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

No Author Given

No Institute Given

Abstract. Homomorphic Encryption (HE), first demonstrated in 2009, is a class of encryption schemes that enables computation over encrypted data. Recent advances in the design of better protocols have led to the development of two different lines of HE schemes – Multi-Party Homomorphic Encryption (MPHE) and Multi-Key Homomorphic Encryption (MKHE). These primitives cater to different applications as each approach has its own pros and cons. At a high level, MPHE schemes tend to be much more efficient but require the set of computing parties to be fixed throughout the entire operation, frequently a limiting assumption. On the other hand, MKHE schemes tend to have poor scaling (quadratic) with the number of parties but allow us to add new parties to the joint computation anytime since they support computation between ciphertexts under different keys.

In this work, we formalize a new variant of HE called Multi-Group Homomorphic Encryption (MGHE). Stated informally, an MGHE scheme provides a seamless integration between MPHE and MKHE, thereby enjoying the best of both worlds. In this framework, a group of parties generates a public key jointly which results in the compactness of ciphertexts and the efficiency of homomorphic operations similar to MPHE. However, unlike MPHE, it also supports computations on encrypted data under different keys similar to MKHE.

We provide the first construction of such an MGHE scheme from BFV and demonstrate experimental results. More importantly, the joint public key generation procedure of our scheme is fully non-interactive so that the set of computing parties does not have to be determined and no information about other parties is needed in advance of individual key generation. At the heart of our construction is a novel re-factoring of the relinearization key.

1 Introduction

With the rapid growth of data science and industry, it has been a significant issue to effectively utilize a large amount of data, yet, made it challenging for individuals or organizations with limited resources to take the benefit. Fortunately, the cloud computing technologies offer sharing of resources and outsourcing computation on the cloud server. Being stored in plaintext, however, the uploaded

data is exposed to the risk of breach by an attacker or by a malicious service provider. In fact, such issues have been occurred continually within the cloud of global enterprises; Alibaba and Facebook in 2019. To securely compute and take the advantage of outsourced data, Homomorphic Encryption (HE) came into the spotlight as a cryptographic solution.

HE enables us to perform operations over the encrypted data without decryption. Thus, it prevents a leakage of private information while evaluating data within untrusted environment. It requires large resource even when it computes a simple arithmetic operation such as multiplication. Therefore, HE is desirable in applying on the cloud system which is able to support huge amount of resource for evaluation. Private Set Intersection is a representative example of this privacy–preserving cloud service [13, 12]. In the PSI, a client only needs to encrypt their private resource and send it to the server. Then, the server finds intersection without revealing any data about the client. However, a typical HE only supports computations between data encrypted by the same key. When there are multiple data owners, therefore, it assumes a trusted third party who possesses a key released to each party for encryption. It may cause authority issue since the key owner can decrypt and infringe on all data providers' privacy using the key.

To resolve the aforementioned problem, there have been several researches to distribute the authority and design HE schemes for multiple parties. The existing works can be mostly classified into two categories, namely Multi-Party Homomorphic Encryption (MPHE, a.k.a. Threshold HE) and Multi-Key Homomorphic Encryption (MKHE). In MPHE [2, 26, 28], multiple parties work collaboratively to generate a joint public key and the joint secret key is (additively) shared among them. The performance of MPHE is comparable to that of the single-key HE schemes since encryption and homomorphic computation are performed in similar fashion. However, the set of participants should be determined 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 multi-round key generation protocol in which the involved parties should interact with each other to generate a joint evaluation (relienarization) key.

Meanwhile, MKHE [25] has very different pros and cons from MPHE. In the multi-key setting, each party independently generates its own key pair which does require no information about other participants. A message can be encrypted using a single key, and it supports homomorphic operation between ciphertexts which do not necessarily have to be encrypted under the same key. The main advantage of MKHE is its flexibility: it is not necessary to pre-determine the list of participants or computational task. From the performance perspective, however, the size of ciphertexts grows and the complexity of homomorphic operations increases depending on the number of the involved parties.

1.1 Our Contribution

Formalization of Multi-Group Homomorphic Encryption Schemes. We propose and formalize a new variant of homomorphic encryption schemes –

Multi-Group Homomorphic Encryption (MGHE). Conceptually, an MGHE scheme is a generalization of both Multi-Party Homomorphic Encryption and Multi-Key Homomorphic Encryption. In other words, MPHE and MKHE variants of HE are simply special instantiations of the MGHE variant. This formalization is motivated by a number of emerging application scenarios, some of which are mentioned in detail in Section 3.3.

Construction of the first MGHE Scheme. We provide the first concrete construction of an MGHE scheme. At the heart of our construction lies a novel refactoring of the key generation algorithm that makes the key generation non-interactive. Thus, our construction (which by definition is an MPHE scheme), is the first MPHE scheme that has non-interactive key generation. At the same time, our construction (which by definition is an MKHE scheme) enjoys the superior performance when comparing against applicable schemes with the same number of parties. Furthermore, it enables application scenarios that are not completely suited for purely MKHE or MPHE schemes but can benefit from a hybrid combination of both.

Concrete Efficiency. We demonstrate that such a generalized HE scheme, one that enjoys the benefits of both MPHE and MKHE, does not come at a high concrete efficiency cost. In fact, when coupled with improvements we make in the "MKHE part" of our scheme, our MGHE scheme is 1.3 to 1.5× faster when used for a homomorphic multiplication. The source code is written in C++ over Microsoft SEAL [29] version 3.3.0.

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. An MPHE scheme behave 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 use 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 relinearization 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 [2, 26], the key generation process involved a multi-round protocol among the parties: the first round for generating a joint public key as described above, and additional rounds for constructing a joint relinearization key from encryptions of $[s]_i \cdot s$

under s generated by parties $i \in I$. In our MPHE scheme, we propose a new key generation algorithm which is *nearly linear* with respect to the secret. This property enables the parties to generate their public keys $[\mathsf{pk}]_i$ independently from individual secrets $[s]_i$ which add up to a valid joint public (encryption and evaluation) key $\mathsf{jpk} = \sum_{i \in I}[\mathsf{pk}]_i$ corresponding to the joint secret $s = \sum_{i \in I}[s]_i$. The technical details of our MPHE construction are described in Section 4.

Finally, to construct our MGHE scheme, we show how the functionality of our MPHE scheme can be extended so that it supports homomorphic computation between ciphertexts under different (joint) secret keys. For example, if we perform homomorphic computation on ct_j's which are MPHE ciphertexts encrypted 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 the output is a "multi-group" ciphertext under the secret $(s_1, \ldots, s_k)$. Moreover, no additional interaction or computation is required among the parties since the same joint public keys of the involved groups can be reused in the relinearization process of multi-group ciphertexts. The technical details of our MGHE construction are described in Section 5.

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 Works

Asharov et al. [2] proposed the notion of MPHE and designed the first MPHE scheme from BGV. Mouchet et al. [26] simplified the idea of [2] and presented an MPHE scheme from BFV. They also improved the functionality to build MPC and demonstrated some experimental results. Recently, Park [28] modified the key generation process to reduce the communication between the parties and suggested a conversion method between MPHE and MKHE. However, prior works had a common limitation that the key generation procedure is a multi-round protocol which requires interaction between all the involved parties.

After López-Alt et al. [25] presented the first MKHE scheme, there have been several studies to design MKHE schemes from GSW [20, 27], LWE (TFHE) [7, 9], BGV [14], and BFV/CKKS [10] for better performance and functionality. Our relinearization algorithm is inspired from that of [10], but we make an additional CRS assumption to possess the linearity property.

Aloufi et al. [1] 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 relinearization key.

In the usual MPHE or MKHE setting, the decryption procedure has N-out-of-N access structure, *i.e.,* all the involved parties should collaborate to decrypt

a ciphertext. Badrinarayanan et al. [3] combined MKHE and a linear secret sharing scheme to construct a threshold MKHE scheme whose decryption works at least t partial decryptions are collected.

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 $< n$ 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_σ a distribution over R which samples n coefficients independently from the discrete Gaussian distribution of variance σ^2.

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, χ, σ), 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* $\mathbf{g} = (g_i) \in \mathbb{Z}^d$ 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 [5, 6], digit decomposition [17], and Residue Number System (RNS) based decompositions [4, 24]. 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 μ under secret s if $\mathbf{u}_0 + s \cdot \mathbf{u}_1 \approx \mu \cdot \mathbf{g} \pmod q$. Chillotti et al. [17] 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 μ.

2.4 Special Modulus

The special modulus technique first proposed in [23] handles the noise growth in key-switching process. It temporarily increases the ciphertext modulus q upto $Q = q \cdot q'$ for some q', and the key switching procedure is executed over R_Q. Then the modulus is reduced back to q so that the key-switching error is cut down by a factor of about q'.

In our implementation, the special modulus technique is applied to the external product of gadget decomposition. The gadget encryptions have ciphertext modulus Q which can be multiplied to the regular ciphertexts of modulus q using the modified external product defined by $c \boxdot \mathbf{v} := \left\lfloor q'^{-1} \cdot \langle \mathbf{g}^{-1}(c), \mathbf{v} \rangle \right\rceil \pmod q$.

3 Formalizing a Multi-Group Homomorphic Encryption Scheme

In this section, we formally describe what is a Multi-Group Homomorphic Encryption (MGHE) scheme, it's correctness and security properties. We also discuss the connection of MGHE schemes with MPHE and MKHE schemes and describe application scenarios that are enabled by MGHE schemes.

3.1 Syntax

Let $\mathcal{M}$ be a plaintext space. An MGHE scheme over $\mathcal{M}$ is a tuple $\mathtt{MGHE} = (\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$ of algorithms and multi-party protocols.

- **Setup:** $\mathsf{pp} \leftarrow \mathtt{MGHE.Setup}(1^\lambda, 1^d)$. On input the security parameter λ and a depth bound d, the setup algorithm outputs a public parameter set pp.
- **Key Generation Protocol:** $\mathtt{MGHE.KeyGen}(\mathsf{pp}, I)$. Initially the parties in I hold pp and run the key-generation protocol. At the end of the protocol, it outputs a public key pk and each party $i \in I$ obtains a private share $[\mathsf{sk}]_i$. We denote by sk the (implicitly defined) secret key which can be recovered from the shares $[\mathsf{sk}]_i, i \in I$.
- **Encryption:** $\mathsf{ct} \leftarrow \mathtt{MGHE.Enc}(\mathsf{pk}; m)$. Given a public key pk and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
  For convenience, we assume that every ciphertext implicitly includes the references to the associated public keys. For example, a fresh ciphertext contains one reference to the public key that is used in its encryption.
- **Evaluation:** $\mathsf{ct} \leftarrow \mathtt{MGHE.Eval}(\mathsf{pk}_1, \ldots, \mathsf{pk}_k; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$. Given input a circuit $C : \mathcal{M}^\ell \to \mathcal{M}$, ℓ ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$, let $\mathsf{pk}_1, \ldots, \mathsf{pk}_k$ be the public keys which are associated to at least one input ciphertext. The evaluation algorithm outputs a ciphertext ct. The output ciphertext contains ℓ references to the associated public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell$.
- **Decryption:** $m \leftarrow \mathtt{MGHE.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell; \mathsf{ct})$. Given a ciphertext ct and the associated secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell$, the decryption algorithm outputs a message $m \in \mathcal{M}$.

Note that while the decryption algorithm requires access to the individual secret keys, when implemented in practice, an additional algorithm known as distributed decryption is used to preserve privacy of individual secret keys. For our MGHE construction described in Section 5, we also provide such a distributed decryption algorithm. Our algorithm uses a standard MPC based interactive protocol to achieve secure decryption over an MGHE scheme setup.

Security. We require that an MGHE scheme is semantically secure against a dishonest majority. In other words, for any adversarial subset of parties $\mathcal{A} \subsetneq I$ and for any messages $m_1, m_2 \in \mathcal{M}$, the advantage of the adversary in distinguishing the distributions $\mathtt{MGHE.Enc}(\mathsf{pk}; m_1)$ and $\mathtt{MGHE.Enc}(\mathsf{pk}; m_2)$ is negligible in the security parameter λ.

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

$$\mathtt{MGHE.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_k; \mathtt{MGHE.Eval}(\mathsf{pk}_1, \ldots, \mathsf{pk}_k; C, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) = C(m_1, \ldots, m_\ell)$$

with an overwhelming probability in λ.

3.2 Connections to MPHE and MKHE schemes

MPHE and MKHE are generalized notions of HE with distributed authority. Since they have different (dis)advantages, one is not compatible with the other. Our suggestion, the MGHE primitive, can be considered as a generalization of MPHE and MKHE. In other words, both MPHE and MKHE are specific instantiations of MGHE which are obtained by restricting the number of groups or parties in the MGHE setting. Recall that an MGHE scheme works on groups of parties, where the evaluation is done within a group in MPHE sense and among groups in MKHE sense. 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.

From technical aspect, our MGHE construction is not just a combination of the existing MPHE and MKHE schemes but we made a significant improvement in the MPHE part. The existing MPHE schemes had a common limitation that the key generation procedure requires multiple rounds of interaction between the key owners due to the quadratic structure of the relinearization key. In our scheme, however, key owners only need to publish their own key pairs independently from the groups they belong to. This makes our MGHE scheme achieve the non-interactivity property of the setup phase and especially come in handy when a party belongs to multiple groups since the same key can be reused to build joint public keys of belonging groups and eventually saves the communication cost proportional to the number of groups. In addition, this property allows

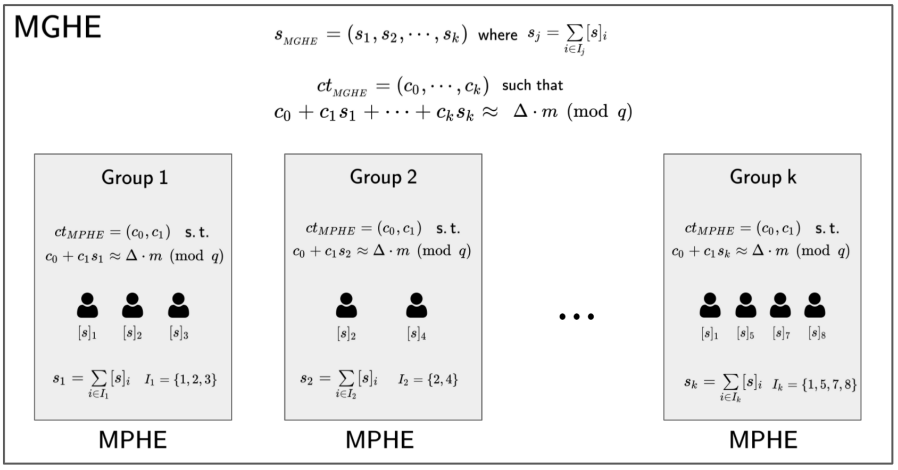

Fig. 1. A schematic presenting the overall structure of MGHE schemes. Each boxed group of participants acts as an MPHE scheme. The secret keys and ciphertext equations for each group and the entire set of participants (including between groups) are described above.

us to form the computing groups dynamically on-the-fly by data providers or the cloud without interacting with key owners.

3.3 Application

Our MGHE scheme has three important properties that make it a key enabler for certain application scenarios. In particular, an MGHE scheme is (1) an HE scheme (2) enjoy non-interactivity of the setup (key generation) phase and (3) provides efficient computation between ciphertexts encrypted under different keys. Below we describe a few applications that can be enabled by such a scheme.

Vertical Federated Learning (VFL): Federated learning is a technique for decentralized machine learning where the data is available in silos with privacy constraints. Vertical federated learning is a sub-category of this field where the various datasets differ in the feature space. For instance, two datasets might share the same set of user (rows) but store different attributes of these users (columns). VFL can further be split into two components, the feature engineering and the training/learning. Examples of the former include private set intersections protocols to securely identify common column identifiers while examples of the latter include traditional training of ML models using FL (once a common feature space is identified). Feature engineering is an important open problem in the field of VFL and designing secure techniques for the same is an important unsolved challenge.

The only promising solutions, with cryptographic privacy, that exist today rely on HE or MPC. However, each of these approaches have severe practical

limitations when considering the above scenario. While traditional HE schemes such as MKHE suffer from huge overheads when datasets need to be adaptively selected, MPC schemes incur the overhead of resharing the features with every group of parties. MGHE schemes can provide a transformative new tool to share features one-time, and adaptively process them making it an ideal technical tool for solving the feature engineering challenge.

Privacy-preserving Datalake Infrastructure: Privacy regulations such as GDPR envision a goal that enables users to take control of their data in our increasing digital infrastructure. While such a goal has not been realized, our proposed MGHE scheme can provide a practical solution to the vision.

Let us assume that normal users (individuals) would like to share their data for privacy conscious analytics. However, given the overhead of hosting such databases, it is not feasible to expect each such individual to host their data. One promising approach is to have data hosting services launched by a small number of entities (such as Google, Alibaba, Facebook etc.). Thus, each user encrypt and share their data to one of the datasets held by one such entity. Thus, we have datasets $D_{\mathsf{Google}}, \cdots, D_{\mathsf{Facebook}}$, which are encrypted under the secret keys of different entities and operated by a dynamic and flexible MKHE scheme. In addition, these entities may wish to distribute the authority to eliminates the risks of a single point of failure. Hence multiple key centers can share the secret key using an MPHE scheme since it is acceptable to use a fixed key access structure in a single entity.

As a result, the use of MGHE here (1) allows for adaptive data selection where an analyst can selectively use or ignore entire datasets as well as parts of datasets (2) allows users to inherently gain control of their data, and (3) enables such a system with low overhead. In this way, MGHE schemes combine the best of both MPHE and MKHE schemes and make for an ideal solution to such a privacy-preserving datalake infrastructure.

4 Construction of MPHE with Non-Interactive Setup

In this section, we present our MPHE scheme which will be extended to an MGHE scheme in the next section. We describe our scheme over the BFV scheme [5, 21].

4.1 Basic Scheme

Our scheme is based on the Common Random String (CRS) model, *i.e.,* all the involved parties have access to the same random vectors sampled in the setup phase. 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 p, the ciphertext modulus q, the key distribution χ over R, and the error parameter σ. Sample random vectors $\mathbf{a}, \mathbf{u} \leftarrow U(R_q^d)$. Return the parameter set $\mathsf{pp} = (n, p, q, \chi, \sigma, \mathbf{a}, \mathbf{u})$. We write $\Delta = \lfloor q/p \rceil$.

The joint key generation procedure consists of two steps; each party first generates (locally) a key pair and publishes the public key, and then the joint public key of a group of parties is built from the collection of public keys from the associated parties. The most distinguishing feature of our scheme is that it is fully non-interactive. The generation of the individual keys can be done locally without knowing any information about other parties, and building a joint public key can be done publicly with no interaction with the involved parties.

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

• $\underline{\texttt{MPHE.JointKeyGen}(\{[\mathsf{pk}]_i : i \in I\})}$: 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 $\mathsf{jpk} = (\mathbf{b}, \mathbf{d}, \mathbf{v}) \in R_q^d \times R_q^d \times R_q^d$ where $\mathbf{b} = \sum_{i \in I} [\mathbf{b}]_i$, $\mathbf{d} = \sum_{i \in I} [\mathbf{d}]_i$ and $\mathbf{v} = \sum_{i \in I} [\mathbf{v}]_i$. We denote the joint encryption key as $\mathsf{jek} = (\mathbf{b}[0], \mathbf{a}[0]) \in R_q^2$.

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. Note that the public key $[\mathsf{pk}]_i$ is *nearly linear* with respect to $[s]_i$ and $[r]_i$ so that the joint public key $\mathsf{jpk} = (\mathbf{b}, \mathbf{d}, \mathbf{v})$ satisfies the same properties as the individual keys:

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

Note that the encryption key 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 $m \in R_p$, sample $t \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Return the ciphertext $\mathsf{ct} = t \cdot \mathsf{jek} + (\Delta \cdot m + e_0, e_1) \pmod q$.

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

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

Correctness and security. 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.

We claim that our MPHE scheme is semantically secure against a dishonest majority under the RLWE assumption with parameter (n, q, χ, σ). Suppose that the adversary $\mathcal{A} = I \backslash \{i\}$ is the set of all parties except i and let

$[\mathsf{pk}]_i = ([\mathbf{b}]_i, [\mathbf{d}]_i, [\mathbf{v}]_i)$ be a public key generated by the i-th party. Then the rows of $([\mathbf{b}]_i, \mathbf{a})$ are RLWE samples of secret $[s]_i$, and the encryption key $\mathbf{b}[0] = [\mathbf{b}]_i[0] + \sum_{j \in I \setminus \{i\}} [\mathbf{b}]_j[0]$ is computationally indistinguishable from a uniform random variable over R_q to $\mathcal{A}$. Since our encryption algorithm is the usual BFV encryption, our scheme also achieves the semantic security.

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 that our scheme still remains secure even if $[\mathbf{d}]_i$ and $[\mathbf{v}]_i$ are public. We also remark that our key generation algorithm is identical to that of [10] except that all parties share the random vector $\mathbf{u}$ instead sampling it independently.

Distributed decryption protocol. 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 flooding technique [2, 26]. 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.

• $\underline{\mathtt{MPHE.DistDec}(\{[\mathsf{sk}]_i : i \in I\}, \sigma'; \mathsf{ct})}$: 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 $m = \lfloor (p/q) \cdot \mu \rceil$.

4.2 Arithmetic Operations

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

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

• $\underline{\mathtt{MPHE.Mult}(\mathsf{jpk}; \mathsf{ct}, \mathsf{ct}')}$: Given two ciphertexts $\mathsf{ct} = (c_0, c_1)$, $\mathsf{ct}' = (c_0', c_1')$ and a joint public key jpk, let $\mathsf{ct}_{\mathsf{mul}} = (c_0'', c_1'', c_2'')$ where $c_0'' = \lfloor (p/q) \cdot (c_0 c_0') \rceil \pmod q$, $c_1'' = \lfloor (p/q) \cdot (c_0 c_1' + c_1 c_0') \rceil \pmod q$ and $c_2'' = \lfloor (p/q) \cdot (c_1 c_1') \rceil \pmod q$. Return the ciphertext $\mathsf{ct}_{\mathsf{relin}} \leftarrow \mathtt{MPHE.Relin}(\mathsf{jpk}; \mathsf{ct}_{\mathsf{mul}})$ where $\mathtt{MPHE.Relin}(\cdot;\cdot)$ is the relinearization procedure described in Alg. 1.

Correctness of homomorphic multiplication. Let $[s]_i$, $[r]_i$ be the polynomials sampled from χ during the generation of a key pair $[\mathsf{sk}]_i = [s]_i$ and

Algorithm 1 Relinearization procedure of MPHE

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

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

$[\mathsf{pk}]_i = ([\mathbf{b}]_i, [\mathbf{d}]_i, [\mathbf{v}]_i)$ 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 ct and 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 $(c_0^*, c_1^*) \leftarrow \mathtt{MPHE.Relin}(\mathsf{jpk}; (c_0'', c_1'', c_2''))$ 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.

4.3 Homomorphic Automorphism

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 [30, 22]. 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. In the single-key setting, the homomorphic evaluation of an automorphism can be done by taking the automorphism on input ciphertext and then performing the key-switching procedure.

We present a multi-party variant of homomorphic automorphism such that the joint automorphism key is generated non-interactively. The following setup and automorphism key generation procedures can be added to the basic scheme to support homomorphic evaluation of an automorphism ψ. We note that it is required to sample an additional CRS and include it in the public parameter.

• $\mathtt{MPHE.Setup}(1^\lambda)$: Sample a random vector $\mathbf{k} \leftarrow U(R_q^d)$ and put it into the parameter set pp.

• $\mathtt{MPHE.AutKeyGen}([s]_i)$: Let $[s]_i$ be the secret key of party i. Sample $[\mathbf{e}]_i \leftarrow D_\sigma^d$ and compute $[\mathbf{h}]_i = -[s]_i \cdot \mathbf{k} + \psi([s]_i) \cdot \mathbf{g} + [\mathbf{e}]_i \pmod q$. Return the automorphism key $[\mathsf{ak}]_i = [\mathbf{h}]_i \in R_q^d$.

• $\underline{\texttt{MPHE.JointAutKeyGen}(\{[\mathsf{ak}]_i : i \in I\})}$: Given a set of automorphism keys of parties $i \in I$, return the joint automorphism key $\mathsf{jak} = \mathbf{h} \in R_q^d$ where $\mathbf{h} = \sum_{i \in I} [\mathbf{h}]_i \pmod q$.

The automorphism key generation algorithm is also nearly linear with respect to the secret $[s]_i$. Hence the joint automorphism key, together with CRS $\mathbf{k}$, forms a gadget encryption of $\psi(s)$ under the secret s which can be used as a key-switching key from $\psi(s)$ to s for the usual automorphism evaluation algorithm as follows.

• $\underline{\texttt{MPHE.EvalAuto}(\mathsf{jak}; \mathsf{ct})}$: 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 and security of homomorphic automorphism. Let $\mathsf{ct}_{\mathsf{aut}} = (c_0', c_1') \leftarrow \texttt{EvalAuto}(\mathsf{jak}; \mathsf{ct})$ 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

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

The security of homomorphic automorphism relies on the same RLWE assumption as the basic scheme. It also requires additional security assumption similar to the standard BFV scheme.

5 Extension to MGHE

In this section, we design an MGHE scheme by improving the functionality of our MPHE scheme. Recall that, in the MPHE setting, we can perform computations on encrypted data only if input ciphertexts are encrypted under the same joint key, but MGHE supports homomorphic operations between multi-party ciphertexts which do not necessarily have to be encrypted under the same key.

5.1 Scheme description

Let us give a technical overview of our multi-group variant of the BFV scheme. As we shall see, the setup, (joint) key generation, and encryption procedures happen to 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.

• $\underline{\texttt{MGHE.Setup}(1^\lambda)}$: Run $\texttt{MPHE.Setup}(1^\lambda)$ and return the public parameter $\mathsf{pp} = (n, p, q, \chi, \sigma, \mathbf{a}, \mathbf{u})$. If necessary, sample additional $\mathbf{k} \leftarrow U(R_q^d)$ for homomorphic automorphism and incorporate it into the parameter set.

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

• $\underline{\texttt{MGHE.AutKeyGen}([s]_i)}$: The automorphism key can be generated if a CRS $\mathbf{k}$ is included in pp. Given the secret key $[s]_i$ of party i, run $\texttt{MPHE.AutKeyGen}([s]_i)$ and output the automorphism key $[\mathbf{h}]_i = [\mathsf{ak}]_i$.

Note that both $\texttt{KeyGen}(i)$ and $\texttt{AutKeyGen}([s]_i)$ are *non-interactive* and thus do not require any knowledge of groups or other parties involved.

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

• $\underline{\texttt{MGHE.JointAutKeyGen}(\{[\mathsf{ak}]_i : i \in I\})}$: Given a set of automorphism keys of $i \in I$, output the joint automorphism key $\mathsf{jak} = \mathbf{h} \leftarrow \texttt{MPHE.JointAutKeyGen}(\{[\mathsf{ak}]_i : i \in I\})$.

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

As we discussed in Section 3, 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 $\mathcal{I} = \{I_1, \ldots, I_k\}$ 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 the j-th group I_j for $1 \le j \le k$.

Homomorphic operations include a pre-processing step which aligns the components of input ciphertexts as follows. Suppose that we are given two multi-group ciphertexts such that they are associated to the ordered sets of groups $\mathcal{I}$ and $\mathcal{I}'$, respectively. We write their union as $\mathcal{I} \cup \mathcal{I}' = \{I_1, \ldots, I_k\}$ by permuting the elements of $\mathcal{I}$ and $\mathcal{I}'$ properly if needed. 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 $\mathcal{I} \cup \mathcal{I}'$ even if it is not explicitly mentioned in the algorithm description.

In [10], the authors introduced an MKHE scheme from BFV with a relinearization method. Our MGHE scheme and [10] have similar ciphertext and decryption structures $c_0 + c_1 \cdot s_1 + \cdots + c_k \cdot s_k \approx \Delta \cdot m$ although each secret s_i is generated by a single party in [10]. Our relinearization algorithm is also similar to the one in [10], but our scheme enjoys better performance by reducing the number of external products by almost a factor of 2. More concretely, the former method computes $c''_{i,j} \leftarrow c_{i,j} \boxdot \mathbf{b}_j$ and updates (c'_0, c'_i) by $(c'_0, c'_i) + c''_{i,j} \boxdot (\mathbf{v}_j, \mathbf{u})$ running over $1 \le i, j \le k$, inducing $4k^2$ external products in total. We observe that $\sum_{1 \le j \le k} c_{i,j} \boxdot \mathbf{b}_j$ can be precomputed before the update and reused for the

Algorithm 2 Relinearization procedure of MGHE

Input: $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$, $\mathsf{jpk}_j = (\mathbf{b}_j, \mathbf{d}_j, \mathbf{v}_j)$ 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

relinearization of multiple ciphertext components which consequently reduces the number of external products down to $2k^2 + 2k$.

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

• $\underline{\mathtt{MGHE.Mult}(\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two multi-group ciphertexts $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$, $\overline{\mathsf{ct}}' = (c_0', \ldots, c_k')$ and k joint public keys $\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k$, compute $\overline{\mathsf{ct}}_{\mathsf{mul}} = (c_{i,j})_{0 \le i,j \le k}$ where $c_{i,j} = \lfloor (p/q) \cdot c_i c_j' \rceil \pmod q$ for $0 \le i, j \le k$. Return the ciphertext $\overline{\mathsf{ct}}_{\mathsf{relin}} \leftarrow \mathtt{MGHE.Relin}(\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k; \overline{\mathsf{ct}}_{\mathsf{mul}})$ where $\mathtt{MGHE.Relin}(\cdot)$ is the relinearization procedure described in Alg. 2.

The idea of homomorphic automorphism for MPHE can be also extended to the multi-group case. Given a multi-group ciphertext $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$ linked to k groups $I_1, \ldots, I_k$, 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 $\psi \in \mathcal{G}\mathsf{al}(K/\mathbb{Q})$.

• $\underline{\mathtt{MGHE.EvalAuto}(\mathsf{jak}_1, \ldots, \mathsf{jak}_k; \overline{\mathsf{ct}})}$: Given a ciphertext $\overline{\mathsf{ct}} = (c_0, c_1, \ldots, c_k)$ and the joint automorphism keys $\mathsf{jak}_j = \mathbf{h}_j$ for $1 \le j \le k$, compute and return the ciphertext $\overline{\mathsf{ct}}_{\mathsf{aut}} = (c_0', c_1', \ldots, c_k')$ 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 $1 \le j \le k$.

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.

• $\underline{\mathtt{MGHE.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_k; \overline{\mathsf{ct}})}$: 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 $m = \left\lfloor (p/q) \cdot (c_0 + \sum_{1 \le j \le k} c_j \cdot s_j) \right\rceil \pmod p$.

• $\underline{\mathtt{MGHE.DistDec}(\cup_{1 \le j \le k} I_j, \sigma'; \overline{\mathsf{ct}})}$: Let $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$ be a multi-group ciphertext corresponding to the set of groups $\mathcal{I} = \{I_1, \ldots, I_k\}$ and $[\mathsf{sk}]_i = [s]_i$ be the secret key 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 $m = \left\lfloor (p/q) \cdot (c_0 + \sum_{1 \le j \le k} \sum_{i \in I_j} [\mu_j]_i) \right\rceil \pmod p$.

Security. As described above, our MGHE scheme is based on the same (joint) key generation and encryption algorithms as the previous MPHE scheme. Therefore, it achieves the semantic security against a dishonest majority under the same RLWE assumption of parameter (n, q, χ, σ) and additional circular security assumptions.

Correctness. The correctness of the encryption algorithm is obvious so we focus on the homomorphic multiplication (relinearization) and automorphism algorithms of our MGHE scheme.

Suppose that $\overline{\mathsf{ct}}$ and $\overline{\mathsf{ct}}'$ are encryptions of m and m' under secret $\overline{\mathsf{sk}} = (s_1, \ldots, s_k)$, respectively, and let $\overline{\mathsf{ct}}_{\mathsf{mul}} = (c_{i,j})_{0 \le i,j \le k} = \left\lfloor (p/q) \cdot \overline{\mathsf{ct}} \otimes \overline{\mathsf{ct}}' \right\rceil \pmod q$. Then, we have $\langle \overline{\mathsf{ct}}_{\mathsf{mul}}, (1, \overline{\mathsf{sk}}) \otimes (1, \overline{\mathsf{sk}}) \rangle \approx \Delta \cdot mm' \pmod q$. We claim that if $\overline{\mathsf{ct}}_{\mathsf{relin}} \leftarrow \mathtt{MGHE.Relin}(\{\mathsf{jpk}_j\}_{1 \le j \le k}; \overline{\mathsf{ct}}_{\mathsf{mul}})$, then the output ciphertext $\overline{\mathsf{ct}}_{\mathsf{relin}} = (c^*_0, \ldots, c^*_k)$ 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.

Finally, we show below the correctness of multi-group homomorphic automorphism algorithm:

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

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

5.2 Other issues

Applying to other HE schemes. 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 [6] and CKKS [16]. In particular, we implement MGHE schemes from both BFV and CKKS and present experimental results in the next section. We provide a formal description of multi-group CKKS in Appendix A.

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. [11, 15, 8]) share the same workflow consisting of arithmetic operations and linear transformations which can be also represented by basic operations and homomorphic automorphisms. As these basic operations (multiplication, rotation) are supported in our MGHE schemes, the bootstrapping procedure can be also performed in a similar manner.

5.3 Building MPC from MGHE

MGHE being a generalization of both the MKHE and MPHE primitives, it can be used as a drop-in replacement for any application build using these primitives. Thus, MGHE can be used for general 2-round MPC computation, for outsourced computation applications and in distributed machine learning set-ups. Similarly, it can be used a building block in MPC protocols that require a varying number of parties [18, 19].

Both MPHE and MKHE can be used as building blocks for MPC [25, 26, 27], however, each primitive has its own limitations that constrain the set of applications where these techniques are useful. For example, to apply MPHE scheme in MPC protocol, all parties have to communicate with each other to

generate evaluation keys. On the other hand, MKHE schemes are more expensive than MPHEs in terms of time and space complexity because ciphertexts expand when 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. Below we describe the high level flow of a 2-round MPC computation.

- **Setup:** The setup phase is driven by the application however, in abstract terms, the number of parties have two hierarchies: there is a group of parties that have a distributed secret key (the MPHE component) and there exist a number of such groups with their own independent set-up (interaction between groups is the MKHE component). As the first step of the protocol, all these parties in every group together instantiate the MGHE scheme. They determine cryptographic public parameter set (e.g. RLWE dimension, plaintext modulus). Based on the parameter set, they generate their own key pair, consist of the public key and the private share, and broadcast it. We can treat this step as an offline phase since all of these procedures has to be run only once and each party is able to run this step independently
- **Circuit evaluation:** In the next step, inputs are encrypted by joint encryption key which is generated by the summation of published key pairs in each group. After encryption, the ciphertexts are provided to a computing party which may be an external entity such as a cloud service provider. In general, semi-honest cloud service provider or parties themselves in MPC assume the role of computing party. The circuit is evaluated using the homomorphic properties of the encryption scheme and thus does not require any interaction.
- **Distributed decryption:** To securely decrypt a result, *i.e.,* without revealing the secret keys of each party, we can use an interactive protocol known as distributed decryption. Each party partially decrypts the ciphertext using their own secret keys. As mentioned in distributed decryption of MGHE scheme, each party has to add an additional error to prevent data leakage from partial decryption. Such data leakage prevention uses techniques such as noise smudging [2] where an additional error super-poly size compared to existing error is sampled and added to mask each individual partial decryption. Finally, the decryption is performed by adding all of the partially decrypted results.

6 Experimental Results

We implemented our MGHE scheme and measured the elapsed time of basic operations. The source code is written in C++ over Microsoft SEAL [29] version 3.3.0 with implementations of both BFV and CKKS. We introduce some optimization techniques and report the timing results of basic operations under several parameter settings. The experiment was conducted on Intel(R) Core(TM) i9-10900 CPU @ 2.80GHz and 64GB RAM. In our implementation, the key distribution χ samples the coefficients uniformly from the ternary set $\{-1, 0, 1\}$,

the error parameter is fixed as $\sigma = 3.2$, and the plaintext modulus is fixed as $p = 65537$. Our experimentation is performed on three parameter sets: the ring dimension $n = 2^{13}$, 2^{14}, or 2^{15} and the ciphertext modulus $\lceil \log q \rceil = 218$, 438, or 862, respectively, which achieve at least 128-bit security level. In addition, we set q as a product of 4, 8, or 16 distinct primes of bitsize ≤ 60.

6.1 Execution time for basic operations

Table 1 shows the execution times to operate multiplication with relinearization and rotation, both in the BFV and CKKS schemes. Although not included in the table, the number of parties in groups does not affect the execution time in both schemes. It is because, as described in Section 4.1, 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 base ring and the number of groups participating in evaluation. As dimension of the plaintext increases, the ciphertext modulus increases and eventually affects the execution time of arithmetic operations. Moreover, according to the Section 5.1, relinearization or automorphism requires more external products when there are more groups involved in the evaluation. Therefore, it takes more time to complete multiplication or rotation.

n	k	Mult + Relin				Auto			
		BFV		CKKS		BFV		CKKS	
		Ours	[10]	Ours	[10]	Ours	[10]	Ours	[10]
2^{13}	1	17	20	6	8	3	3	4	4
	2	32	44	14	22	6	7	7	8
	4	77	116	37	67	11	14	13	16
	8	213	365	110	229	22	28	27	31
2^{14}	1	100	110	51	59	21	22	25	24
	2	206	257	122	165	42	47	47	49
	4	514	717	331	521	81	88	92	95
	8	1,490	2,350	1,018	1,845	160	176	178	193
2^{15}	1	651	675	427	465	161	170	170	172
	2	1,443	1,715	1,035	1,364	317	333	333	359
	4	3,731	5,025	2,874	4,287	631	646	671	711
	8	11,425	17,450	9,040	15,159	1,303	1,332	1,338	1,413

Table 1. Performance of our MGHE scheme and the MKHE scheme by Chen et al. [10]: execution times to operate homomorphic multiplication (Mult + Relin) and automorphism (Auto), taken in milliseconds (ms). n denotes the dimension of base ring and k denotes the number of the associated groups (keys) to the ciphertext.

We also show the performance of the MKHE scheme [10] for comparison. As described in the Section 5.1, we reduce the number of external products during the relinearization. According to the Table 1, our algorithm achieves better performance in homomorphic multiplication when more than one groups are involved in the computation. It reduces the complexity of homomorphic multiplication of BFV and CKKS by about 1.3 and 1.5 times, respectively, when the input ciphertext is associated to eight groups.

6.2 Noise growth

We also evaluate how much noise increases when proceeding multiplication with relinearization. Table 2 shows the growth of ciphertext noise from homomorphic multiplication. In this experimentation, we generate k fresh ciphertexts using different joint public keys and compute their summation to obtain an k-group ciphertext (depth 0). Then, we perform the squaring operation repeatedly and obtain ciphertexts of depth from 1 to 5.

We observe that the ciphertext noise grows more quickly when as the number of the involved parties increases, which aligns with our theoretic estimation. We refer the reader Appendix B for more detailed noise analysis of the relinearization and multiplication algorithms. In addition, the automorphism algorithm introduces a noise in an additive manner (unlike multiplicative noise growth of homomorphic multiplication), so it has almost no effect on the bitsize of noise since the additional automorphism noise is insignificant compared to the total noise.

6.3 Application to Oblivious Neural Network

One possible application of MGHE is to enable secure workflow of machine learning models comprising the privacy of multiple data owners. Suppose the model is trained with datasets owned by multiple providers. If 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 client may not be determined aforehand and the model may deal with multiple independent clients. Thus, it is more reasonable to perform inference using an MKHE scheme which shows better flexibility. The model and the user data encrypted under the different keys of the model owners and the client, respectively, are converted into encrypted multi-key ciphertext under those two keys. After evaluating neural network inference in 2-key MKHE, the output is obtained by distributed decryption among the model owners and the client. Our MGHE interpolates between MPHE (where the neural network is trained) and MKHE (where the neural network inference is performed) so that the entire process can be done in the same encryption scheme. Aloufi et al. [1] presented a similar scenario in the random forest model where each party owns a decision tree, not a single data. This is apparently involved in our scenario as their set-up is a special case of our set-up where we instantiate an MGHE scheme with multiple parties and inference is performed by 2-key evaluation, *i.e.,* 2-group evaluation.

k	N	Depth 0	Depth 1	Depth 2	Depth 3	Depth 4	Depth 5	Avg. Diff.
1	1	15.0	45.3	79.3	114.7	150.1	185.4	34.1
	2	15.0	46.0	80.6	116.9	152.7	188.7	34.7
	4	15.0	47.1	83.0	119.6	156.0	192.7	35.5
	8	15.0	48.1	85.0	122.7	160.3	197.8	36.6
2	1	15.0	47.0	81.8	118.3	154.7	191.0	35.2
	2	15.0	47.4	82.8	119.4	156.1	192.9	35.6
	4	15.0	48.3	85.0	122.6	160.2	197.8	36.6
	8	15.0	49.4	86.6	124.4	162.4	200.4	37.1
4	1	15.0	48.3	83.3	119.9	156.0	192.6	35.5
	2	15.0	49.1	84.4	121.4	158.3	195.3	36.1
	4	15.0	49.9	86.8	125.0	162.0	199.7	36.9
	8	15.0	50.2	87.1	125.2	163.3	201.3	37.3
8	1	15.0	49.7	84.6	121.3	158.3	195.0	36.0
	2	15.0	50.3	86.7	124.4	162.1	200.3	37.1
	4	15.0	51.3	87.8	126.1	164.6	203.3	37.7
	8	15.0	51.9	88.9	127.4	166.1	204.1	37.8

Table 2. Size of noise (bits) to operate multiplication with relinearization (Mult + Relin) several times according to the number of groups (k) and the number of parties (N). Initial level is 9 and evaluation is examined on the BFV scheme and $n = 2^{14}$.

In this section, we apply our MGHE scheme to a convolution neural network (CNN) model and compared the performance with the previous MKHE paper [10]. Our experimentation is based on the same model with the following structure: the convolutional layer takes 28×28 input image and perform convolution with 4×4 window and stride $(2, 2)$. The five output channels are followed by the square activation function on 845 inputs. Then the fully connected layer with 845 inputs and 64 outputs is again followed by the square activation function. Finally, the second fully connected layer outputs 10 values, then the softmax is applied to obtain probabilistic values.

Table 3 shows the performance of MKHE and MGHE schemes for evaluating inference over encrypted CNN model. In MKHE scheme, the model is evaluated under two different keys each from the model owner and the client. On the other hand, the good property of our MGHE scheme supporting multiple model owners replace the key of a single model owner by the joint public key of parties that owns the model. The number of model owners, again, does not influence the performance and is set to one in Table 3 for comparison. It is shown that our MGHE scheme is slightly faster in all layers thanks to our optimization technique in the relinearization process.

Scheme	Convolution	Square-1	FC-1	Square-2	FC-2
MKHE [10]	600	121	621	62	112
MGHE	579	107	603	55	108

Table 3. Time taken in milliseconds (ms) to perform oblivious CNN inference to the MNIST dataset. The parameter set of dimension $n = 2^{14}$ is used.

7 Conclusion

In this work, we propose Multi-Group Homomorphic Encryption, a new variant of homomorphic encryption for multiple parties. MGHE generalizes the notion of Multi-Key Homomorphic Encryption and Multi-Party Homomorphic Encryption. We also provide the first construction of an MGHE scheme that enjoys non-interactive key generation. We propose some optimizations to implement this scheme and our benchmarks indicate it is between $1.3 - 1.5\times$ faster than existing HE schemes for the same applications.

References

1. 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)
2. 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)
3. 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)
4. 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)
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
7. Brakerski, Z., Perlman, R.: Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: Annual Cryptology Conference. pp. 190–213. Springer (2016)
8. 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)
9. 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)

10. 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)
11. 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)
12. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled psi from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1223–1237 (2018)
13. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1243–1255 (2017)
14. 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)
15. 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)
16. 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)
17. 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)
18. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid mpc: Secure multiparty computation with dynamic participants. Cryptology ePrint Archive, Report 2020/754 (2020), `https://ia.cr/2020/754`
19. 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)
20. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled fhe from learning with errors. In: Annual Cryptology Conference. pp. 630–656. Springer (2015)
21. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2012**, 144 (2012)
22. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 465–482. Springer (2012)
23. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
24. 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)
25. 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)
26. Mouchet, C., Troncoso-Pastoriza, J., Bossuat, J.P., Hubaux, J.P.: Multiparty homomorphic encryption from ring-learning-with-errors. Cryptology ePrint Archive, Report 2020/304 (2020), `https://ia.cr/2020/304`

27. 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)
28. Park, J.: Homomorphic encryption for multiple users with less communications. Cryptology ePrint Archive, Report 2021/1085 (2021), https://ia.cr/2021/1085
29. Microsoft SEAL (release 3.3). https://github.com/Microsoft/SEAL (Jun 2019), microsoft Research, Redmond, WA.
30. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, codes and cryptography **71**(1), 57–81 (2014)

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

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

• $\underline{\texttt{MP-CKKS.Add}(\mathsf{ct}, \mathsf{ct}')}$: If ct and 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.

• $\underline{\texttt{MP-CKKS.Mult}(\mathsf{jpk}; \mathsf{ct}, \mathsf{ct}')}$: If ct and 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 public key jpk, 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 $\texttt{MP-CKKS.Relin}(\mathsf{jpk}; \mathsf{ct}_{\mathsf{mul}})$ where $\texttt{MP-CKKS.Relin}(\cdot)$ is the relinearization procedure described in Alg. 1.

• $\underline{\texttt{MP-CKKS.Rescale}(\mathsf{ct})}$: 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 $l - 1$.

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

• $\underline{\texttt{MP-CKKS.DistDec}(\{[\mathsf{sk}]_i : i \in I\}, \sigma'; \mathsf{ct})}$: 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 $m = (c_0 + \sum_{i \in I} [\mu]_i)$.

A.2 Extension to MGHE with CKKS

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

• $\underline{\texttt{MG-CKKS.Add}(\overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: 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.

• $\underline{\texttt{MG-CKKS.Mult}(\{\mathsf{jpk}_j\}_{1 \le j \le k}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Set ct and 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 $\{\mathsf{jpk}_j\}_{1 \le j \le k}$ the collection of the joint public 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 $\texttt{MG-CKKS.Relin}(\{\mathsf{jpk}_j\}_{1 \le j \le k}; \overline{\mathsf{ct}}_{\mathsf{mul}})$ where $\texttt{MG-CKKS.Relin}(\cdot)$ is the relinearization procedure described in Alg. 2.

• $\underline{\texttt{MP-CKKS.Rescale}(\overline{\mathsf{ct}})}$: 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 $l - 1$.

• $\underline{\texttt{MG-CKKS.Dec}(\{\mathsf{sk}_j\}_{1 \le j \le k}; \overline{\mathsf{ct}})}$: 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 $m = \langle \mathsf{ct}, \mathsf{sk} \rangle = (c_0 + \sum_{1 \le j \le k} c_i \cdot s_j) \pmod p$.

• $\underline{\texttt{MG-CKKS.DistDec}(\{[\mathsf{sk}_j]_i\}_{1 \le j \le k, i \in I_j}, \sigma'; \overline{\mathsf{ct}})}$: Let $\overline{\mathsf{ct}} = (c_0, \ldots, c_k)$ be a multi-group ciphertext corresponding to the set of groups $\mathcal{I} = \{I_1, \ldots, I_k\}$ 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 $m = (c_0 + \sum_{1 \le j \le k} \sum_{i \in I_j} [\mu_j]_i) \pmod p$.

B Noise analysis

In BFV (or CKKS) scheme, the correct decryption is guaranteed when the size of error term, or the noise, in the ciphertext is smaller than a certain fraction of ciphertext modulus q. A fresh ciphertext has a small initial noise, yet it grows along with homomorphic operation, especially multiplication (with relinearization). We analyze an average-case noise growth on the variance of polynomial coefficients.

Before estimating a noise growth, we specify some distributions for sampling randomness or errors. Let the key distribution χ as the uniform distribution

over the set of binary polynomials and the error distribution ψ as the Gaussian distribution with variance σ^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 $\mathbf{a} \in R^k$ 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}.

B.1 Encryption

Suppose that there are N parties in a group. Let $\mathsf{ct} = (c_0, c_1)$ be an encryption of $m \in R_p$ generated by the randomness $r \leftarrow \chi$ and errors $e_0, e_1 \leftarrow \psi$. Then, it satisfies

$$c_0 + c_1 s = \Delta \cdot m + r \cdot (b + as) + (e_0 + e_1 s) = \Delta \cdot m + (re + e_0 + e_1 s) \pmod q,$$

where $b = \sum_i [b]_i$, $a = \sum_i [a]_i$, and $e = \sum_i [e]_i$. Since there are N parties in the group, the variance of e is $N \cdot \sigma^2$. Therefore, the encryption noise $e_{enc} = re + e_0 + e_1 s$ has the variance of

$$V_{enc} = \sigma^2 \cdot (\frac{nN}{2} + 1 + \frac{n}{2}) \approx \frac{nN\sigma^2}{2}.$$

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

B.2 Relinearization

We now provide an analysis of noise growth during relinearization. Recall that each secret key is sampled from the key distribution χ and error from D_σ. For simplicity, we analyze the noise growth of k-group case, each comprising N_i parties for $1 \le i \le k$. We also denote by $M = \sum_{1 \le i \le k} N_i$ the number of parties in all groups. With an input $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$ in Algorithm 2 of Section 5.1, we observe 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

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

where $e'_{i,j} = c_{i,j}\boxdot(r_i\cdot\mathbf{e}_{j,0}+s_j\cdot\mathbf{e}_{i,1}) \pmod q$. We denote by $V_g = \mathsf{Var}(\mathbf{g}^{-1}(a))$ where a is a uniform random variable over R_q. Similar to the previous analysis, we have $\mathsf{Var}(r_i) = \mathsf{Var}(s_i) = N_i/2$, and $\mathsf{Var}(\mathbf{e}_{i,k}) = N_i\cdot\sigma^2$. Therefore, the variance of $e'_{i,j}$ is obtained as $n^2d\cdot\sigma^2\cdot V_g\cdot N_i\cdot N_j$.

Finally, the variance of a relinearization error $e_{lin} = \sum_{1\le i,j\le k}(c''_i\boxdot\mathbf{e}_{i,2}+e'_{i,j})$ is obtained by

$$V_{lin} = nd\cdot\sigma^2\cdot V_g\cdot\left(n\sum_{1\le i,j\le k}N_iN_j + k\sum_{1\le i\le k}N_i\right) \approx n^2d\cdot\sigma^2\cdot V_g\cdot M^2$$

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. Then, we can obtain $d = \lceil \log q/\log p_i\rceil$ and the variance $V_g = \frac{1}{12d}\sum_{i=1}^{d} p_i^2$.

B.3 Multiplication

We only consider about the BFV scheme since, unlike the BFV scheme, the error is added to the message itself during the encryption in CKKS scheme. Let $\overline{\mathsf{ct}}_i$ is an encryption of the message m_i for $i = 1, 2$ which are input ciphertexts of multiplication. The ciphertext $\overline{\mathsf{ct}}_i$ satisfying $\langle\overline{\mathsf{ct}}_i, \overline{\mathsf{sk}}\rangle = q\cdot I_i + \Delta\cdot m_i + e_i$ for $I_i = \lfloor\frac{1}{q}\langle\overline{\mathsf{ct}}_i,\overline{\mathsf{sk}}\rangle\rceil$ and some e_i. We can consider $\frac{1}{q}\cdot\overline{\mathsf{ct}}_i$ as an uniform random variable over $\frac{1}{q}\cdot R_q^{k+1}$ which variance is about $\frac{1}{12}$. Also, as same as relinearization analysis, we consider about k-group case, comprising N_i parties for $1\le i\le k$. Then, the variance of $\overline{\mathsf{sk}}$ is $Var(\overline{\mathsf{sk}}) = \frac{1}{2}\sum_{1\le i\le k}N_i$ and we can calculate the variance of I_i by $\mathsf{Var}(I_i)\approx\frac{1}{12}(1+\frac{1}{2}nM)\approx\frac{1}{24}nM$.

The result of multiplication without relinearization satisfies

$$\begin{aligned}
&\langle\overline{\mathsf{ct}}_1\otimes\overline{\mathsf{ct}}_2, \overline{\mathsf{sk}}\otimes\overline{\mathsf{sk}}\rangle = \langle\overline{\mathsf{ct}}_1,\overline{\mathsf{sk}}\rangle\cdot\langle\overline{\mathsf{ct}}_2,\overline{\mathsf{sk}}\rangle \\
=&\Delta^2\cdot m_1m_2 + q\cdot(I_1e_2+I_2e_1) + \Delta\cdot(m_1e_2+m_2e_1) + e_1e_2 \pmod{q\cdot\Delta}
\end{aligned}$$

After mutiplying scaling factor $\frac{p}{q}$ with plaintext modulus p, the result $\overline{\mathsf{ct}}_{mul}$ satisfies

$$\begin{aligned}
&\langle\overline{\mathsf{ct}}_{mul}, \overline{\mathsf{sk}}\otimes\overline{\mathsf{sk}}\rangle = \langle\left\lfloor\frac{p}{q}\cdot\overline{\mathsf{ct}}_1\otimes\overline{\mathsf{ct}}_2\right\rceil, \overline{\mathsf{sk}}\otimes\overline{\mathsf{sk}}\rangle \\
&= \Delta\cdot m_1m_2 + (p\cdot(I_1e_2+I_2e_1) + (m_1e_2+m_2e_1)) + \Delta^{-1}\cdot e_1e_2 + e_{rd}
\end{aligned}$$

where $e_{rd} = \langle \frac{p}{q} \cdot \overline{\mathsf{ct}}_1 \otimes \overline{\mathsf{ct}}_2 - \overline{\mathsf{ct}}_{mul}, \overline{\mathsf{sk}} \otimes \overline{\mathsf{sk}} \rangle$. Therefore, the multiplication error e_{mul} is observed by

$$e_{mul} = p \cdot (I_1 e_2 + I_2 e_1) + (m_1 e_2 + m_2 e_1) + \Delta^{-1} \cdot e_1 e_2 + e_{rd}$$

From above equation, the first term $p \cdot (I_1 e_2 + I_2 e_1)$ dominates the whole multiplication error. If we write $V_e = \max\{\mathsf{Var}(e_1), \mathsf{Var}(e_2)\}$, then the variance of multiplication error is

$$V_{mul} \leq 4np^2 \cdot \mathsf{Var}(I_i) \cdot V_e \approx \frac{1}{6} n^2 p^2 M \cdot V.$$

The relinearization error has a fixed size depending on the parameters, but the multiplication error amplifies by a certain ratio as the computation progresses. Therefore, the total noise is eventually dominated by the multiplication error unless V_e is very small (e.g. fresh ciphertext).