstrapping methods of BFV or CKKS (e.g. [16, 18, 13]) follow a similar workflow that includes arithmetic operations and linear transformations, which can be represented using basic arithmetics and homomorphic automorphisms. As these basic operations are supported in our MGHE schemes, the bootstrapping procedure can be also performed in a similar manner.

**Asymptotically faster multiplication.** Recent research [33] has enhanced the multiplication of BFV and CKKS in MKHE to achieve a linear time complexity. They leverage a newly proposed concept called *homomorphic gadget decomposition*, which satisfies $\langle \mathbf{g}^{-1}(a) \odot \mathbf{g}^{-1}(b), \mathbf{g} \rangle = ab \pmod q$ for $a, b \in R_q$, to replace the term $\mathbf{g}^{-1}(c_{i,j})$ with $\mathbf{g}^{-1}(c_i) \odot \mathbf{g}^{-1}(c'_j)$. As our MGHE is a natural extension of MKHE, we can directly adopt their algorithm to both BFV and CKKS. Notably, their multiplication in BFV entails additional (homomorphic) gadget decomposition $\tilde{\mathbf{g}}^{-1} : R_{\tilde{q}} \to R^{\tilde{d}}$ on $\tilde{q} := q^2$ with a gadget vector $\mathbf{g}^{-1} \in R_{\tilde{q}}^{\tilde{d}}$ and the corresponding external product $c \tilde{\boxdot} \mathbf{v} = \langle \tilde{\mathbf{g}}^{-1}(c), \mathbf{v} \rangle \pmod{\tilde{q}}$. We briefly describe our updated algorithm below and refer the reader to [33] for further details.

• $\mathtt{MGHE.Mult}(\mathsf{jrlk}_1, \ldots, \mathsf{jrlk}_k; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')$: Given two multi-group ciphertexts $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$, $\overline{\mathsf{ct}}' = (c'_0, \ldots, c'_k)$ and $k$ joint relinearization keys $\mathsf{jrlk}_1, \ldots, \mathsf{jrlk}_k$, run Alg. 3 and return the ciphertext $\overline{\mathsf{ct}}_{\mathsf{relin}} = (c^*_0, \ldots, c^*_k)$.

### 4.4 Security

In this section, we show that our MGHE scheme achieves a semantic security that we defined in Section 4.1 under the RLWE assumption.

**Lemma 2 (Security of MGHE).** *The MGHE scheme described above is semantically secure under the RLWE assumption with parameter* $(n, q, \chi, \sigma)$.

*Proof.* Let $I_i$ be sets such that $I = \cup_{0 \leq i \leq k} I_i$ and $H = I \backslash A$ for any set $A \subsetneq I$. We define some hybrid games as follows:

- **Game 0:** This is a real world execution of the security game defined in Definition 6.
- **Game 1:** It is similar to **Game 0**, but the challenger samples $[\mathsf{pk}]_i$ uniformly at random from $R_q^{d \times 4}$ for $i \in H$.
- **Game 2:** It is similar to **Game 1**, but the challenger encrypts 0 instead of $m_b$.

The computational indistinguishability of **Game 0** and **Game 1** is directly inferred from the indistinguishability of **Game 0** and **Game 1** of MPHE, which is described in proof of Lemma 1. This is because the public key used in our scheme is identical to the one used in MPHE.

In both **Game 1** and **Game 2**, the adversary sends a group index $j$ to the challenger in the security game. The encryption key $\mathbf{b}[0]$ used in these games is given by $\mathbf{b}[0] = \sum_{i \in I_j \cap A} [\mathbf{b}]_i[0] + \sum_{i \in I_j \cap H} [\mathbf{b}]_i[0]$. Since $I_j \cap H$ is non-empty and each $[\mathbf{b}]_i$ is uniformly sampled from $R_q^d$ for all $i \in H$, $\mathbf{b}[0]$ is computationally indistinguishable from a uniform random variable over $R_q$. Thus, under the RLWE assumption, the encryptions of 0 and $m_b$ in both games are also computationally indistinguishable. Therefore, the difference in advantage between these two games is negligible.

According to the aforementioned reasons, we can conclude that the advantage of the adversary in **Game 0** is negligible. Since **Game 0** is a real world-execution game with the MGHE scheme, our MGHE scheme achieves semantic security against semi-malicious corruptions. □

## 5 Constructing MPC from MGHE

The MGHE sheme, being a generalization of both the MKHE and MPHE primitives, can serve as a drop-in replacement for these primitives in any application built with them. As a result, MGHE can be effectively utilized in general 2-round MPC computation [39], outsourced computation applications [38], and distributed machine learning setups [23]. Additionally, it can be employed as a building block in MPC protocols that require varying number of parties [21].

### 5.1 Overview

MPHE and MKHE are both viable options for building an MPC protocol [35, 38, 39], but each has limitations that restrict their usefulness in certain applications. For example, MPHE-based MPC protocols require parties to communicate with each other to generate a shared key. On the other hand, MKHE schemes are more time and space intensive than MPHE because ciphertexts expand as they interact with other ciphertexts under different keys. Thus, an MGHE scheme that integrates the strengths of both these schemes can be used to construct round-efficient MPC protocols. In Figure 2, we describe a high-level structure of an MPC computation in three phases. Here, we assume three entities consisting of key owners, data owners, and a cloud server.

- **[Phase I] Setup:** In the first step of the protocol, key owners generate their key pairs and broadcast the public keys. We can treat this step as an offline phase since these procedures have to be run only once and each party is able to produce a key pair independently. When Phase I is ended, a joint public key is built publicly by summing up the individual public keys without any interaction between the parties.
- **[Phase II] Encryption:** After encrypting inputs with the joint encryption key, the ciphertexts are provided to the server which may be an external entity such as a cloud service provider. In general, semi-honest cloud service providers or parties themselves in MPC may play the role of computing party. When Phase II is ended, the circuit is evaluated using the homomorphic properties of the encryption scheme and thus does not require any interaction.
- **[Phase III] Decryption:** When the evaluation is over, we use an interactive protocol known as distributed decryption to securely decrypt the result without revealing the secret key of each party. In the protocol, each party partially decrypts the ciphertext using its own secret key with noise smudging technique [5], and the output message is obtained by adding all of the partially decrypted results.

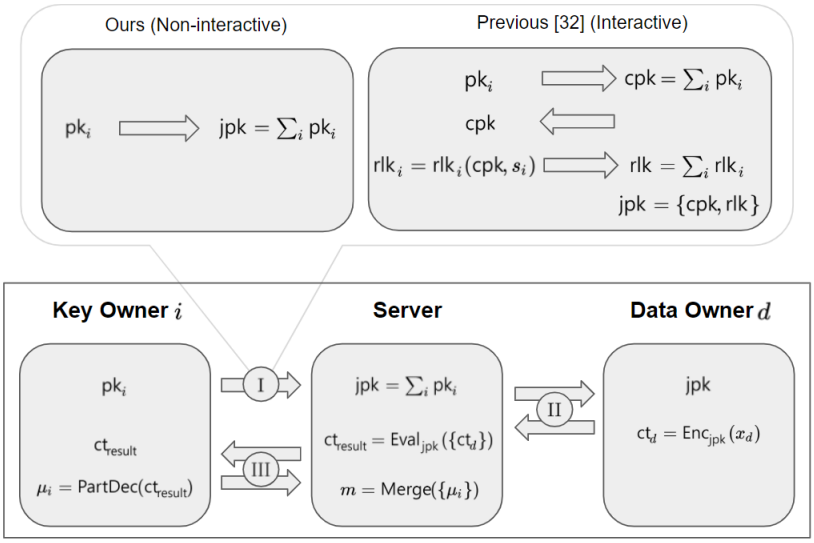


**Fig. 2.** MPC protocol using MGHE and previous work [38]. In [38], cpk and rlk represent the common public key for encryption and evaluation key, respectively. In our MGHE scheme, these keys can be obtained from the joint public key jpk directly.

**Implication of Non-interactive Key Generation.** Recall that all prior MPHE yields multi-round key generation in Phase I due to the quadratic structure of the evaluation key with respect to the individual secret keys. In the MPC protocol derived from the previous MPHE, each party broadcasts twice for the key generation: (1) individual encryption key to generate the joint encryption key and (2) individual evaluation key, which is constructed using the joint encryption key, to generate the joint evaluation key. In our scheme, the novel refactoring of the evaluation key enables the parties to broadcast their keys only once. Each party broadcasts the individual public key, which implicitly contains the shares of the evaluation key. Then, the joint public key is generated publicly to be used for encryption and evaluation. By sharing the individual key pair in the first round itself, each party does not require interaction with other parties in the rest of the process (and can be offline until the decryption process). Thus our setup phase is non-interactive in the sense of Non-Interactive MPC [8, 29] that each party independently and asynchronously broadcasts a single message.

The advantages of this non-interactivity are even more pronounced when a key owner belongs to multiple groups. For example, in the MPC protocol with interactive setup, a key owner must join several key generation protocols to generate joint public keys corresponding to the groups containing the party. Moreover, all parties in the group have to participate simultaneously since the key generation requires communications between the parties. However, in the non-interactive setup, the server or the parties can generate joint public keys after each party broadcasts its own public key without any interaction with other parties. Therefore, we can achieve better efficiency since there is no need to participate in the key generation protocol multiple times and each party can broadcast its own key at any time before generating the joint public key.

### 5.2 MPC Protocol Secure Against Semi-Malicious Corruptions

We provide a concrete MPC protocol in Figure 3 for a polynomial-time deterministic circuit $C$. The correctness of the protocol follows from the correctness of the MGHE construction. In this section, we prove the protocol's security against a semi-malicious adversary. Note that a semi-malicious adversary follows the honest protocol specification with arbitrary values for their random coins [5, 39, 35].

To prove the security of the MPC protocol from MGHE, we begin by demonstrating the simulation security of the distributed decryption process in MGHE. For a circuit $C$, let us denote by $B_C$ an error

**Setup:** A public parameter $\mathsf{pp}$ is generated by $\mathtt{MGHE.Setup}(1^\lambda)$. All parties share the same parameter set.

**Input:** A circuit $C : \mathcal{M}^L \to \mathcal{M}$ where $L$ is the number of inputs. The inputs $x_1, x_2, \ldots, x_L$ are held among the parties.

**The Protocol**

**Phase I:** Let $I$ be the set of parties. Each party $i \in I$ generates a key pair $([\mathsf{sk}]_i, [\mathsf{pk}]_i) \leftarrow \mathtt{MGHE.IndKeyGen}(i)$ and an automorphism key $[\mathsf{ak}]_i \leftarrow \mathtt{MGHE.IndAutKeyGen}([\mathsf{sk}]_i)$, then broadcasts $([\mathsf{pk}]_i, [\mathsf{ak}]_i)$.

**Phase II:**

- Now anybody can compute the joint public and automorphism keys of an arbitrary group. We suppose that the joint keys of $k$ groups $I_1, I_2, \ldots, I_k \subseteq I$ are generated as follows:

$$\mathsf{jpk}_j \leftarrow \mathtt{MGHE.JointKeyGen}(\{[\mathsf{pk}]_i : i \in I_j\}),$$
$$\mathsf{jak}_j \leftarrow \mathtt{MGHE.JointAutKeyGen}(\{[\mathsf{ak}]_i : i \in I_j\}).$$

  We denote by $\mathsf{jek}_j$ the encryption key of $I_j$.
- For each $1 \le \ell \le L$, the party with input $x_\ell$ encrypts it using a joint public key $\mathsf{jpk}_j$ for some $1 \le j \le k$ and broadcasts the ciphertexts $\mathsf{ct}_\ell \leftarrow \mathtt{MGHE.Enc}(\mathsf{jek}_j; x_\ell)$.

**Phase III:**

- The circuit $C$ is evaluated as following:

$$\overline{\mathsf{ct}} \leftarrow \mathtt{MGHE.Eval}(\{\mathsf{jpk}_j\}_{1\le j\le k}, \{\mathsf{jak}_j\}_{1\le j\le k}; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_L).$$

- Finally, the parties concurrently take part in the distributed decryption protocol with the error parameter $\sigma'$ to deduce the output $m$:

$$m \leftarrow \mathtt{MGHE.DistDec}(I, \sigma'; \overline{\mathsf{ct}})$$

**Output:** Return the decrypted message $m$.

**Fig. 3.** $\pi_C$: MPC protocol for a circuit $C$ using MGHE

bound of a ciphertext obtained by evaluating the circuit $C$ over fresh ciphertexts. Given $B_C$, we can guarantee the correctness and simulation security of the distributed decryption if $\sigma'$ is exponentially larger than the bound $B_C$.

**Lemma 3 (Correctness of Distributed Decryption).** *Let $n$ be the number of parties in $I = \cup_{1\le j\le k} I_j$ and $B_{\sigma'}$ be bound of the samples from $D_{\sigma'}$ with non-negligible probability. If $q \ge 2nt(B_C + B_{\sigma'})$, then the distributed decryption procedure* `MGHE.DistDec` *satisfy correctness.*

*Proof.* Given the partial decryptions $[\mu]_i$ of parties $i \in I$, we have

$$\begin{aligned} c_0 + \sum_{i\in I} \mu_i &= c_0 + \sum_{i\in I} \left( \sum_{1\le j\le k, i\in I_j} c_j \right) \cdot [s]_i + \sum_{i\in I} [e']_i \\ &= \Delta \cdot m + e + e' \end{aligned}$$

where $e$ is bounded by $nB_C$ and $e' = \sum_{i\in I}[e']_i$ is bounded by $nB_{\sigma'}$. Since $q \ge 2nt(B_C + B_{\sigma'})$, we have $|e + e'| \le q/2t$, which ensures the correctness. □

Below, we show that the simulation security of the distributed decryption and the MPC protocol. We consider a scenario where $n-1$ out of $n$ parties are corrupted.

**Lemma 4 (Security of Distributed Decryption).** *If $\sigma' > 0$ is a real number such that the samples from $D_{\sigma'}$ are larger than $2^\lambda B_C$ without negligible probability, then the distributed decryption procedure* `MGHE.DistDec` *achieves statistical simulation security against any static semi-malicious adversary corrupting exactly $n-1$ parties.*

*Proof.* Let a party $h$ be the only honest party. We construct a simulator $\mathcal{S}$ against the adversary $\mathcal{A}$ which has an access to the inputs and secret keys of all parties except $h$ and receives the output message $m$ from the ideal functionality. For given evaluated ciphertext $\overline{\mathsf{ct}} = (c_0, \ldots, c_L)$, the simulator computes and publishes the simulated partial decryption $[\mu]'_h$ of the honest party $h$ using a smudging error $[e']_h^{sm} \leftarrow D_{\sigma'}$:

$$[\mu]'_h = \Delta \cdot m + [e']_h^{sm} - \sum_{i\ne h} \gamma_i - c_0 \tag{1}$$

where $\gamma_i = \left(\sum_{1\le j\le k, i\in I_j} c_j\right) \cdot [s]_i \pmod q$ for $i \ne h$.

Then, the partial decryption of $h$ is generated from the partial decryptions of corrupted parties and the output message as $\Delta\cdot m + [e']_h^{sm} - \sum_{i\ne h}\gamma_i - c_0$. On the other hand, the real partial decryption also can be written as $\Delta\cdot m + e + [e']_h^{sm} - \sum_{i\ne h}\gamma_i - c_0$ where $e$ is the noise in the ciphertext $\overline{\mathsf{ct}}$. By the smudging lemma [5], the distributions of $[e']_h^{sm}$ and $e + [e']_h^{sm}$ are statistically indistinguishable. It concludes that the simulated partial decryption and the real partial decryption are statistically indistinguishable. □

**Theorem 1 (Security of MPC protocol).** *Given a poly-time computable deterministic circuit $C$ with $L$ inputs, the protocol $\pi_C$ described in Figure 3 UC-realizes the circuit $C$ against any static semi-malicious adversary corrupting exactly $n-1$ parties.*

*Proof.* Let a party $h$ be the only honest party. We construct a simulator $\mathcal{S}$ against the adversary $\mathcal{A}$ as follows.

**The Simulator.** In Phase I, the simulator samples the public key of $h$ from uniform distribution over $R_q^{d\times 4}$ instead of `MGHE.IndKeyGen`$(h)$. The simulator also plays Phase II honestly on behalf of the honest party, but encrypts 0 instead of the real input from $h$, if any. As the simulator has access to the inputs and secret keys of all parties except $h$ from the witness tape, the simulator can evaluate the circuit $C$ on ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_L$ and obtain the resulting ciphertext $\overline{\mathsf{ct}}$. In addition, it also receives the output message $m$ from the ideal functionality. In Phase III, the simulator computes the partial decryption for the party $h$ as same as the simulator introduced in the security proof of Lemma 4.

Now, we define some hybrid games and prove the computational indistinguishability between the real and ideal worlds.

- **The game** $REAL_{(\pi,\mathcal{A},\mathcal{Z})}$: An execution of the protocol $\pi$ in the real world with environment $\mathcal{Z}$ and semi-malicious adversary $\mathcal{A}$.
- **The game** $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$: This is the same as $REAL_{(\pi,\mathcal{A},\mathcal{Z})}$ except the output of partial decrpytion of $h$. In Phase III, it publishes the simulated partial decryption which is computed via (1).
- **The game** $HYB^2_{(\pi,\mathcal{A},\mathcal{Z})}$: This is similar to $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$, but in Phase II the party $h$ encrypts 0 instead of the real input if any.
- **The game** $IDEAL_{(\mathcal{F},\mathcal{S},\mathcal{Z})}$: It executes the MPC protocol with the simulator $\mathcal{S}$. The difference from $HYB^2_{(\pi,\mathcal{A},\mathcal{Z})}$ is that the public key of $h$ is sampled from a uniform distribution over $R_q^{d\times 4}$ instead of the individual key generation algorithm $\texttt{MGHE.IndKeyGen}(h)$ in Phase I.

From the above games, we consider the following claims.

**Claim 1**. $REAL_{(\pi,\mathcal{A},\mathcal{Z})}$ and $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$ are statistically indistinguishable.

*Proof.* According to the description of the simulator, the partial decryption of $h$ in the game $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$ is generated from the partial decryptions of corrupted parties and the output message as $\Delta\cdot m+[e']^{sm}_h-\sum_{i\neq h}\gamma_i-c_0$, while the real partial decryption also can be written as $\Delta\cdot m+e+[e']_h-\sum_{i\neq h}\gamma_i-c_0$ where $e$ is the noise in the ciphertext $\overline{\mathsf{ct}}$. By Lemma 4, the distributions of $[e']^{sm}_h$ and $e+[e']_h$ are statistically indistinguishable. This indicates that $REAL_{(\pi,\mathcal{A},\mathcal{Z})}$ and $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$ are also statistically indistinguishable.

**Claim 2**. $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$, $HYB^2_{(\pi,\mathcal{A},\mathcal{Z})}$, and $IDEAL_{(\mathcal{F},\mathcal{S},\mathcal{Z})}$ are computationally indistinguishable.

*Proof.* The differences in three games correspond to the differences in **Game 0**, **Game 1**, and **Game 2** of Lemma 2. In detail, the difference between $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$ and $HYB^2_{(\pi,\mathcal{A},\mathcal{Z})}$ is that the party $h$ encrypts the real input in $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$ while it encrypts 0 in the game $HYB^2_{(\pi,\mathcal{A},\mathcal{Z})}$, if any. Furthermore, the difference between $HYB^2_{(\pi,\mathcal{A},\mathcal{Z})}$ and $IDEAL_{(\mathcal{F},\mathcal{S},\mathcal{Z})}$ is in the public key $\mathsf{pk}_h$. In $HYB^2_{(\pi,\mathcal{A},\mathcal{Z})}$, $\mathsf{pk}_h$ is a valid public key generated by $h$ while it is sampled from a uniform distribution over $R_q^{d\times 4}$ in the game $IDEAL_{(\mathcal{F},\mathcal{S},\mathcal{Z})}$. Thus, by Lemma 2, the three games $HYB^1_{(\pi,\mathcal{A},\mathcal{Z})}$, $HYB^2_{(\pi,\mathcal{A},\mathcal{Z})}$, and $IDEAL_{(\mathcal{F},\mathcal{S},\mathcal{Z})}$ are computationally indistinguishable.

According to the claims, we conclude that the MPC protocol $\pi_C$ is secure in the semi-malicious model against $n-1$ corrupted parties. □

To handle the arbitrary number of corruptions, we can establish security proof by constructing the extended protocol as outlined in [39]. In addition, we can transform our MPC protocol, which is secure against semi-malicious attackers, into a protocol that offers security against malicious corruptions without introducing any additional rounds. This transformation can be achieved by leveraging non-interactive zero-knowledge proofs, as described in [5].

## 6 Experimental Results

We implemented our MGHE scheme based on BFV and CKKS. The source code is written in GO programming language and is built on Lattigo [1] version 2.3.0. We conducted experiments on a system equipped with Intel(R) Core(TM) i9-10900 CPU @ 2.80GHz and 64GB RAM. In our implementation, the key distribution $\chi$ samples the coefficients from the ternary set $-1, 0, 1$ with equal probabilities of 0.25 for $-1$ and 1, and a probability of 0.5 for 0. The error parameter is set to $\sigma = 3.2$.

### 6.1 Basic Operations

Table 1 shows the execution time of multiplication of our MGHE scheme. The experiment was conducted using two different parameter sets: $(N, \lceil \log pq \rceil) = (2^{14}, 438)$ and $(2^{15}, 880)$ where $p$ is a special modulus. Both parameter sets ensure a security level of at least 128 bits [2]. Ours shows the performance of Alg. 2 where a minor optimization is introduced and Ours$^+$ shows that of Alg. 3 which applies the technique

of [33]. As our MGHE supports computation on groups of parties, we measured the performance varying the number of groups.

According to Table 1, the number of parties in groups does not affect the execution time in both BFV and CKKS. It is because, as described in Section 3.1, the size of ciphertext does not expand even if there are many parties in the group. On the other hand, the execution time depends on the dimension of the base ring and the number of groups participating in the evaluation. As the dimension of the plaintext increases, the ciphertext modulus increases and eventually affects the execution time of arithmetic operations. Moreover, according to Section 4.3, relinearization (or automorphism) requires more external products when there are more groups involved in the evaluation. Therefore, the execution time increases with the number of groups.

| $N$ | $n$ | $k$ | Mult + Relin | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | BFV | | | | CKKS | | | |
| | | | Ours | Ours$^+$ | [15] | [38] | Ours | Ours$^+$ | [15] | [38] |
| $2^{14}$ | 1 | 1 | 78.7 | 118.4 | 84.1 | 32.6 | 51.6 | 72.9 | 51.2 | 17.7 |
| | 2 | 1 | 77.9 | 120.4 | - | 33.4 | 50.8 | 75.3 | - | 17.1 |
| | | 2 | 173.4 | 224.4 | 196.2 | - | 122.8 | 133.4 | 139.7 | - |
| | 4 | 1 | 80.1 | 121.6 | - | 32.9 | 51.6 | 76.9 | - | 17.4 |
| | | 2 | 175.3 | 224.1 | - | - | 124.5 | 133.6 | - | - |
| | | 4 | 476.4 | 420.8 | 589.7 | - | 335.8 | 250.5 | 450.7 | - |
| | 8 | 1 | 78.4 | 118.9 | - | 33.3 | 50.8 | 77.2 | - | 17.2 |
| | | 2 | 178.1 | 223.5 | - | - | 123.6 | 135.1 | - | - |
| | | 4 | 461.4 | 422.7 | - | - | 337.1 | 249.4 | - | - |
| | | 8 | 1473.0 | 811.7 | 2014.2 | - | 1081.9 | 495.9 | 1600.7 | - |
| $2^{15}$ | 1 | 1 | 595.9 | 1036.8 | 605.5 | 202.0 | 414.2 | 642.5 | 404.1 | 165.7 |
| | 2 | 1 | 593.1 | 1019.4 | - | 201.5 | 412.9 | 640.7 | - | 170.8 |
| | | 2 | 1308.3 | 1929.8 | 1477.9 | - | 1014.9 | 1094.3 | 1089.7 | - |
| | 4 | 1 | 599.2 | 1024.4 | - | 204.2 | 413.7 | 643.1 | - | 164.6 |
| | | 2 | 1324.9 | 1945.7 | - | - | 1008.4 | 1100.6 | - | - |
| | | 4 | 3556.5 | 4006.8 | 4582.9 | - | 2844.5 | 2177.3 | 3553.1 | - |
| | 8 | 1 | 593.2 | 1033.9 | - | 202.7 | 413.3 | 645.5 | - | 168.7 |
| | | 2 | 1319.8 | 1987.1 | - | - | 1011.4 | 1103.5 | - | - |
| | | 4 | 3515.2 | 3954.5 | - | - | 2825.1 | 2147.2 | - | - |
| | | 8 | 10681.1 | 6871.5 | 15257.5 | - | 9008.9 | 4449.3 | 13052.8 | - |

**Table 1.** Performance of our MGHE schemes, the MKHE scheme by Chen et al. [15], and the MPHE scheme by Mouchet et al. [38]: execution times to operate homomorphic multiplication (Mult + Relin), taken in milliseconds (ms). $N$ denotes the dimension of base ring, $n$ and $k$ denote the number of the associated parties and groups (keys), respectively, to the ciphertext. Ours$^+$ refers to our MGHE scheme combined with the technique of [33].

We also present the performance of the MKHE scheme [15] and MPHE scheme [38] for comparison. Since MKHE and MPHE are instances of MGHE where each group consists of a single party and single group, respectively, the MKHE scheme and MPHE scheme have its results in Table 1 only when $n = k$ and $k = 1$, respectively. Upon comparing the performance of MKHE and our method, the table shows that our multiplication algorithm exhibits slightly faster operation times than previous MKHE. This is due to our approach, as explained in Section 4.2, where we reduce the number of external products during the relinearization. Moreover, in the case of a large number of groups, it is even faster than our method when we apply the recent technique introduced in [33]. We also remark that although MPHE shows

| Data | | | Performance | |
|---|---|---|---|---|
| MNIST | # Training Samples | 11,982 | Training Time (single-key) | 75 min |
| | # Validation Samples | 1,984 | Validation Time (2-key) | 8 min |
| | # Features | 196 | Accuracy | 90.6% |
| | # Iterations | 5 | | |
| | Batch Size | 1,024 | | |
| | Learning Rate | 1.0 | | |

**Table 2.** Result of logistic regression on MNIST dataset. The parameter set $(N, \lceil \log pq \rceil) = (2^{16}, 1761)$ is used.

better performance than other methods, it requires interactions among the parties before the evaluation to generate the joint public key.

### 6.2 Application to Machine Learning Service

One possible application of MGHE is to enable a secure workflow of machine learning comprising multiple model owners. Consider a scenario where a model is trained using datasets owned by multiple providers. If the data owners can be determined before training, the MPHE scheme would be a reasonable solution for privacy-preserving training of the model. In the case of inference, however, the clients may not be determined aforehand and the model may deal with multiple independent clients. In this case, employing an MKHE scheme for inference becomes more practical due to its enhanced flexibility. The model and the client's data are encrypted using distinct keys owned by the model owners and the client, respectively. These encrypted inputs are then transformed into encrypted multi-key ciphertext under their respective keys. After performing inference in 2-key MKHE, the output can be obtained through distributed decryption involving both the model owners and the client. As a result, our MGHE interpolates between MPHE (where the model is trained) and MKHE (where the inference is performed) so that the entire process can be done in the same encryption scheme.

Several researches [43, 26, 42] uses only MPHE to realize secure machine learning. Similar to our approach, the owners of the training data generate the joint key and train the model in the MPHE scheme. However, in the inference stage, a client's data is encrypted using the model owners' joint key and then evaluated to produce the output. Subsequently, the output is key-switched to be encrypted under the client's key, enabling the client to decrypt it. We notice that this scenario has a drawback that, since the client's data is encrypted under the model owners' key, the model owners can work together to peek the client's data, which further stress the advantage of MGHE in secure machine learning services.

As a proof-of-concept, we implemented a logistic regression model on top of our MGHE scheme. We did not apply the technique of [33] since, in the 2-group case, our multiplication algorithm (Alg. 2) shows better performance than the method employing the aforementioned technique (Alg. 3). Following the model of [31], we solved the classification problem between 3 and 8 of the MNIST dataset [24]. The original images of $28 \times 28$ are compressed to $14 \times 14$ pixels by taking the mean of $2 \times 2$ pixel blocks. The sigmoid function in the interval $[-16, 16]$ is approximated by $y = -0.0002x^3 + 0.0843x + 0.5$. Using the parameter set $(N, \lceil \log pq \rceil) = (2^{16}, 1761)$, our model achieved an accuracy of 90.6% with a training time of 75 minutes and a validation time of 8 minutes. Additional details about the dataset, hyperparameters, and performance of the experiment are provided in Table 2.

## A Construction of MGHE with CKKS

The CKKS supports approximate arithmetic operations for complex numbers. The BFV and CKKS have similar structure, we can easily extend MGHE scheme of the CKKS. The difference is that it adds an error into the plaintext itself and additionally supports the rescaling algorithm to control the size of ciphertext. The ciphertext has a level and it decreases whenever rescaling is performed. To proceed

arithmetics between two ciphertexts, they should have same level and it requires bootstrapping when level is low in order to continue evaluation. We are going to transform MPHE scheme without interactive setup first, and extend it into the MGHE scheme. In both cases, we skip setup, key generation, and joint key generation phase since they are same as BFV. Galois automorphism is also not included since it has same procedure with the BFV. We assume the ciphertext modulus $q = \prod_{i=1}^{L} p_i$ for some integers $p_i$ and denote $q_l = \prod_{i=1}^{l} p_i$.

### A.1 MPHE with Non-Interactive Setup

• $\underline{\mathtt{MP-CKKS.Enc}(\mathsf{jek}; m)}$: Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. For an input message $m \in R_t$, return the ciphertext $\mathsf{ct} = w \cdot \mathsf{jek} + (m + e_0, e_1) \pmod q$.

• $\underline{\mathtt{MP-CKKS.Add}(\mathsf{ct}, \mathsf{ct}')}$: If $\mathsf{ct}$ and $\mathsf{ct}'$ have same level, return $\mathsf{ct}_{\mathsf{add}} = \mathsf{ct} + \mathsf{ct}' \pmod q$. If not, lower the high-level ciphertext to low-level ciphertext before the computation.

• $\underline{\mathtt{MP-CKKS.Mult}(\mathsf{jrlk}; \mathsf{ct}, \mathsf{ct}')}$: If $\mathsf{ct}$ and $\mathsf{ct}'$ have different level, make two ciphertexts have the same level. Given two ciphertexts $\mathsf{ct} = (c_0, c_1)$, $\mathsf{ct}' = (c_0', c_1')$ and a joint relinearization key $\mathsf{jrlk}$, let $\mathsf{ct}_{\mathsf{mul}} = \mathsf{ct} \otimes \mathsf{ct}' = (c_0c_0', c_0c_1' + c_1c_0', c_1c_1')$. Return the ciphertext $\mathtt{MP-CKKS.Relin}(\mathsf{jrlk}; \mathsf{ct}_{\mathsf{mul}})$ where $\mathtt{MP-CKKS.Relin}(\cdot)$ is the relinearization procedure described in Alg. 1.

• $\underline{\mathtt{MP-CKKS.Rescale}(\mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_{q_l}^2$ at level $l$, return $\mathsf{ct}' = (\lfloor p_l^{-1} \cdot c_0 \rceil, \lfloor p_l^{-1} \cdot c_1 \rceil) \in R_{q_l - 1}^2$ which is at level $l - 1$.

• $\underline{\mathtt{MP-CKKS.Dec}(\mathsf{sk}; \mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1)$ and a secret key $\mathsf{sk} = s$, output $m = \langle \mathsf{ct}, \mathsf{sk} \rangle = (c_0 + c_1 \cdot s) \pmod q$.

• $\underline{\mathtt{MP-CKKS.DistDec}(\{[\mathsf{sk}]_i : i \in I\}, \sigma'; \mathsf{ct})}$: Let $\mathsf{ct} = (c_0, c_1)$ be a multi-party ciphertext, $\sigma' > 0$ an error parameter, and $[\mathsf{sk}]_i = [s]_i$ the secret key of party $i \in I$. The distributed decryption protocol consists of the following procedures:

- Partial decryption: Each party $i \in I$ samples $[e']_i \leftarrow D_{\sigma'}$, then computes and publishes $[\mu]_i = c_1 \cdot [s]_i + [e']_i \pmod q$.
- Merge: Compute $m = (c_0 + \sum_{i \in I} [\mu]_i) \pmod q$.

### A.2 Extension to MGHE with CKKS

• $\underline{\mathtt{MG-CKKS.Enc}(\mathsf{jek}; m)}$: For a joint encryption key $\mathsf{jek}$ and a message $m$, return $\mathsf{ct} \leftarrow \mathtt{MP-CKKS.Enc}(\mathsf{jek}; m)$.

• $\underline{\mathtt{MG-CKKS.Add}(\overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: If two given ciphertexts $\overline{\mathsf{ct}}$ and $\overline{\mathsf{ct}}'$ has same level, return the ciphertext $\overline{\mathsf{ct}}_{add} = \overline{\mathsf{ct}} + \overline{\mathsf{ct}}' \pmod q$. If not, modify ciphertexts to have same level before the computation.

• $\underline{\mathtt{MG-CKKS.Mult}(\{\mathsf{jrlk}_j\}_{1 \le j \le k}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Set $\mathsf{ct}$ and $\mathsf{ct}'$ have same level. Let $\overline{\mathsf{ct}} = (c_i)_{0 \le i \le k}$ and $\overline{\mathsf{ct}}' = (c_i')_{0 \le i \le k}$ be two multi-group ciphertexts and $\{\mathsf{jrlk}_j\}_{1 \le j \le k}$ the collection of the joint relinearization keys of groups $I_j$ for $1 \le j \le k$. Compute $\overline{\mathsf{ct}}_{\mathsf{mul}} = (c_{i,j})_{0 \le i,j \le k}$ where $c_{i,j} = c_i c_j' \pmod q$ for $0 \le i, j \le k$. Return the ciphertext $\mathtt{MG-CKKS.Relin}(\{\mathsf{jrlk}_j\}_{1 \le j \le k}; \overline{\mathsf{ct}}_{\mathsf{mul}})$ where $\mathtt{MG-CKKS.Relin}(\cdot)$ is the relinearization procedure described in Alg. 2.

• $\underline{\mathtt{MG-CKKS.Rescale}(\overline{\mathsf{ct}})}$: Given a ciphertext $\overline{\mathsf{ct}} = (c_0, c_1, \ldots, c_k) \in R_{q_l}^{k+1}$ at level $l$, compute $c_i' = \lfloor p_l^{-1} \cdot c_i \rceil$ for $1 \le i \le k$, and return $\overline{\mathsf{ct}}' = (c_0', c_1', \ldots, c_k') \in R_{q_l - 1}^{k+1}$ which is at level $l - 1$.

• $\underline{\mathtt{MG-CKKS.Dec}(\{\mathsf{sk}_j\}_{1 \le j \le k}; \overline{\mathsf{ct}})}$: Given a ciphertext $\overline{\mathsf{ct}} = (c_0, c_1, \ldots, c_k)$ and joint secret keys $\mathsf{sk}_j = s_j$ for $1 \le j \le k$, return $m = \langle \mathsf{ct}, \mathsf{sk} \rangle = (c_0 + \sum_{1 \le j \le k} c_i \cdot s_j) \pmod q$.

• $\underline{\mathtt{MG-CKKS.DistDec}(\{[\mathsf{sk}_j]_i\}_{1 \le j \le k, i \in I_j}, \sigma'; \overline{\mathsf{ct}})}$: Let $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$ be a multi-group ciphertext corresponding to the set of groups $\{I_1, \ldots, I_k\}$ and $[\mathsf{sk}]_i = [s]_i$ be the secret of party $i \in I_j$.

- Partial decryption: For $1 \le j \le k$, each party $i \in I_j$ samples $[e_j']_i \leftarrow D_{\sigma'}$, then computes and publishes $[\mu_j]_i = c_j \cdot [s]_i + [e_j']_i \pmod q$.
- Merge: Compute $m = (c_0 + \sum_{1 \le j \le k} \sum_{i \in I_j} [\mu_j]_i) \pmod q$.

## B Noise analysis

Before estimating a noise growth, we specify some distributions for sampling randomness or errors. Let the key distribution $\chi$ be the distribution where each coefficient is sampled from the set $\{0, \pm 1\}$ with probability 0.25 for each of $-1$ and 1 and with probability 0.5 for 0. Set the error distribution $\psi$ be the discrete Gaussian distribution of variance $\sigma^2$. We also assume that the coefficients of the polynomials are independent zero-mean random variables with the same variances. We denote by $\mathsf{Var}(a) = \mathsf{Var}(a_i)$ the variance of coefficients for random variable $a = \sum_i a_i \cdot X^i$ over the ring $R$. Then the variance of the product $c = a \cdot b$ of two polynomials with degree $n$ can be represented as $\mathsf{Var}(c) = n \cdot \mathsf{Var}(a) \cdot \mathsf{Var}(b)$ if $a$ and $b$ are independent. Similarly, we define variance for a vector $\mathbf{a} \in R^d$ of random variables as $\mathsf{Var}(\mathbf{a}) = \frac{1}{d}\sum_{i=1}^{d} \mathsf{Var}(\mathbf{a}[i])$. We also assume that each ciphertext behaves as if it is a uniform random variable over $R_q^{k+1}$. We analyze the noise growth of $k$-group case, each comprising $N_i$ parties for $1 \le i \le k$.

### B.1 Encryption

Recall that the encryption $\mathsf{ct} = (c_0, c_1) \in R_q^2$ of $m \in R_p$ is $\mathsf{ct} = t \cdot \mathsf{jek} + (\Delta \cdot m + e_0, e_1) \pmod q$ where $t \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. For $\mathsf{jek} = (\mathbf{b}[0], \mathbf{a}[0]) \in R_q^2$, we remark that $\mathbf{b}[0] + \mathbf{a}[0] \cdot s = \sum_{i \in I} [\mathbf{e}_0]_i[0]$ and each $[\mathbf{e}_0]_i[0]$ is sampled from $D_\sigma$. Then, it satisfies that $c_0 + c_1 \cdot s = \Delta \cdot m + t(\mathbf{b}[0] + \mathbf{a}[0] \cdot s) + (e_0 + e_1 \cdot s) = \Delta \cdot m + (t\sum_{i \in I} [\mathbf{e}_0]_i[0] + e_0 + e_1 \cdot s) \pmod q$. The encryption noise $e_{\mathsf{enc}} = t\sum_{i \in I} [\mathbf{e}_0]_i[0] + e_0 + e_1 \cdot s$ has the variance of $V_{\mathsf{enc}} = \sigma^2 \cdot (\frac{n|I|}{2} + 1 + \frac{n}{2}) \approx \frac{n\sigma^2(|I|+1)}{2}$.

The CKKS scheme has the same encryption error as the BFV scheme. The only difference is that there is no scaling factor $\Delta$ in the result of decryption.

### B.2 Relinearization

In Alg. 2 of Section 4.2, it satisfies that

$$\begin{aligned}\sum_{1 \le i \le k} c_i'' \boxdot (\mathbf{v}_i + s_i \cdot \mathbf{u}) &= -\sum_{1 \le i \le k} r_i \cdot c_i'' + \sum_{1 \le i \le k} c_i'' \boxdot \mathbf{e}_{i,2} \\ &= -\sum_{1 \le i,j \le k} r_i \cdot (c_{i,j} \boxdot \mathbf{b}_j) + \sum_{1 \le i \le k} c_i'' \boxdot \mathbf{e}_{i,2} \pmod q\end{aligned}$$

and

$$\begin{aligned}&\sum_{1 \le i,j \le k} (c_{i,j} \boxdot \mathbf{d}_i) \cdot s_j \\ &= \sum_{1 \le i,j \le k} r_i \cdot (c_{i,j} \boxdot (\mathbf{b}_j - \mathbf{e}_{j,0})) + \sum_{1 \le i,j \le k} s_i s_j \cdot c_{i,j} + \sum_{1 \le i,j \le k} s_j \cdot (c_{i,j} \boxdot \mathbf{e}_{i,1}) \\ &= \sum_{1 \le i,j \le k} r_i \cdot (c_{i,j} \boxdot \mathbf{b}_j) + \sum_{1 \le i,j \le k} s_i s_j \cdot c_{i,j} + \sum_{1 \le i,j \le k} e'_{i,j} \pmod q\end{aligned}$$

where $e'_{i,j} = c_{i,j} \boxdot (s_j \cdot \mathbf{e}_{i,1} - r_i \cdot \mathbf{e}_{j,0})$.

We denote by $V_g = \mathsf{Var}(\mathbf{g}^{-1}(a))$ where $a$ is a uniform random variable over $R_q$. Then, the variance of relinearization error $e_{\mathsf{relin}} = \sum_{1 \le i \le k} c_i'' \boxdot \mathbf{e}_{i,2} + \sum_{1 \le i,j \le k} e'_{i,j}$ is obtained as follows:

$$V_{\mathsf{relin}} = ndV_g\sigma^2 \sum_{1 \le i \le k} N_i^2 + 2n^2 dV_g\sigma^2 k^2 \sum_{1 \le i \le k} N_i^2 \approx 2n^2 dV_g\sigma^2 k \sum_{1 \le i \le k} N_i^2$$

In our implementation, we use RNS-friendly decomposition $R_q = \prod_i R_{p_i}$ such that $p_i$'s have the same bit-size. Here, we have $V_g = \frac{1}{12d}\sum_{i=1}^{d} p_i^2$ for $d = \lceil \log q / \log p_i \rceil$.

### B.3 Multiplication

We again consider $k$-group case, each comprising $N_i$ parties for $1 \leq i \leq k$. Let $\overline{\mathsf{ct}}_1$ and $\overline{\mathsf{ct}}_2$ be the input ciphertexts of messages $m_1$ and $m_2$ respectively. Each ciphertext $\overline{\mathsf{ct}}_i$ satisfies that $\langle \overline{\mathsf{ct}}_i, \overline{\mathsf{sk}} \rangle = q \cdot I_i + \Delta \cdot m_i + e_i$ for $I_i = \lfloor \frac{1}{q} \langle \overline{\mathsf{ct}}_i, \overline{\mathsf{sk}} \rangle \rceil$ and some $e_i$. Here, we have the variance $\mathsf{Var}(I_i) \approx \frac{1}{12}(1+\frac{1}{2}kn) \approx \frac{1}{24}kn$ since $\frac{1}{q} \cdot \overline{\mathsf{ct}}_i$ behaves as an uniform random variable over $\frac{1}{q} \cdot R_q^{k+1}$.

The result of tensor product satisfies that $\langle \overline{\mathsf{ct}}_1 \otimes \overline{\mathsf{ct}}_2, \overline{\mathsf{sk}} \otimes \overline{\mathsf{sk}} \rangle = \langle \overline{\mathsf{ct}}_1, \overline{\mathsf{sk}} \rangle \cdot \langle \overline{\mathsf{ct}}_2, \overline{\mathsf{sk}} \rangle = \Delta^2 \cdot m_1 m_2 + q \cdot (I_1 e_2 + I_2 e_1) + \Delta \cdot (m_1 e_2 + m_2 e_1) + e_1 e_2 \pmod{q \cdot \Delta}$ and for $\overline{\mathsf{ct}}_{\mathsf{mul}} = \left\lfloor \frac{p}{q} \cdot \overline{\mathsf{ct}}_1 \otimes \overline{\mathsf{ct}}_2 \right\rceil$, we have $\langle \overline{\mathsf{ct}}_{\mathsf{mul}}, \overline{\mathsf{sk}} \otimes \overline{\mathsf{sk}} \rangle = \Delta \cdot m_1 m_2 + p \cdot (I_1 e_2 + I_2 e_1) + (m_1 e_2 + m_2 e_1) + \Delta^{-1} \cdot e_1 e_2 + e_{rd}$ where $e_{rd} = \left\langle \frac{p}{q} \cdot \overline{\mathsf{ct}}_1 \otimes \overline{\mathsf{ct}}_2 - \overline{\mathsf{ct}}_{mul}, \overline{\mathsf{sk}} \otimes \overline{\mathsf{sk}} \right\rangle$. That is, the multiplication error is obtained by $e_{\mathsf{mul}} = p \cdot (I_1 e_2 + I_2 e_1) + (m_1 e_2 + m_2 e_1) + \Delta^{-1} \cdot e_1 e_2 + e_{rd}$. From the above equation, the first term $p \cdot (I_1 e_2 + I_2 e_1)$ dominates the whole multiplication error. Therefore, we have the variance of multiplication error by

$$V_{\mathsf{mul}} \approx np^2 \cdot (\mathsf{Var}(I_1)\mathsf{Var}(e_2) + \mathsf{Var}(I_2)\mathsf{Var}(e_1)) \approx \frac{1}{24} kn^2 p^2 (\mathsf{Var}(e_1) + \mathsf{Var}(e_2)).$$

While the relinearization error has a fixed size depending on the parameters, the multiplication error increases by a certain ratio as the computation proceeds. Therefore, the total noise is eventually dominated by the multiplication error unless $(\mathsf{Var}(e_1) + \mathsf{Var}(e_2))$ is very small (e.g. fresh ciphertext).

## References


1. Lattigo v4. Online: `https://github.com/tuneinsight/lattigo` (Aug 2022), ePFL-LDS, Tune Insight SA
2. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption security standard. Tech. rep., HomomorphicEncryption.org, Toronto, Canada (November 2018)
3. Aloufi, A., Hu, P., Wong, H.W., Chow, S.S.: Blindfolded evaluation of random forests with multi-key homomorphic encryption. IEEE Transactions on Dependable and Secure Computing (2019)
4. Ananth, P., Jain, A., Jin, Z., Malavolta, G.: Multi-key fully-homomorphic encryption in the plain model. In: Theory of Cryptography Conference. pp. 28–57. Springer (2020)
5. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold fhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 483–501. Springer (2012)
6. Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: Secure mpc: laziness leads to god. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 120–150. Springer (2020)
7. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full rns variant of fv like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 423–442. Springer (2016)
8. Beimel, A., Gabizon, A., Ishai, Y., Kushilevitz, E., Meldgaard, S., Paskin-Cherniavsky, A.: Non-interactive secure multiparty computation. In: Annual Cryptology Conference. pp. 387–404. Springer (2014)
9. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Annual International Cryptology Conference. pp. 565–596. Springer (2018)
10. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
11. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
12. Brakerski, Z., Perlman, R.: Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: Annual Cryptology Conference. pp. 190–213. Springer (2016)
13. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 34–54. Springer (2019)
14. Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from TFHE. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 446–472. Springer (2019)

15. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 395–412 (2019)
16. Chen, H., Han, K.: Homomorphic lower digits removal and improved fhe bootstrapping. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 315–337. Springer (2018)
17. Chen, L., Zhang, Z., Wang, X.: Batched multi-hop multi-key FHE from Ring-LWE with compact ciphertext extension. In: Theory of Cryptography Conference. pp. 597–627. Springer (2017)
18. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 360–384. Springer (2018)
19. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)
20. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: international conference on the theory and application of cryptology and information security. pp. 3–33. Springer (2016)
21. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid mpc: Secure multiparty computation with dynamic participants. In: Annual International Cryptology Conference. pp. 94–123. Springer (2021)
22. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled fhe from learning with errors. In: Annual Cryptology Conference. pp. 630–656. Springer (2015)
23. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. pp. 643–662. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
24. Deng, L.: The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE signal processing magazine **29**(6), 141–142 (2012)
25. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2012**, 144 (2012)
26. Froelicher, D., Troncoso-Pastoriza, J.R., Raisaro, J.L., Cuendet, M.A., Sousa, J.S., Cho, H., Berger, B., Fellay, J., Hubaux, J.P.: Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption. Nature communications **12**(1), 5910 (2021)
27. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
28. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference. pp. 75–92. Springer (2013)
29. Halevi, S., Ishai, Y., Jain, A., Komargodski, I., Sahai, A., Yogev, E.: Non-interactive multiparty computation without correlated randomness. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 181–211. Springer (2017)
30. Halevi, S., Polyakov, Y., Shoup, V.: An improved rns variant of the bfv homomorphic encryption scheme. In: Cryptographers' Track at the RSA Conference. pp. 83–105. Springer (2019)
31. Han, K., Hong, S., Cheon, J.H., Park, D.: Efficient logistic regression on large encrypted data. Cryptology ePrint Archive (2018)
32. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: International algorithmic number theory symposium. pp. 267–288. Springer (1998)
33. Kim, T., Kwak, H., Lee, D., Seo, J., Song, Y.: Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition. Cryptology ePrint Archive (2022)
34. López-Alt, A., Tromer, E., Vaikuntanathan, V.: Cloud-assisted multiparty computation from fully homomorphic encryption. Cryptology ePrint Archive (2011)
35. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the forty-fourth annual ACM symposium on Theory of computing. pp. 1219–1234. ACM (2012)
36. Mouchet, C., Bertrand, E., Hubaux, J.P.: An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. Cryptology ePrint Archive (2022)
37. Mouchet, C., Bertrand, E., Hubaux, J.P.: An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. Journal of Cryptology **36**(2), 10 (2023)
38. Mouchet, C., Troncoso-Pastoriza, J., Bossuat, J.P., Hubaux, J.P.: Multiparty homomorphic encryption from ring-learning-with-errors. Proceedings on Privacy Enhancing Technologies **2021**(4), 291–311 (2021)

39. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key fhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 735–763. Springer (2016)
40. Park, J.: Homomorphic encryption for multiple users with less communications. IEEE Access **9**, 135915–135926 (2021)
41. Peikert, C., Shiehian, S.: Multi-key fhe from lwe, revisited. In: Theory of Cryptography Conference. pp. 217–238. Springer (2016)
42. Sav, S., Bossuat, J.P., Troncoso-Pastoriza, J.R., Claassen, M., Hubaux, J.P.: Privacy-preserving federated neural network learning for disease-associated cell classification. Patterns **3**(5), 100487 (2022)
43. Sav, S., Pyrgelis, A., Troncoso-Pastoriza, J.R., Froelicher, D., Bossuat, J.P., Sousa, J.S., Hubaux, J.P.: Poseidon: Privacy-preserving federated neural network learning