

Reusable Dynamic Multi-Party Homomorphic Encryption

Jung Hee Cheon
Seoul National University & CryptoLab Inc.
Seoul, Republic of Korea
jhcheon@snu.ac.kr

Seunghong Kim
Samsung SDS
Seoul, Republic of Korea
0815roy@gmail.com

Hyeongmin Choe
CryptoLab Inc.
Seoul, Republic of Korea
hyeongmin.choe@cryptolab.co.kr

Yongdong Yeo
Seoul National University
Seoul, Republic of Korea
yongdong@snu.ac.kr

ABSTRACT

Homomorphic Encryption (HE) is a promising primitive for evaluating arbitrary circuits while keeping the user’s privacy. We investigate how to use HE in the multi-party setting where data is encrypted with several distinct keys. One may use the Multi-Key Homomorphic Encryption (MKHE) in this setting, but it has space/computation overhead of $O(n)$ for the number of users n , which makes it impractical when n grows large. On the contrary, Multi-Party Homomorphic Encryption (MPHE) is the other Homomorphic Encryption primitive in the multi-party setting, where the space/computation overhead is $O(1)$; however, is limited in terms of ciphertext reusability and dynamicity, that ciphertexts are encrypted just for a group of parties and cannot be reused for other purposes, and that additional parties cannot join the computation dynamically.

Contrary to MKHE, where the secret key owners engage only in the decryption phase, we consider a more relaxed situation where the secret key owners can communicate before the computation. In that case, we can reduce the size of a ciphertext and the evaluation complexity from $O(n)$ to $O(1)$ as in a single-key HE setting. We call this primitive as *Reusable Dynamic Multi-Party Homomorphic Encryption*, which is more suitable in real-world scenarios.

We show that 1) the procedures before the computation can be done in a very few rounds of communications, 2) the evaluation/space complexities are independent of the number of users, and 3) the functionalities are as efficient as MKHE, with asymptotic analysis and with implementation.

KEYWORDS

Multi-Party Homomorphic Encryption, Multi-Key Homomorphic Encryption, Data Lake, Cloud Computing

1 INTRODUCTION

Homomorphic Encryption (HE) [8, 10, 16, 18, 24, 25] enables computing over the encrypted ciphertexts without decryption. Based on this nature, HE is widely used for delegating computations to a server while keeping the users’ privacy. One of the possible applications of HE is Privacy-Preserving Machine Learning (PPML) [26, 31, 36, 37].

Machine Learning (ML) requires data from many sources, possibly including sensitive private data, e.g., face images, medical data, and various user logs. Hence, a combination of HE and ML may allow training and inferencing on private data while eliminating concerns about privacy. In a privacy-preserving manner, providing an inference result on personal data, sometimes referred to as ML as a Service (MLaaS), can be securely implemented by using HE—the data owner encrypts their data and delegates the inference computation over the encrypted data to the model owner—and is widely studied from theory [6, 22, 52] to practice [4, 7, 12, 28, 29]. On the other hand, PPML training requires accumulating and aggregating lots of data from possibly various users, and thus privacy should be preserved in a multi-party setting, securely against not only the party who computes but also other parties who participate in the computation.

In some scenarios like Federated Learning (FL), a sub-field of ML focusing on collaborative training scenarios, multiple parties participate in the model training but locally with their own data, and iteratively by a fraction of the whole parties. That is, the training is divided into several phases, and at every phase, some users are invited to participate in the global model training by using their own data. To this end, the users could dynamically join the training, while the data (including the locally trained result, which is biased and may leak personal data) from each user should not be revealed to the computing server or other users, which makes HE in the multi-party setting to be desirable.

Another realistic requirement for HE in the multi-party setting can be found in the privacy-preserving Digital Asset Management (DAM) scenario [38]. In the DAM scenario, each party sends the encrypted private data to the server, and the server obliviously uses the data multiple times for computing different circuits. This scenario is similar to the existing concept of *Data Lake* [17, 46, 48, 50], where big data from different resources are accumulated. The multiple calls of data for different purposes do not require the parties to set up, encrypt, and send to the server multiple times, but only once. It is natural to extend the data-centric computing scenario also to the multi-party, privacy-preserving settings for HE, where the users store their encrypted data in the server, the data lake, under their own secret keys.

In this work, we specifically consider the homomorphic computation scenario as follows: 1) the data owners *encrypt* their data with a *public key encryption* scheme where the secret key is owned by themselves and stores the ciphertexts and their public keys to the server; 2) when a set of parties agree on a joint computation

This work was conducted while Hyeongmin Choe and Seunghong Kim were affiliated with Seoul National University.

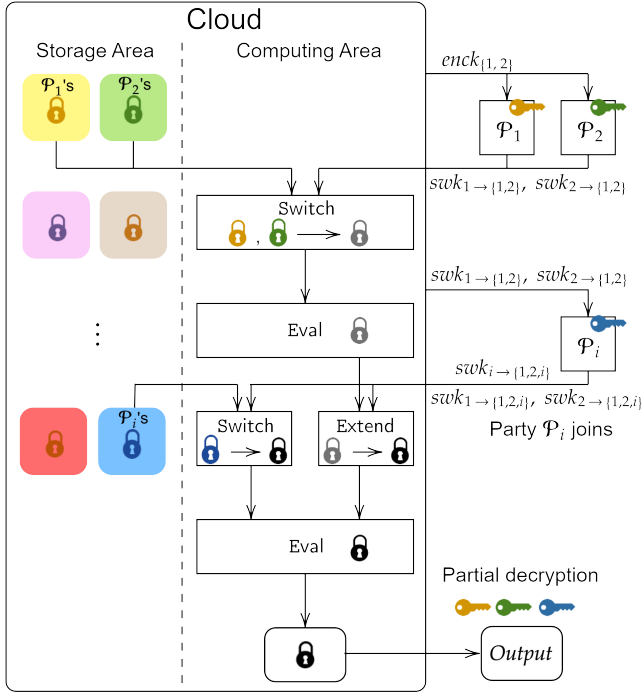


Figure 1: Procedures of Reusable Dynamic Multi-Party Homomorphic Encryption.

using their data, a common public key for HE is generated, and the parties provide public switching keys that can be used to switch the ciphertexts of individual parties to a HE ciphertext, so that the homomorphic computation could be possible; 3) when the HE public keys and the ciphertexts are ready, we do the homomorphic computation over the encrypted data; 4) then obtain the result by jointly decrypting the resulting ciphertext.

Currently, there are two different but similar primitives of HE in the multi-party setting: Multi-Key HE (MKHE) [11, 13, 34, 41] and Multi-Party HE (MPHE) [3, 42, 44, 45].¹ MPHE has a rigid user structure that should be pre-determined before the protocol starts. On the other hand, MKHE has a structure that grows linearly to quadratically in the number of parties, leading to a similar magnitude of inefficiency. In this regard, most of the currently available MPHEs are efficient in a static setting where the parties are fixed but have no dynamicity (in the sense that no new part can join after the keys are generated) and ciphertext reusability (in the sense that the ciphertexts can not be reused for a different set of parties). For the MKHEs, they can be used for this scenarios since they have dynamicity and ciphertext reusability; however they are inefficient in terms of the computational complexity and the size of the ciphertexts and the public keys.

1.1 Our Contributions

In this paper, we present the following contributions.

¹As mentioned in [35], still the terminology for HE in the multi-party setting has not been agreed in the literature, so we follow their classifications with the terms ‘MPHE’ and ‘MKHE’.

- We define a new primitive, Reusable Dynamic MPHE (rdMPHE), which allows ciphertext reusability and dynamicity for homomorphic computation. We formally define the semantic security of rdMPHE and give its real-world applications.
- We construct a rdMPHE scheme by extending the MPHE scheme of [35] based on RLWE-based HE. To this end, we first construct a Dynamic MPHE (dMPHE) scheme as a building block, then extend it to rdMPHE. We prove its semantic security based on the decision-RLWE assumption, which was also the case for the base MPHE scheme. We note that the construction of dMPHE can be of independent interest.
- We compare the existing HE solutions, including our rdMPHE, and study the situations in which each solution may be beneficial.
- We implemented our rdMPHE scheme in GO language, based on the code from [35], and benchmarked the latencies and sizes. To compare with our rdMPHE solution, we also partly extended the code from [35] to allow their MKHE scheme to be run in dynamic situations when new parties are joining.

1.2 Technical Overview

We start with the RLWE-based HE scheme with a ciphertext in a polynomial ring \mathcal{R}_Q^2 , where $\mathcal{R} = \mathbb{Z}[x]/(x^N + 1)$, where N is a power-of-two integer and Q is a positive integer modulus, and $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$. We use a polynomial $\mathbf{s} \in \mathcal{R}$ as a secret key, where the norm of \mathbf{s} , defined on a vector representation of polynomial \mathbf{s} , is small. A ciphertext of a plaintext $\mathbf{m} \in \mathcal{R}_t$ is a Ring Learning With Errors (RLWE) sample added by a plaintext, as $(\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$, where \mathbf{a} is uniformly chosen, and \mathbf{b} satisfies $\mathbf{a}\mathbf{s} + \mathbf{b} = \Delta \cdot \mathbf{m} + \mathbf{e}$ in \mathcal{R}_Q for $\Delta = \lfloor q/t \rfloor$ and a small random error \mathbf{e} sampled from a distribution χ_{err} . From the hardness of the decision-RLWE problem, it is computationally infeasible to distinguish (\mathbf{a}, \mathbf{b}) from a uniform random tuple sampled from \mathcal{R}_Q^2 . All known RLWE-based HE schemes are based on this structure, which is naturally additive and can be modified to multiplicative with an additional public key hiding some information on \mathbf{s} .

In RLWE-based Multi-Key HE (MKHE), a ciphertext is a tuple of polynomial elements $(\mathbf{a}_1, \dots, \mathbf{a}_n, \mathbf{b}) \in \mathcal{R}_Q^{(n+1)}$ where $\sum_i \mathbf{a}_i \mathbf{s}_i + \mathbf{b} = \Delta \cdot \mathbf{m} + \mathbf{e} \in \mathcal{R}_Q$ for n the number of parties. Each public key also consists of $n + 1$ components, sometimes generated with a Common Reference String (CRS), and thus, the overall computational complexity of MKHE grows linearly to quadratically in the number of parties. However, we note that if all of the \mathbf{a}_i s except one are zero, it can be seen as a ciphertext with respect to a party’s key. Thus, a new party can easily join, but with the cost of extending the ciphertexts and keys length to $|I| + 1$.

On the other hand, RLWE-based Multi-Party HE (MPHE) has exactly the same ciphertext structure as the single-key HE. Thus, it has an efficient computational complexity that does not depend on the number of parties. However, as the ciphertexts are encrypted by a single secret key that no one knows among the parties, the public keys should be jointly generated using a CRS, and also, it cannot be extended when a new party wants to join with his keys.

For this reason, Park [45] suggested a conversion between MKHE and MPHE to take advantage of the two. However, the conversion cannot be composed; a round-trip conversion from MPHE to MKHE

to MPHE was not applicable, since the MKHE after the first conversion is not exactly the MKHE. It can be rather seen as a special-case MGHE [35], with two groups; one has n parties, but the other has only one. It can be made applicable by generating the switching keys from a group to the whole parties and a party to the whole parties but requiring all the parties to be online—they encrypt their own secret key with the common key for the whole parties.

Our dMPHE scheme can be understood as an efficient round-trip conversion from MPHE to MPHE, which can be (almost) unlimitedly composable:

$$\text{MPHE}_n \rightarrow \text{MGHE}_{n,1,1,\dots,1} \rightarrow \text{MPHE}_{n+k_1} \rightarrow \dots \rightarrow \text{MPHE}_{n+k_1+\dots+k_\ell},$$

where the subscript denotes the number of parties for MPHE and the number of parties in each group for MGHE. In the following, let us focus on the first two conversions, and use k instead of k_1 .

For the second conversion, we are required to switch a ciphertext encrypted under a party \mathcal{P}_i 's key ($i \in [n]$) to a ciphertext under a common key for the $n+k$ parties. This can be done by using the key-switching technique in the HE, but requiring a key-switching key $\text{swk}_{[n] \rightarrow [n+k]}$, which is basically an encryption of a common secret key $\mathbf{s}_{[n]}$ of the n parties, encrypted under the common secret key $\mathbf{s}_{[n+k]}$ of the $(n+k)$ parties, i.e. if $\text{swk}_{[n] \rightarrow [n+k]} = (\mathbf{a}, \mathbf{b})$, then it satisfies $\mathbf{a}\mathbf{s}_{[n+k]} + \mathbf{b} = P \cdot \mathbf{s}_{[n]} + \mathbf{e}$ in \mathcal{R}_{PQ} , where \mathbf{e} is a small error polynomial, and P is an auxiliary modulus for keys. We denote $\mathbf{s}_{[n]} = \sum_{i \in [n]} s_i$.

In this work, we use the *plus 1 trick* to generate the switching keys, while keeping the original n parties offline and requiring the joining parties to operate independently:

- (1) we first encrypt 0 with a common public key $\text{pk}_{[n]}$ to get a ciphertext $(\mathbf{a}, \mathbf{b}) \in \mathcal{R}_{PQ}^2$, but in a higher modulus for the keys,
- (2) we then add P to the first component to get a ciphertext $(\mathbf{a}' = \mathbf{a} + P, \mathbf{b})$, encrypting $P \cdot \mathbf{s}_{[n]}$ under a common secret $\mathbf{s}_{[n]}$: $\mathbf{a}'\mathbf{s}_{[n]} + \mathbf{b} = \mathbf{a}\mathbf{s}_{[n]} + \mathbf{b} + P \cdot \mathbf{s}_{[n]} = P \cdot \mathbf{s}_{[n]} + \mathbf{e}$.
- (3) we are now able to add the RLWE samples $\mathbf{a}'s_j + \mathbf{e}_j \in \mathcal{R}_Q$ for $j \in [n+1, n+k]$ to the second component \mathbf{b} , to have

$$\mathbf{a}'\mathbf{s}_{[n]} + \mathbf{b}' = P \cdot \mathbf{s}_{[n+k]} + \mathbf{e}',$$

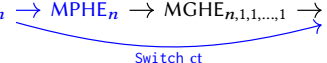
in \mathcal{R}_{PQ} , as desired.

We note that the plus 1 trick moves the secret key to the plaintext domain, hence was used for IND-CCA or IND-CPA^D attacks against HE, where the decryption oracle was given. However, it can be used for the keys in the semi-honest setting, which can be obviously obtained by anyone.

The light of this construction is that the original parties do not need to work, and only the joining parties work. The new joining parties are, in general, required to be registered to the server, and thus, it is reasonable to work for the joining in addition, and the communication round will be fused.

Such a dMPHE scheme enjoys the dynamicity and the compactness at the same time; however, it is somewhat limited compared to MKHE: there is no ciphertext reusability. Thus, we again extend the composed conversion in the opposite direction, by one:

$$\text{MKHE}_n \rightarrow \text{MPHE}_n \rightarrow \text{MGHE}_{n,1,1,\dots,1} \rightarrow \text{MPHE}_{n+k} \rightarrow \dots$$



For this extension, one should also consider the third conversion, since it is unlikely that if one wants to (re)use a ciphertext, he/she has to do the conversion from the beginning, which becomes infeasible when many parties are joined. Thus we are required to update the switching keys $\text{swk}_{i \rightarrow [n]}$ into $\text{swk}_{i \rightarrow [n+k]}$ for $i \in [n]$. They are both encrypting s_i , but with different keys $\mathbf{s}_{[n]}$, and $\mathbf{s}_{[n+k]}$, thus it can be switched via a double-key-switching key (a switching key for a switching key). However, this requires another auxiliary modulus, say R , to have the double-key-switching key over the ring \mathcal{R}_{PQR} . Having a limited modulus margin for λ -bit secure RLWE instance, this wastes an amount of available modulus.

We, instead, use *key-switching with sk_i* technique: updating the switching key by adding the $\mathbf{a}_i + \mathbf{e}_i$ to the \mathbf{b} part, which is an RLWE sample with respect to a random element \mathbf{a} .

Finally, our rdMPHE scheme enables us to take both the advantages of MKHE and MPHE over the entire evaluation procedures by converting between them efficiently.

Code Availability. Our Go implementation code of rdMPHE is publicly available at an anonymous GitHub repository: <https://github.com/Anonymous-rdmphe/rdMPHE.git>.

1.3 Related Works

Multi-Key Homomorphic Encryption (MKHE). Lopez-Alt et al. [41] came up with the first MKHE scheme based on NTRU HE scheme. It was followed by multiple multi-key variants of GSW scheme. Brakersi and Perlman et al. [11] proposed LWE-based MKHE which has shorter ciphertexts than previous works. Chen et al. [13] designed the multi-key version of TFHE.

The main advantage of MKHE lies in its flexibility of the participating parties. Despite its strong dynamicity, MKHE has been considered to be far from practical: the ciphertext size grows linearly or quadratically to the number of parties, which causes the communication and computation complexity to grow as well. There have been lines of work to improve the efficiency of MKHE, including [11, 13], but by far, MKHE is considered to be less efficient than MPHE in general.

Multi-Party Homomorphic Encryption (MPHE). Asharov et al. [3] suggested the first MPHE scheme from BGV. Mouchet et al. [44] proposed the simplified construction from BFV and recently improved the work by applying t -out of n threshold structure [42]. Park [45] modified the key generation algorithm to reduce the interaction in the Setup phase and additionally suggested the conversion between MPHE and MKHE.

Mouchet et al. [44] has several advantages in terms of low communication cost and no interaction required between parties in circuit computation. Kwak et al. [34] made further improvements by rendering the relinearization to be implemented in a single round and the key generation to be conducted without communication between parties. To the best of our knowledge, however, the dynamicity of the participating parties remains an open problem of MPHE so the reuse of encrypted ciphertexts for various circuits is hard with MPHE.

Multi-Group Homomorphic Encryption (MGHE). To compute a ciphertext under two different keys, Aloufi et al. [2] merged MKHE and MPHE ciphertexts into a ciphertext, each is for the model

	Reusability	Dynamicity	Ctxt Cpct.	PubKey Cpct.	Round-optimal	Security
MPHE [35, 42]	✗	✗	✓	✓	✓	✓
MKHE [32]	✓	✓	✗	✓	✓	✓
MGHE [35]	▲ (limited)	✓	✗	✓	✗	✓
Liu et al. [40]	✓	▲ (bounded)	✓	✗	✓	✓
rdMPHE (Ours)	✓	✓	✓	✓	✗	✓

Table 1: Comparison of the existing HE in the multi-party setting. The ciphertext reusability, dynamicity, ciphertext and public key compactness, round-optimality, and security against semi-honest adversaries are shown.

owners and clients, respectively. Such merge can also be obtained from Park’s conversion from MPHE to MKHE [45], which is later extended by Kwak et al. [34] to a more general notion called MGHE. In MGHE, participants are divided into multiple groups. Within a group, MPHE is used with a single key per group; on the other hand, between the groups, MKHE is used, treating each MPHE ciphertext as a component of the MKHE ciphertext.

Paper Organization. In Section 2, we define the notations and the formalisms of HE and recap the necessary related techniques. Then, we describe our homomorphic computation scenario with the threat model, and define the Reusable Dynamic MPHE and its formalisms with real-world applications in Section 3. We introduce our Dynamic MPHE scheme in Section 4 and use it as a building block for the Reusable Dynamic MPHE scheme in Section 5 with correctness and security analysis. In Section 6, we compare our Reusable Dynamic MPHE with other HE in the multi-party setting and give the GO implementation results for our scheme with benchmarks, which are available in the public domain. We also compare the benchmark result with the prior works.

2 PRELIMINARIES

2.1 Notations

We denote a set of positive integers from 1 to N as $[N]$. Vectors and polynomials are denoted in bold fonts. For an index set I and a set of polynomials $\{s_i\}_{i \in I}$, we denote s_I to be their sum $\sum_{i \in I} s_i$.

Throughout the paper, we will interchangeably use the two notations sk and s for the same secret key, for better readability. The secret and error distributions are denoted as χ_{sk} and χ_{err} , respectively. We denote \mathcal{M} as a plaintext space and \mathcal{C} as a ciphertext space. We let \mathcal{R}_Q refer to a quotient ring $\mathbb{Z}_Q[x]/\Phi_M(x)$, where $\Phi_M(x)$ is a M -th cyclotomic polynomial and Q is a positive integer modulus. If $M = 2N$ is a power-of-two integer, then $\Phi_M(x) = x^N + 1$, which is the most popularly used case.

We denote n to be the number of parties participating in a computation unless new parties are joined at some point. Each party is denoted as \mathcal{P}_i for $i \in [n]$, where an index set $I \subset [n]$ is more generally used.

2.2 Ring Learning with Errors (RLWE)

Let χ be a distribution defined over $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and \mathcal{R}_Q be a ring $\mathbb{Z}_Q[X]/(X^N + 1)$, where Q is a positive integer. Let DG_σ be a discrete gaussian distribution of variance σ over \mathcal{R}_Q and DG_σ samples N polynomial coefficients independently. Throughout this paper, we use DG_σ for the error distribution χ_{err} . The Ring Learning

with Errors (RLWE) associated with the parameter $(N, Q, \chi_{sk}, \chi_{err})$ is that for given polynomially many samples of either (\mathbf{a}, \mathbf{b}) or $(\mathbf{a}, -\mathbf{a}s + \mathbf{e})$ are computationally indistinguishable, where $\mathbf{a}, \mathbf{b} \leftarrow \mathcal{R}_Q$, $s \leftarrow \chi_{sk}$, and $\mathbf{e} \leftarrow \chi_{err}$.

2.3 Homomorphic Encryptions in the Multi-Party Setting

Below, we recap the formalisms of (Fully) Homomorphic Encryption ((F)HE), Multi-Party Homomorphic Encryption (MPHE), and Multi-Key Homomorphic Encryption (MKHE). Since they share most of the structures, we first give a definition for MKHE, which is most generally defined among the three, and then study the remaining.

Multi-Key Homomorphic Encryption. MKHE is defined as follows.

Definition 2.1 (Multi-Key Homomorphic Encryption [41]). A Multi-Key Homomorphic Encryption (MKHE) is a tuple of efficient algorithms $(\text{KeyGen}, \text{Enc}, \text{Eval}, \text{Dec})$ run by (possibly) multiple parties $\{\mathcal{P}_i\}_{i \in I}$ with the following specifications:

- **KeyGen** takes as inputs a security parameter 1^λ , run by \mathcal{P}_i , outputs a secret key sk_i and a public key pk_i ;
- **Enc** takes as inputs a common public key pk_I and a plaintext $m \in \mathcal{M}$, and outputs a ciphertext ct_I (or ct in short);
- **Eval** takes as inputs a common public key pk_I , a circuit C , and a tuple of ciphertexts ct_I^1, \dots, ct_I^k where k is the number of input wires of C , and outputs a ciphertext ct_I ;
- **Dec** takes as inputs a set of secret keys $\{sk_i\}_{i \in I}$ and a ciphertext ct_I , and outputs a plaintext \mathbf{m} .

Correctness (informal, [44]). We require that for a negligibly small $\epsilon > 0$, for any random coin used for KeyGen, for any arithmetic circuit C , and for any plaintexts $m^1, \dots, m^k \in \mathcal{M}$ where k is the number of input wires of C , the following holds with probability $\geq 1 - \epsilon$ over $ct_I^\ell := \text{Enc}_{pk_I}(m^\ell)$ for each $\ell \leq k$:

$$\text{Dec}_{\{sk_i\}_{i \in I}} \left(\text{Eval}_{pk_I}(C, (ct_I^1, \dots, ct_I^k)) \right) = C(m^1, \dots, m^k).$$

Semantic Security (informal, [44]). When assuming the KeyGen is done following the specifications, we require that for any adversarial subset $P \subset \{\mathcal{P}_i\}_{i \in I}$ of parties of size less or equal to $|I| - 1$, and for any two messages $m^1, m^2 \in \mathcal{M}$, the advantage of the adversary in distinguishing between distributions $\text{Enc}_{pk_I}(m^1)$ and $\text{Enc}_{pk_I}(m^2)$ should be smaller than $2^{-\lambda}$, where λ is a security parameter.

We note that the Dec algorithm should be run in a distributed way, as the secret key owners will not hand over their secret keys.

Thus, Dec is commonly instantiated via partial decryptions, each run by a party \mathcal{P}_i , outputting partially decrypted ciphertexts μ_i , and are combined by any (possibly third-party) receiver.

(Fully) Homomorphic Encryption. We call an MKHE scheme a single-key HE scheme if the index set I is a fixed set of size 1. Single-key HE is generally constructed based on lattices, using the RLWE-based encryption [9, 16, 24] or the LWE-based encryption [19, 23]. As a result, each ciphertext has a modulus Q , which is set corresponding to the parameters for the single-key HE to ensure semantic security. Throughout the paper, we focus on the constructions based on RLWE, which is much more practical when the amount of data is huge so that it can be batched in the RLWE ciphertexts.

The homomorphic multiplication consumes a modulus in all of the state-of-the-art implementations of RLWE-based HE [5, 21, 49]. Thus, to ensure the correctness of a homomorphic computation of a certain depth of a circuit, the modulus should be set large enough. However, if the modulus is too big, the HE scheme becomes not semantically secure. So, in practice, we use the largest possible modulus and can homomorphically compute circuits having a depth less than a fixed bound. To enable a longer computation, the bootstrapping technique is used. Depending on the possibility of the bootstrapping, the single-key HE can be categorized into two: Somewhat HE (SHE) and Fully HE (FHE).

One of the properties of HE, the *composability* of the unit homomorphic operations, guarantees that homomorphic addition and multiplication are possible; then, any circuit can be computed over the homomorphic encryption, regardless of its practicality and efficiency.

Multi-Party Homomorphic Encryption. MPHE is identically defined with MKHE but with an additional algorithms called CombKey:

- CombKey takes as inputs a set of public keys $\{\text{pk}_i\}_{i \in I}$, and outputs a common public key pk_I (or pk in short),

and use the common public key instead of the set of individual public keys, everywhere. We note that the evaluation process is also defined equivalently to a single-key HE. Hence, the ciphertext size may reach the same size (if we use the same parameters) as the single-key HE independent of the number of parties. The state-of-the-art constructions indeed have a constant ciphertext size, which does not depend on the number of parties.

As the MPHE algorithm can also be instantiated with $\{\text{pk}_i\}_{i \in I}$ rather than a common public key pk , MPHE is sometimes called *compact MKHE*, meaning that it has a constant ciphertext size. As in [35], we use the terms ‘MPHE’ and ‘MKHE’ to classify the MKHE schemes based on ciphertext sizes, distinguishing between constant and linear in the number of parties.

In the below, we recap the dynamic MPHE with additional property, which originated from the dynamic MKHE of [14].

Definition 2.2 (Dynamic Multi-Party Homomorphic Encryption [14]). A Dynamic Multi-Party Homomorphic Encryption (Dynamic MPHE) is an MPHE additionally with efficient algorithms ExtKeyGen and Extend with the following specifications:

- ExtKeyGen, run by $\{\mathcal{P}_i\}_{i \in I'}$, takes as inputs a set of public keys $\{\text{pk}_i\}_{i \in I'}$, and a set of secret keys $\{\text{sk}_i\}_{i \in I'}$, and outputs a public extension key $\text{extk}_{I \rightarrow I'}$, where $I \subseteq I'$;
- Extend takes as inputs a public extension key $\text{extk}_{I \rightarrow I'}$ and a (fresh, extended, or evaluated) ciphertext ct_I (encrypted under $\{\text{sk}_i\}_{i \in I}$), and a set of public keys $\{\text{pk}_i\}_{i \in I'}$, and outputs a ciphertext $\text{ct}_{I'}$ (encrypted under $\{\text{sk}_i\}_{i \in I'}$).

Correctness and Semantic Security (informal). The correctness and semantic security of Dynamic MPHE are identical to that of MPHE, except that the ciphertexts ct_I^f can be fresh, extended, or evaluated ciphertexts encrypted under $\{\text{sk}_i\}_{i \in I}$.

We again note that Dynamic MKHE is also similarly defined without the CombKey algorithm.

2.4 Key Switching

We briefly introduce the key switching technique in the RLWE-based (fully) homomorphic encryptions, which will be heavily utilized in our constructions.

Key switching is a homomorphic operation that transforms a ciphertext encrypting a plaintext m under a secret key into a ciphertext encrypting the same plaintext but under a different secret key. To key-switch a ciphertext $\text{ct} = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$, without decrypting it, one requires a public key-switching key $\text{swk} = (\alpha, \beta) \in \mathcal{R}_{PQ}^2$. The additional modulus P is called the auxiliary modulus, which enables us to reduce the error added during the key-switching.

Specifically, for a polynomial $\alpha \in \mathcal{R}_Q$, we let $\beta + \alpha s' = P \cdot \mathbf{s} + \epsilon \in \mathcal{R}_{PQ}$, for some error $\epsilon \in \mathcal{R}$. A ciphertext $\text{ct} = (\mathbf{a}, \mathbf{b})$ also has a similar structure, having $\mathbf{b} + \mathbf{a}\mathbf{s} = \Delta \cdot m + \mathbf{e} \in \mathcal{R}_Q$ for some error $\mathbf{e} \in \mathcal{R}$. The key-switching procedure outputs a ciphertext $\text{ct}' = (\mathbf{a}', \mathbf{b}') := (0, \mathbf{b}) + [P^{-1} \cdot \mathbf{a} \cdot (\alpha, \beta)] \in \mathcal{R}_Q^2$, which satisfies

$$\begin{aligned} \mathbf{b}' + \mathbf{a}'\mathbf{s}' &= \mathbf{b} + [P^{-1} \cdot \mathbf{a} \cdot \beta] + [P^{-1} \cdot \mathbf{a} \cdot \alpha] \mathbf{s}' \\ &= \mathbf{b} + P^{-1} \cdot \mathbf{a} \cdot (\beta + \alpha \mathbf{s}') + \mathbf{e}_{\text{rnd}} \\ &= \mathbf{b} + \mathbf{a}\mathbf{s} + P^{-1} \cdot \mathbf{a} \cdot \epsilon + \mathbf{e}_{\text{rnd}} = \Delta \cdot \mathbf{m} + \mathbf{e}', \end{aligned}$$

where $\mathbf{e}' = \mathbf{e} + P^{-1} \cdot \mathbf{a} \cdot \epsilon + \mathbf{e}_{\text{rnd}}$ for some rounding error $\mathbf{e}_{\text{rnd}} = \mathbf{e}_\alpha + \mathbf{e}_\beta \mathbf{s}'$ and $\mathbf{e}_\alpha, \mathbf{e}_\beta$ having their coefficients in $[-0.5, 0.5]$. If the Hamming weight of \mathbf{s}' is set small enough, and P is set large enough, then we have a small bound for the error \mathbf{e}' , and thus ct' becomes a ciphertext encrypting the same message under \mathbf{s}' .

Since the switching keys and the ciphertexts have a similar format, the RLWE encryption structure, one can use a ciphertext encrypting $P \cdot \mathbf{s}$ for the switching keys, but with larger error and larger modulus, say PQ , to key-switch a ciphertext modulo Q .

The result can be extended to the case when using the gadget decompositions, which dramatically decreases P and increases the possible maximum Q ,² allowing a longer depth of computations before bootstrappings. The high-level idea is to encrypt $P \cdot G_j \cdot \mathbf{s}'$ instead of $P \cdot \mathbf{s}$ for some gadget vector $\vec{G} = (G_1, \dots, G_d)$, which satisfies $\langle \vec{G}, h(\mathbf{a}) \rangle = \mathbf{a}$ for some function $h : \mathcal{R}_Q \rightarrow \mathcal{R}_Q^d$ and $\mathbf{a} \in \mathcal{R}_Q$.

²the maximum possible PQ depends on the choice of N and the secret key and error distributions for a fixed security level.

3 HOMOMORPHIC COMPUTATION SCENARIO WITH REUSABLE CIPHERTEXTS

In this section, we consider a real-world-aware Multi-Party Computation (MPC) scenario with reusable ciphertexts stored in a semi-honest cloud server. Then, we define a new primitive named Reusable Dynamic Multi-Party Homomorphic Encryption (rdMPHE), which can be seen as an extension of the existing homomorphic encryption in the multi-party setting to the real-world-aware scenario. We also define their correctness and semantic security. Lastly, we show that the semantic security of rdMPHE implies the security of the real-world-aware scenario using rdMPHE.

3.1 MPHE with KeySwitch

In real applications, we consider that multiple parties want to run a computation on their own private data jointly and do not want to leak any information other than the computed result. It is demanded that the ciphertexts are reusable and compact, and the homomorphic computation is efficient, and new parties can dynamically join at any time. In this setting, we define Reusable Dynamic Multi-Party Homomorphic Encryption (rdMPHE) as follows.

Definition 3.1 (Reusable Dynamic Multi-Party Homomorphic Encryption). A Reusable Dynamic Multi-Party Homomorphic Encryption (rdMPHE) is a tuple of efficient algorithms (KeyGen , CombKey , Enc , SwKeyGen , Switch , Eval , Dec , ExtKeyGen , Extend) run by (possibly) multiple parties $\{\mathcal{P}_i\}_{i \in I}$ with the following specifications:

- KeyGen , run by \mathcal{P}_i , outputs a secret key sk_i and a public key pk_i ;
- CombKey takes as inputs a set of public keys $\{pk_i\}_{i \in I}$, and outputs a common public key pk_I ;
- Enc takes as inputs a public key pk_i (or pk_I) and a plaintext $m \in \mathcal{M}$, and outputs a ciphertext ct_i (or ct_I);
- SwKeyGen , run by \mathcal{P}_i , takes as inputs a public key pk_I and a secret key sk_i where $i \in I$, and outputs a public switching key $swk_{i \rightarrow I}$;
- Switch takes as inputs a public switching key $swk_{i \rightarrow I}$ and a ciphertext ct_i encrypted under sk_i , and outputs ct_I ;
- Eval takes as inputs a common public key pk_I , a circuit C , and a tuple of ciphertexts ct_I^1, \dots, ct_I^k where k is the number of input wires of C , and outputs a ciphertext ct_I ;
- Dec takes as inputs a set of secret keys $\{sk_i\}_{i \in I}$ and a ciphertext ct_I , and outputs a plaintext m .
- ExtKeyGen , run by $\{\mathcal{P}_i\}_{i \in I' \setminus I}$, takes as inputs a set of public keys $\{pk_i\}_{i \in I'}$, a set of switching keys $swk_{i \rightarrow I}$ and a set of secret keys $\{sk_i\}_{i \in I' \setminus I}$, and outputs a public extension key $extk_{I \rightarrow I'}$ and updated public switching keys $swk_{i \rightarrow I'}$, where $I \subsetneq I'$ and $i \in I$;
- Extend takes as inputs a public extension key $extk_{I \rightarrow I'}$ and a (fresh, extended, or evaluated) ciphertext ct_I (encrypted under $\{sk_i\}_{i \in I}$), and a set of public keys $\{pk_i\}_{i \in I'}$, and outputs a ciphertext $ct_{I'}$ (encrypted under $\{sk_i\}_{i \in I'}$).

Correctness and Semantic Security (informal). Correctness and semantic security of Reusable rdMPHE are defined exactly the same as Dynamic MPHE, except that the ciphertexts ct_I^f can be fresh, key-switched, extended, or evaluated ciphertexts encrypted under $\{sk_i\}_{i \in I}$.

The rdMPHE can be seen as a variant of MPHE in the sense that the public keys are combined; however, the ciphertexts are first encrypted with each party's keys, as in MKHE. The ciphertexts are key-switched into a common key corresponding to a combination of parties, specific to the applications. Thus, the ciphertexts can be re-used for different homomorphic computations with other sets of participating parties, which saves huge communication costs for transmission of the ciphertexts.

3.2 Real-World Scenarios

Our rdMPHE can be utilized for privacy-preserving arbitrary circuit evaluation scenarios that require adding extra inputs during computation or high-efficiency computation. The following protocol outline presents how it can be effectively applied in real-world scenarios.

- **Data Accumulation:** Data providers generate their own secret key and corresponding public key set (KeyGen) and encrypt their data with their own secret key (Enc). The encrypted data is uploaded to a cloud with the public keys. This process can be done once, but data and the public keys can be used repeatedly for various computations.
- **Setup:** The server determines who will participate in the protocol, and generates a common public key set with the uploaded data in the cloud (CombKey). The generated common public key is broadcast to the participants. Each participant generates its own partial switching key as permission for utilizing their encrypted data in the cloud (SwKeyGen).
- **Computation:** The server switches the input ciphertexts to be encrypted under the common key (Switch) and performs the desired homomorphic evaluations (Eval). Depending on demands, a new data provider with its data can be added during the evaluation at any time by extending the rdMPHE structure by the new data provider (ExtKeyGen) and by the server (Extend).
- **Decryption:** When the evaluation ends, the output ciphertexts are sent to participants, and each participant performs partial decryption and outputs the result to a receiver (Dec).

3.3 Security Model

The security model using rdMPHE can achieve semantic security in the semi-honest adversary model, where the majority of parties and the server can be honest-but-curious adversaries. This means that correctness and security may not be obtained if the participating parties or the server do not follow the protocol. However, data privacy is still preserved even if some of the participating parties collude. We define the semantic security of rdMPHE to have computational indistinguishability against semi-honest adversaries.

3.4 Applications

In this section, we discuss the applications of rdMPHE. The strength of rdMPHE lies in its data-centric structure for the reusability of the encrypted data. In addition, rdMPHE advantages in both the lightness of a single-key HE and the flexibility of MKHE, with small communication before the evaluation.

Privacy-Preserving Machine Learning. When the proposed rdMPHE can be adopted for Privacy-Preserving Machine Learning (PPML) training, the above scenario is applied as follows.

The server wants to train a model with some part of the encrypted data accumulated in the cloud as input training data. The training data are encrypted by various data providers' keys, and the corresponding data providers may permit for using the data. The server computes the model training homomorphically, and new data with a new data provider can easily be added to improve the model knowledge without degradation of performances of the underlying HE during the training.

For privacy-preserving federated learning, each user trains its own data by itself, and each trained result is collected into the cloud. The locally trained pieces of information are aggregated to train a global model by the server, but the information may cause privacy issues unless they are encrypted. The locally trained information may be added to enhance the global model quality. The only difference to the aforementioned scenario is that the encrypted data collected to the cloud consists of locally trained results by each user.

Privacy-Preserving Digital Asset Management. One application of rdMPHE is privacy-preserving Data Lake [17, 46, 48, 50], where each user accumulates its encrypted raw data to the Data Lake without pre-processing, and the computing server can utilize the raw data stored in the Data Lake in future. For Data Lake protocol, Data Lake becomes the server in our case and a server may create synergy, including concepts of brand/product management, and Media Asset Management (MAM).

REMARK 1. *For the PPML training and Data Lake, one of our protocol's key features is the cloud data's re-usability. In particular, considering the fact that famous public training datasets (e.g. MNIST, CIFAR-10, etc.) are used a huge number of times, our protocol enables us to utilize them even for private data.*

Previous PPML works [30, 33, 37, 51] restricted the data providers from the beginning of the task or suffered from the low homomorphic evaluation efficiency. Our proposed scheme can handle the issues of maintaining the communication cost which does not depend on the number of parties.

4 DYNAMIC MPHE SCHEME

In this section, we introduce a Dynamic MPHE scheme that becomes a building block of the rdMPHE. We first recap the state-of-the-art MPHE scheme, which is used as a basic structure for our Dynamic MPHE, then construct a Dynamic MPHE with a ciphertext extension algorithm as an ad-hoc, based on the key-switching technique. Lastly, we give the correctness and the security proof.

4.1 Base MPHE Scheme

Our dynamic MPHE is constructed based on the MPHE scheme from [43] with a non-interactive relinearization key generation technique from [35] based on BFV [9] and CKKS [16] schemes.

- **MPHE.KeyGen:** For a party \mathcal{P}_i , a secret key sk_i is a small polynomial $\mathbf{s} \leftarrow \chi_{sk} (\in \mathcal{R})$, and a public key pk_i consists of
 - encryption key $enck_i = -\mathbf{a}_{enck} \mathbf{s}_i + \mathbf{e}_{enck,i} \in \mathcal{R}_Q$,

- rotation keys $autk_{\sigma,i} = -\mathbf{a}_{\sigma} \mathbf{s}_i + P \cdot \sigma(\mathbf{s}_i) + \mathbf{e}_{\sigma,i} \in \mathcal{R}_{PQ}$ with respect to automorphisms σ ,
- relinearization key $rlk_i = (rlk_{i,0}, rlk_{i,1}, rlk_{i,2}) \in \mathcal{R}_{PQ}^3$ for
 - * $rlk_{i,0} = -\mathbf{a}_{rlk,0} \mathbf{s}_i + \mathbf{e}_{rlk,i,0}$,
 - * $rlk_{i,1} = -\mathbf{a}_{rlk,0} \mathbf{r}_i + P \cdot \mathbf{s}_i + \mathbf{e}_{rlk,i,1}$,
 - * $rlk_{i,2} = -\mathbf{b}_{rlk,1} \mathbf{s}_i - P \cdot \mathbf{r}_i + \mathbf{e}_{rlk,i,2}$,

where all the common polynomials $\mathbf{a}_{(\cdot)} \in \mathcal{R}$ are induced from common reference strings, and all the errors $\mathbf{e}_{(\cdot)} \in \mathcal{R}$ are sampled from χ_{err} ;

- **MPHE.CombKey:** A common public key pk_I consisting of
 - common encryption key $enck_I = \sum_{i \in I} enck_i \in \mathcal{R}_Q$,
 - common rotation keys $autk_{\sigma,I} = \sum_{i \in I} autk_{\sigma,i} \in \mathcal{R}_{PQ}$,
 - common relinearization key $rlk_I = (rlk_{I,0}, rlk_{I,1}, rlk_{I,2}) = \sum_{i \in I} rlk_i \in \mathcal{R}_{PQ}^3$;
- **MPHE.Enc:** A RLWE public key encryption outputs $ct = \mathbf{u} \cdot (\mathbf{a}_{enck}, enck_I) + (\mathbf{e}_0, \Delta \mathbf{m} + \mathbf{e}_1) \in \mathcal{R}_Q^2$, where $\mathbf{u} \leftarrow \mathcal{R}$ and $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \chi_{err}$;
- **MPHE.Dec:** A decryption of a ciphertext $ct = (\mathbf{c}_0, \mathbf{c}_1) \in \mathcal{R}_Q^2$ is done in a distributed way:
 - (1) each party \mathcal{P}_i generates partial decryption $\mu_i = \mathbf{c}_0 \mathbf{s}_i + \mathbf{e}_i$ of ct , where $\mathbf{e}_i \leftarrow \chi_{smudge}$,
 - (2) aggregate the partial decryptions and outputs a plaintext $\sum_{i \in I} \mu_i + \mathbf{c}_1 \in \mathcal{R}_Q$.

We note that χ_{smudge} be an error distribution for a secure distributed decryption [20, 39], where the standard deviation is set $\lambda/2$ bit larger than that of the ciphertext error $\mathbf{c}_0 \cdot \sum_{i \in I} \mathbf{s}_i + \mathbf{c}_1$, to smudge the error not to reveal the secret, even when $|I| - 1$ parties collude.

The list of automorphisms σ varies depending on the applications, but in general, tens of automorphisms are included for faster linear operations. We also note that the automorphism and relinearization keys are also applicable in the RNS setting with gadget decompositions, i.e., constructions based on RNS-BFV [27] or RNS-CKKS [15]. For the sake of simplicity, we sometimes denote the keys as $swk_{s_I \rightarrow 0} := (\mathbf{a}_{enck}, enck_I)$ or $swk_{s_I \rightarrow \sigma(s_I)} := (\mathbf{a}_{\sigma}, autk_{\sigma,I})$ and so on, each having a format of $(\mathbf{a}, -\mathbf{a} \mathbf{s}_I + P \mathbf{s}' + \mathbf{e}) \in \mathcal{R}_{PQ}^2$ for some $\mathbf{s}' \in \mathcal{R}$, where $\mathbf{s}_I = \sum_{i \in I} \mathbf{s}_i$.

For homomorphic evaluation, we recall homomorphic addition, multiplication, and automorphisms, which can be composed. One can then evaluate arbitrary arithmetic circuits of longer depth by composing them. We note that the multiplication algorithm is a bit more costly than that in RLWE-based (F)HE schemes in order to generate the relinearization key non-interactively. For ciphertexts $ct = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$ and $ct' = (\mathbf{a}', \mathbf{b}') \in \mathcal{R}_Q^2$ encrypting plaintexts \mathbf{m}_0 and \mathbf{m}_1 in \mathcal{R} under $\{sk_i\}_{i \in I}$, homomorphic addition, multiplication, key switching, and automorphisms can be done as follows.

- **MPHE.Add(ct_I, ct_I'):** Outputs $(\mathbf{a} + \mathbf{a}', \mathbf{b} + \mathbf{b}') \in \mathcal{R}_Q^2$, a ciphertext encrypting $\mathbf{m}_0 + \mathbf{m}_1$;
- **MPHE.Mul $_{rlk_I}(ct_I, ct_I')$:**
 - (1) It tensor products the two ciphertexts and obtain $\mathbf{d} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2) = (\mathbf{b}\mathbf{b}', \mathbf{a}\mathbf{b}' + \mathbf{a}'\mathbf{b}, \mathbf{a}\mathbf{a}')$,
 - (2) Relinearizes by
 - (a) compute $\mathbf{d}'_2 = [P^{-1} \cdot (\mathbf{d}_2 \cdot rlk_0)] \in \mathcal{R}_Q$,

- (b) compute $(\mathbf{d}'_1, \mathbf{d}'_0) = \lfloor P^{-1} \cdot (\mathbf{d}'_2 \cdot (\mathbf{b}_{\text{rlk}}, \text{rlk}_2) + \mathbf{d}_2 \cdot (\text{rlk}_1, 0)) \rfloor + (\mathbf{d}_1, \mathbf{d}_0) \in \mathcal{R}_Q^2$, which is a ciphertext encrypting $\Delta \mathbf{m}_0 \cdot \mathbf{m}_1$,
- (c) outputs $\lfloor \Delta^{-1} \cdot (\mathbf{d}'_1, \mathbf{d}'_0) \rfloor \in \mathcal{R}_{Q/\Delta}^2$;
- MPHE.KS($\text{swk}_{s \rightarrow s'}, \text{ct}$): It outputs $(0, \mathbf{b}) + \lfloor P^{-1} \cdot \mathbf{a} \cdot \text{swk}_{s \rightarrow s'} \rfloor \in \mathcal{R}_Q^2$.
 - MPHE.Aut $_{\text{autk}_{\sigma, I}}$ (ct):
 - (1) It computes $\sigma(\text{ct}) = (\sigma(\mathbf{a}), \sigma(\mathbf{b})) \in \mathcal{R}_Q^2$,
 - (2) outputs $\text{MPHE.KS}(\text{swk}_{s_I \rightarrow \sigma(s_I)} = (\mathbf{a}_\sigma, \text{autk}_{\sigma, I}), \sigma(\text{ct})) \in \mathcal{R}_Q^2$.

Correctness and Semantic Security. For the correctness and the semantic security of the above base MPHE scheme, please refer to [35][Sections 4.3-5], the correctness and the semantic security proofs for MGHE. They directly apply to our base MPHE when the number of groups of the MGHE is 1. In a high-level view, the correctness proof is basically based on the correctness of each homomorphic arithmetic operation and its composability.

4.2 Dynamic MPHE Construction

Our Dynamic MPHE shares the main structures with the above algorithms but is modified a bit to introduce dynamicity. We give the two algorithms ExtKeyGen and Extend as an ad-hoc to the above MPHE as follows, defined for index sets $I \subseteq I'$.

- dMPHE.KeyGen: Identical to MPHE.KeyGen, except that enck_I is generated in \mathcal{R}_{PQ} instead of \mathcal{R}_Q ;
- dMPHE.CombKey: Identical to MPHE.CombKey, except that enck_I is now in \mathcal{R}_{PQ} instead of \mathcal{R}_Q ;
- dMPHE.Enc: Identical to MPHE.Enc, except that one can choose the resulting ciphertext not only to be in \mathcal{R}_Q^2 as in MPHE, but also in \mathcal{R}_{PQ}^2 ;
- dMPHE.Dec: Identical to MPHE.Dec;
- dMPHE.Eval: Identical to MPHE.Eval, except that the homomorphic multiplication and automorphisms cannot be applied to the ciphertexts in \mathcal{R}_{PQ}^2 ;
- dMPHE.ExtKeyGen($\text{enck}_I, \{\text{sk}_i\}_{i \in I' \setminus I}$):
 - (1) It encrypts 0 with enck_I , letting $(\mathbf{a}, \mathbf{b}) = \text{Enc}_{\text{enck}_I}(0) \in \mathcal{R}_{PQ}^2$, then let $\mathbf{a}' = \mathbf{a} + P \in \mathcal{R}_{PQ}$,
 - (2) compute $\text{extk}_{I \rightarrow I'} = (\mathbf{a}', \mathbf{b}) - \sum_{i \in I' \setminus I} (0, \mathbf{a}' s_i + \mathbf{e}_i)$, where $\mathbf{e}_i \leftarrow \chi_{\text{err}}$;
- dMPHE.Extend($\text{extk}_{I \rightarrow I'}, \text{ct}_I$): MPHE.KS($\text{extk}_{I \rightarrow I'}, \text{ct}_I$).

We note that only the new parties need to be online when generating the extension keys. That is, the original parties can remain offline and will only participate in the decryption phase.

In the settings using key switching keys with gadget decompositions, e.g., RNS setting with RNS gadgets or digit decompositions [15, 27], subtracting by P in the ExtKeyGen can be replaced by subtracting $P \cdot G_j$, where G_j are the gadgets.

Correctness of Dynamic MPHE (informal). Thanks to the composability of the base MPHE, it suffices to show whether the extended ciphertext and the original ciphertext are decrypted to the same plaintext with respect to the set of secret keys $\{\text{sk}_i\}_{i \in I'}$ and $\{\text{sk}_i\}_{i \in I}$, respectively. Since $\text{extk}_{I \rightarrow I'} = (\mathbf{a}', \mathbf{b}) - \sum_{i \in I' \setminus I} (0, \mathbf{a}' s_i + \mathbf{e}_i)$, it can

be rewritten as $\text{extk}_{I \rightarrow I'} = (\mathbf{a}', \mathbf{b}') \in \mathcal{R}_{PQ}^2$, where

$$\begin{aligned} \mathbf{a}' s_{I'} + \mathbf{b}' &= \mathbf{a}' s_{I'} + \mathbf{b} - \sum_{i \in I' \setminus I} (\mathbf{a}' s_i + \mathbf{e}_i) = (\mathbf{a}' s_I + \mathbf{b}) - \mathbf{e}_{I' \setminus I} \\ &= \mathbf{a} s_I + (\mathbf{a} s_I + \mathbf{b}) - \mathbf{e}_{I' \setminus I} = P \cdot s_I + \mathbf{e}' \end{aligned}$$

modulo PQ for some $\mathbf{e}' = \mathbf{e} - \mathbf{e}_{I' \setminus I}$, where \mathbf{e} is a freshly encrypted ciphertext error and $\mathbf{e}_{I' \setminus I} = \sum_{i \in I' \setminus I} \mathbf{e}_i$. Thus, the key-switching operation concludes the proof unless the aggregated error is too huge.

4.3 Semantic Security of DMPHE

When assuming the KeyGen, CombKey, and ExtKeyGen are done following the specifications, the ciphertexts that are freshly encrypted, extended, or evaluated ciphertexts do not leak any information non-negligibly in the security parameter.

THEOREM 4.1 (SEMANTIC SECURITY OF OUR DMPHE). *Assuming the hardness of the decision-RLWE Problem and the circular security assumption and that the parameters are set to achieve correctness, our dMPHE construction depicted in Section 4.2 is semantically secure.*

5 REUSABLE DYNAMIC MULTI-PARTY HOMOMORPHIC ENCRYPTION SCHEME

We extend our Dynamic MPHE into a family of MKHE, by introducing the ciphertexts encrypted under each party's key, then key-switch them into ciphertexts under a common key. This can be viewed as an MKHE to Dynamic MPHE conversion, which is an extension of Park's conversion [45]. While keeping the dynamicity and the ciphertext reusability, our Reusable Dynamic MPHE Scheme rdMPHE enjoys the compactness of the MPHE schemes with respect to the ciphertext sizes and the Homomorphic operation complexity.

5.1 Reusable Dynamic Multi-Party Homomorphic Encryption Construction

Here, we give our rdMPHE construction, which also shares the structures a lot with dMPHE.

- KeyGen: Identical to dMPHE.KeyGen;
- CombKey: Identical to dMPHE.CombKey;
- Enc: Identical to dMPHE.Enc;
- SwKeyGen($\text{enck}_I, \text{sk}_i$): If $i \in I$, it outputs $\text{swk}_{i \rightarrow I} := \text{Enc}_{\text{enck}_I}(P \cdot s_i) \in \mathcal{R}_{PQ}^2$;
- Switch($\text{swk}_{i \rightarrow I}, \text{ct}_i$): For $\text{ct}_i \in \mathcal{R}_Q^2$, it outputs dMPHE.KS($\text{swk}_{i \rightarrow I}, \text{ct}_i$);
- Eval: Identical to dMPHE.Eval;
- Dec: Identical to dMPHE.Dec;
- ExtKeyGen($\text{enck}_I, \{\text{sk}_i\}_{i \in I' \setminus I}$):
 - It first does identically with dMPHE.ExtKeyGen and generate $\text{extk}_{I \rightarrow I'}$,
 - for each switching key $\text{swk}_{i \rightarrow I} = (\mathbf{a}_i, \mathbf{b}_i)$, $i \in I$, it updates the key as $\text{swk}_{i \rightarrow I'} := \text{swk}_{i \rightarrow I} - \sum_{j \in I' \setminus I} (0, \mathbf{a}_j s_j + \mathbf{e}_{i,j})$, where $\mathbf{e}_{i,j} \leftarrow \chi_{\text{err}}$,
 - for the joining parties' switching keys, $\text{swk}_{i \rightarrow I'}$ for $i \in I'$, it runs $\text{swk}_{i \rightarrow I'} := \text{SwKeyGen}(\text{enck}_I, \text{sk}_i)$;
- Extend: Identical to dMPHE.Extend.

Correctness of rdMPHE. We first show that the `Switch` algorithm works correctly with the switching keys generated from `SwKeyGen`. Let $\text{swk}_{i \rightarrow I} = (\mathbf{a}_i, \mathbf{b}_i)$, then we have

$$\mathbf{b}_i = -\mathbf{a}_i \mathbf{s}_I + P \cdot \mathbf{s}_i + \mathbf{e}_i$$

in \mathcal{R}_{PQ} for some error \mathbf{e}_i and hence the `Switch`, which is basically the key switching operation, works. Hence, the `Switch` algorithm works correctly if the error term is relatively smaller than Δ . We note that the error has a larger standard deviation compared to the errors in automorphism keys; however, the difference is set to be negligible compared to the smudging error since this is added linearly, not exponentially.

We then show that the `Switch` algorithm works correctly if some new party joins with `ExtKeyGen`. The switching keys are updated from $\text{swk}_{i \rightarrow I}$ to $\text{swk}_{i \rightarrow I'}$ during `ExtKeyGen`. We assume the switching key $\text{swk}_{i \rightarrow I}$ is generated from the `SwKeyGen` algorithm for simplicity. Then, the new switching key can be rewritten as

$$\begin{aligned} \mathbf{a}_i \mathbf{s}_{I'} + \mathbf{b}'_i &= \mathbf{a}_i \mathbf{s}_{I'} + \mathbf{b}_i - \sum_{j \in I' \setminus I} (\mathbf{a}_i \mathbf{s}_j + \mathbf{e}_{i,j}) \\ &= \mathbf{a}_i \mathbf{s}_I + \mathbf{b}_i - \mathbf{e}_{I' \setminus I} \\ &= P \cdot \mathbf{s}_i + \mathbf{e}'_i \end{aligned}$$

where $\text{swk}_{i \rightarrow I} = (\mathbf{a}_i, \mathbf{b}_i)$, $\text{swk}_{i \rightarrow I'} = (\mathbf{a}_i, \mathbf{b}'_i)$, $\mathbf{e}'_i = \mathbf{e}_i - \mathbf{e}_{I' \setminus I}$, and $\mathbf{e}_{I' \setminus I} = \sum_{j \in I' \setminus I} \mathbf{e}_{i,j}$; hence the `Switch` algorithm works correctly if the error term is not too large compared than Δ .

5.2 Semantic Security of rdMPHE

Theorem 5.1 states the semantic security of our rdMPHE construction with respect to the newly introduced notions such as `Switch`.

THEOREM 5.1 (SEMANTIC SECURITY OF OUR rdMPHE). *Assuming the hardness of the decision-RLWE problem and the circular security assumption and that the parameters are set to achieve correctness, our rdMPHE construction depicted in Section 5.1 is semantically secure.*

6 ASYMPTOTIC ANALYSIS AND IMPLEMENTATION RESULTS

In this section, we first give an asymptotic analysis by comparing the known solutions for the scenario with reusable ciphertexts. We then study the cases in which each solution is favorable. Then, we give the implementation results which support our asymptotic analysis. The implementation codes are publicly available at the time of the submission.

6.1 Asymptotic Comparison between Known Solutions

We provide an asymptotic analysis for the computation and communication complexity of the HE in the multi-party setting. We compare MPHE, MKHE, and rdMPHE, in the setting that computations for M different sets of parties are required. In the scenario with reusable ciphertexts, the public key and the ciphertext reusability are one of the main aspects, and MKHE and rdMPHE have this property. Thus, for the complexity of computing and transmitting the public keys and the ciphertexts, we don't have to count them repeatedly, but only once. For MPHE, it cannot reuse the ciphertext as in MKHE or rdMPHE, however, the party's public key can be

reused. Hence, we extend this property to MPHE for a fair comparison. That is, we assumed that the public keys for each party in the MPHE are also shared once and can be aggregated when the set of participating parties is decided for each computation.

In this setting, we compare the complexity of the three concerning:

- Computation cost for public key generation, encryption, preparations (aggregation and key switching, if needed), and the homomorphic evaluations.
- Communication cost for public key and ciphertext transmission and preparations.
- Communication rounds for M computations.

6.1.1 Computational Cost. We first recall the computational complexity of running public key generation, encryption, key switchings, and homomorphic additions and multiplications. For the ring degree N , ciphertext modulus Q , the auxiliary modulus P , and the gadget rank d for the switching keys, it holds that $P \approx Q^{1/d}$ and thus, $d \log PQ \approx (d+1) \log Q$ holds. We assume M computations with different sets of parties, each requiring n_{ct} ciphertexts.

Unit Operations. The public key generation runs in $O(dN \log PQ)$ due to the $O(d)$ number of Hadamard multiplications for \mathcal{R}_{PQ} elements. Similarly, a single encryption runs in $O(N \log Q)$ or $O(dN \log PQ)$ depending on the resulting modulus.

The main factors of the key-switching algorithm are the $O(d)$ `ModUps` (from Q to PQ), $O(1)$ `ModDown` (from PQ to Q), and the $O(d)$ Hadamard multiplications in \mathcal{R}_{PQ} . The key-switching takes $O(dN \log N \log PQ)$ since each `ModUp` and `ModDown` takes NTTs, which require $N \log N \log PQ$ complexity.

A homomorphic addition takes $O(N \log Q)$, and a homomorphic rotation takes $O(dN \log N \log PQ)$ since it is the same as key switchings. A homomorphic multiplication takes $O(dN \log N \log PQ)$ since the key switching cost is again the dominant term compared to tensor products, which are $O(N \log Q)$.

MPHE. We first consider MPHE for its computational complexity. The public key generation is done only once, with a computational complexity of $O(dN \log PQ)$. The encryption is done after a common public key is generated by aggregation (which itself is just a set of additions), resulting in a ciphertext in \mathcal{R}_Q^2 , thus taking $O(MNn_{\text{ct}} \log Q)$ for the whole encryptions. For homomorphic evaluations, MPHE takes the same time as the unit complexity given above. We let C_{single} be a computational complexity of a circuit based on these unit operations. Then it should take $O(M \cdot C_{\text{single}})$. For the sake of simplicity, we let $C_{\text{single}} = c_{\text{single}} N \log Q$ since $N \log Q$ is a factor that exists everywhere, regardless of the operations. In total, it takes $O((Mc_{\text{single}} + Mn_{\text{ct}} + d)N \log Q)$.

MKHE. We then consider MKHE. The public key generation and the encryption are done only once, with a computational complexity of $O(dN \log PQ + n_{\text{ct}} \cdot N \log Q)$ for the whole encryptions, unless a new public key encryption needs to be done. In that case, it requires $O(n)$ times more for MKHE than MPHE or rdMPHE. For the sake of simplicity, we ignore them here. For homomorphic evaluations, MKHE takes $O(n)$ times more time than the unit complexity since the number of components in each ciphertext is $O(n)$ times more.

Thus, it takes $O(Mn \cdot C_{\text{single}})$. In total, it takes $O((Mc_{\text{single}}n + n_{\text{ct}} + d)N \log Q)$.

rdMPHE. We then consider our rdMPHE. The public key generation and the encryption are done only once, with a computational complexity of $O(dN \log PQ + n_{\text{ct}} \cdot N \log Q)$. For the preparation phase, the aggregation of the keys is just additions, and the switching key generation takes $O(dN \log PQ)$. A ciphertext key-switching requires $O(N \log N \log PQ)$, instead of $O(dN \log N \log PQ)$, since the $O(d)$ ModUp can be omitted since they are done identically for the M computations, hence, taking $O(Nn_{\text{ct}} \log N \log PQ)$ in total. Homomorphic evaluations should take $O(M \cdot C_{\text{single}})$ as in the MPHE. In total, it takes $O((Mc_{\text{single}} + n_{\text{ct}} \log N \cdot (1 + 1/d) + d)N \log Q) = O((Mc_{\text{single}} + n_{\text{ct}} \log N + d)N \log Q)$.

6.1.2 Communication Cost. A public key has a size of $O(dN \log PQ) = O(dN \log Q)$, where a ciphertext has a size of $O(N \log Q)$, except for the MKHE ciphertexts. A ciphertext encrypted by a party's key has the same size, but the one encrypted by a common public key or the one after evaluation has a size of $O(nN \log Q)$.

MPHE. Since the MPHE public key is transmitted once, by all the parties, it has a communication cost of $O(ndN \log Q)$. The ciphertexts are transmitted for each set of parties, thus having a communication cost of $O(Mn_{\text{ct}}N \log Q)$. In total, it has $O((Mn_{\text{ct}} + nd)N \log Q)$ cost. The communication round is 3 since each party first sends their public key, then receives a common public key, and then sends the ciphertexts. We omit the decryptions.

MKHE. Since the MKHE public key and the ciphertexts encrypted with each party's public key are transmitted once, the communication cost is $O((n_{\text{ct}} + nd)N \log Q)$. The communication round is 1 since each party sends their public key, with the ciphertexts.

rdMPHE. Since the rdMPHE public key and the ciphertexts encrypted with each party's public key are transmitted once, the communication cost is $O((n_{\text{ct}} + nd)N \log Q)$. The switching keys need to be generated after the common public key is generated, and thus it transmits $O(MndN \log PQ)$. The total cost is $O((n_{\text{ct}} + Mdn)N \log Q)$. The communication round is 3 since each party first sends their public key and the ciphertexts, then receives a common public key, and then sends the switching keys.

6.1.3 Complexity Dependencies in MGHE Ciphertext Structure. For MGHE, however, it is hard to compare apple to apple—since the complexities vary depending on the ciphertext and the group structures. In the worst-case, all the asymptotic complexities are the same as MKHE if we replace n by n_g , the number of groups. When the ciphertexts are well-structured, for example, if two ciphertexts have many of their components 0, the complexity may decrease a bit. In the best case, only two out of the $n_g + 1$ components of each ciphertext are non-zero, and the complexity becomes the same as the MPHE.

We summarize the asymptotic analysis done in this section in Table 2, showing their ciphertext sizes and the complexity of the basic homomorphic operations, and in Table 3, showing the computation and communication complexity in the reusable ciphertexts setting given in the first part of this section.

	Ctxt	Homomorphic operations		
		Add.	Mult.	Aut.
MPHE [35, 43]	$O(1)$	$O(1)$	$O(d \log N)$	$O(d \log N)$
MKHE [32]	$O(n)$	$O(n)$	$O(nd \log N)$	$O(nd \log N)$
MGHE [35]	$O(n_g)$	$O(n_g)$	$O(n_g d \log N)$	$O(n_g d \log N)$
rdMPHE (Ours)	$O(1)$	$O(1)$	$O(d \log N)$	$O(d \log N)$

Table 2: Comparison of the variants of HE in the multi-party setting, where the common factor $N \log Q$ is omitted. For details, see Sections 6.1.1 and 6.1.2.

Since every homomorphic evaluation of rdMPHE and Liu et al. [40] follows the evaluation of a single-key HE scheme, the asymptotic complexity of every evaluation does not depend on the number of parties N . In other words, an rdMPHE ciphertext is always of the form $(ct_0, ct_1) \in \mathcal{R}_{QL}^2$, for any number of participating parties.

On the other hand, every asymptotic computational complexity of MKHE depends on the number of participating parties. Since a ciphertext in MKHE is of the form $(ct_0, ct_1, \dots, ct_N) \in \mathcal{R}_{QL}^{N+1}$ depending on N , the addition between ciphertexts (Add) and key-switching (KS) becomes N -times heavier than a single-key HE scheme. Moreover, the multiplication between ciphertexts (Mult) in MKHE is much heavier than in the aforementioned evaluations. This is because the MKHE multiplication between ciphertexts needs a special relinearization procedure as its subroutine. It converts the cross terms, that correspond to the secret key $sk_i \cdot sk_j$, into the standard MKHE term, that corresponds to a single secret key sk_i . Hence, the asymptotic complexity of the MKHE multiplication between ciphertexts follows $O(N)$.

6.2 Implementation Results

In this section, we report the implementation result of our rdMPHE and compare it with the prior solutions supporting dynamicity, such as MKHE. We excluded the comparison with MGHE in this section. This is because the performance of MGHE highly relies on its predefined group structure, the fair apple-to-apple comparison is hard to be made although our implementation encompasses the implementation of MGHE.

We implemented our rdMPHE scheme based on the MGHE implementation [47] of [35], which is written in Lattigo library [49] version 2.3.0. It basically includes our dMPHE scheme, without the key switching keys from a party's key to a public key. To compare the latencies of each component in our rdMPHE with MKHE, we use the MKHE implementation in [47].

Specifically, we implemented the following:

- dMPHE constructed in Section 4,
- rdMPHE constructed in Section 5,
- MKHE extended from [47] to have the joining functionality.

Experimental Setup. All our experiments are performed on an Intel Xeon Silver 4114 CPU at 2.2GHz with 260GB of RAM with Linux. Our benchmark is performed with the CKKS scheme [16] and BFV scheme [8, 24]. The comparison is performed under the HE parameter set (ring dimension N , key moduli bit $\log PQ$, gadget

	Reusables.	Computation comp.	Communication comp.	Rounds	Dynamicity
MPHE [35, 43]	pk_i	$O(Mc_{\text{single}} + Mn_{\text{ct}} + d)$	$O(Mn_{\text{ct}} + nd)$	3	✗
MKHE [32]	pk_i, ct_i	$O(Mc_{\text{single}}n + n_{\text{ct}} + d)$	$O(n_{\text{ct}} + nd)$	1	✓
rdMPHE (Ours)	pk_i, ct_i	$O(Mc_{\text{single}} + n_{\text{ct}} \log N + d)$	$O(n_{\text{ct}} + Mnd)$	3	✓

Table 3: Comparison of the existing solutions when applied to the scenario with reusable ciphertexts scenario, where the public keys and ciphertexts are already uploaded and maybe reused, with M different sets of parties. The common factor $N \log Q$ is omitted from the complexities. For details, see Sections 6.1.1 and 6.1.2.

rank d) for $(2^{14}, 439, 6)$ and $(2^{15}, 880, 14)$. The key distribution χ_{sk} samples a secret key’s coefficients from the ternary set $\{-1, 0, 1\}$ with equal probability $1/3$ and the error distribution χ_{err} samples the error with a standard deviation 3.2. The parameter set achieves 128 bits of semantic security against the current best-known attacks, estimated with Lattice estimator [1] following the prior works [35, 43].

Benchmark Result. Each of the benchmark evaluations is performed for $2 \sim 2^5$ participating parties for each scheme. We measured the latency of KeyGen + CombKey + SwKeyGen with the number of parties 1, 3, 7, 15, 31, the latencies of Switch/Extend, ExtKeyGen assuming a single party joins, and the latencies of Add, Mult, Rot with the extended number of parties 2, 4, 8, 16, 32. All the evaluations (Add, Mult, Rot, Conj) are performed with ciphertexts at the top level. For the rotation (Rot), we generated the rotation key with respect to the left rotation by a single slot and timed only for the rotation that can be performed by a single key-switching. We perform KS for a single ciphertext during the Extend phase.

Table 4 shows that our proposed rdMPHE outperforms every homomorphic computation compared to MKHE. We emphasize that the performance degradation of homomorphic evaluations upon the number of participating parties does not occur in rdMPHE, while MKHE does. This is from the computational complexity of MKHE that grows linearly to the number of participating parties unlike rdMPHE, as analyzed in Section 6.1.1 and also can be checked in Table 4 and Figure 2.

The main drawback of our rdMPHE comes from the computational cost for Switch, Extend, and ExtKeyGen, which are zero in MKHE. However, Figure 2 shows that the above costs, performed only once per joining, are small enough even compared to a single ciphertext-ciphertext multiplication of MKHE. In addition, we note that the performance of switching key swk_i related factors highly depends on the HE parameter gadget rank d . One can check that the KeyGen latency increased a bit compared to that of MKHE due to additionally generating switching key $swk_{i \rightarrow j}$. However, such extra overheads will be reduced when a parameter set with a smaller gadget rank d is used, including the latencies for ExtKeyGen.

In more detail, rdMPHE, MPHE, single-group MGHE, and single-key HE share the same ciphertext/key structures for evaluation.³ In addition, the multi-group homomorphic evaluations in MGHE with the number of groups n_g can be considered as the multi-party homomorphic evaluations in MKHE with the number of parties n_g that grows with the length of ciphertext and the computational

³In fact, the relinearization algorithm of the single-key HE is slightly faster than that of rdMPHE/MPHE/single-group MGHE. This is because the usual HE does not use the 3-tuple rlk structure [34], which has no advantages for the single-key setting.

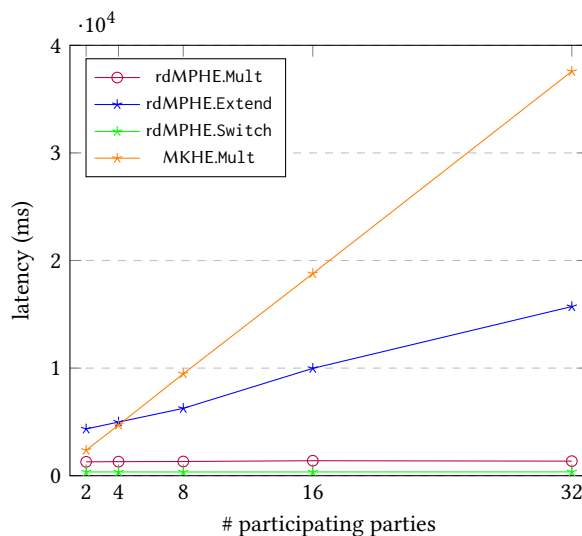


Figure 2: Comparison of the key operations of MKHE and our rdMPHE. The homomorphic multiplication latencies of rdMPHE and MKHE; and joining (ExtKeyGen and Extend) and switching (Switch) latencies of rdMPHE are given over the varying number of participating parties (meaning that each number n corresponds to a party is joining to a group of $n - 1$ parties). The values are based on the implementation of the BFV-based scheme [8, 24] with a parameter set ($N=2^{15}$, $\log(PQ) = 880$, $d = 14$) from the prior works.

complexity. The memory/latency efficiency of rdMPHE is important since the number of parties M might be huge in many use cases. We note that the aforementioned ciphertext size is also directly related to the communication cost.

To summarize, rdMPHE can efficiently evaluate an arbitrary circuit regardless of the number of participating parties, with the advantages of MKHE preserved: data reusability and extension of parties during the computation.

REFERENCES

- [1] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology* 9, 3 (2015), 169–203. <https://doi.org/doi:10.1515/jmc-2015-0016>
- [2] Asma Aloufi, Peizhao Hu, Harry W. H. Wong, and Sherman S. M. Chow. 2021. Blindfolded Evaluation of Random Forests with Multi-Key Homomorphic Encryption. *IEEE Transactions on Dependable and Secure Computing* 18, 4 (2021), 1821–1835. <https://doi.org/10.1109/TDSC.2019.2940020>

Scheme ($\log_2 PQ$)	# Parties (ms)	2		4		8		16		32	
		Ours	MKHE	Ours	MKHE	Ours	MKHE	Ours	MKHE	Ours	MKHE
BFV (439)	KeyGen	4981.9	4868.6	5552.5	5402.0	6830.9	6282.8	9276.4	8402.0	14531.0	12476.0
	Switch/Extend	38.2	0	39.0	0	38.4	0	39.3	0	39.9	0
	ExtKeyGen	190.0	0	311.0	0	429.3	0	831.3	0	1477.4	0
	Add	0.6	1.5	1.0	2.7	0.7	5.5	1.0	8.8	1.0	17.5
	Mult	150.2	275.3	150.0	539.0	150.9	1031.2	152.0	2111.8	163.1	4329.8
	Rot	41.9	83.4	44.0	169.9	42.8	335.3	42.9	677.2	44.2	1376.5
CKKS (439)	KeyGen	853.1	708.3	1333.4	1072.4	2307.3	1739.8	4336.7	3138.3	8430.1	5686.4
	Switch/Extend	37.5	0	37.6	0	37.5	0	38.0	0	37.7	0
	ExtKeyGen	227.9	0	297.1	0	458.0	0	760.7	0	1371.6	0
	Add	0.8	1.8	1.0	2.3	1.6	4.5	0.7	10.4	0.9	18.8
	Mult	69.0	129.3	70.4	248.1	71.0	504.4	73.2	1014.3	72.7	2016.4
	Rot	40.7	81.0	41.2	165.8	41.5	337.9	42.0	677.9	41.2	1341.9
BFV (880)	KeyGen	28415.9	28175.3	34031.1	33104.3	45862.9	42732.1	70989.9	61148.3	119392.8	101217.4
	Switch/Extend	349.5	0	349.5	0	345.7	0	349.0	0	355.3	0
	ExtKeyGen	1977.3	0	2615.6	0	3894.6	0	7608.4	0	13347.3	0
	Add	4.0	9.6	5.3	14.2	3.9	23.4	5.9	36.1	4.5	96.8
	Mult	1288.9	2368.3	1306.7	4708.2	1320.4	9472.7	1392.8	18785.1	1353.3	37580.5
	Rot	372.7	721.1	368.7	1463.9	369.4	2954.0	370.3	5920.9	365.8	11624.0
CKKS (880)	KeyGen	7585.0	6662.6	12028.1	10070.5	21451.6	16290.2	40435.9	28053.5	82317.0	52405.4
	Switch/Extend	355.4	0	338.1	0	334.6	0	338.5	0	346.1	0
	ExtKeyGen	2078.6	0	2760.8	0	4195.9	0	7065.4	0	13492.1	0
	Add	5.6	9.0	4.2	12.8	4.2	25.6	5.1	41.9	4.0	90.6
	Mult	585.2	1096.6	577.1	2254.4	565.8	4077.8	560.9	8623.0	680.3	17642.9
	Rot	356.3	711.9	361.7	1486.1	354.0	2850.1	353.9	5881.9	375.7	11792.4

Table 4: Latency comparison between our Reusable Dynamic Multi-Party Homomorphic Encryption (rdMPHE) and Multi-Key Homomorphic Encryption (MKHE) from [35]. For each party number $n = 2, 4, 8, 16, 32$, the latency of KeyGen (with CombKey and SwKeyGen if needed) is measured with $n - 1$ parties; the latencies of Switch/Extend, or ExtKeyGen are measured when a single party is joining, and the latencies of Add, Mult, Rot are measured after the joining, with n parties. All the latencies are given in milliseconds (ms).

- [3] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Heidelberg, 483–501. https://doi.org/10.1007/978-3-642-29011-4_29
- [4] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. 2017. Secure approximation of edit distance on genomic data. *BMC medical genomics* 10 (2017), 55–67.
- [5] Ahmad Al Badawi, Jack Bates, Flávio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, R. V. Saraswathy, Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. In *WAHC*. Library available at <https://www.openfhe.org/>. IACR Cryptol. ePrint Arch. 2022/915, dated March 12, 2024..
- [6] M. Barni, C. Orlandi, and A. Piva. 2006. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th Workshop on Multimedia and Security (Geneva, Switzerland) (MM&Sec '06)*. Association for Computing Machinery, New York, NY, USA, 146–151. <https://doi.org/10.1145/1161366.1161393>
- [7] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *CRYPTO 2018, Part III (LNCS, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 483–512. https://doi.org/10.1007/978-3-319-96878-0_17
- [8] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Heidelberg, 868–886. https://doi.org/10.1007/978-3-642-32009-5_50
- [9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption without Bootstrapping. Cryptology ePrint Archive, Report 2011/277. <https://eprint.iacr.org/2011/277>.
- [10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 309–325. <https://doi.org/10.1145/2090236.2090262>
- [11] Zvika Brakerski and Renen Perlman. 2016. Lattice-Based Fully Dynamic Multi-key FHE with Short Ciphertexts. In *CRYPTO 2016, Part I (LNCS, Vol. 9814)*, Matthew Robshaw and Jonathan Katz (Eds.). Springer, Heidelberg, 190–213. https://doi.org/10.1007/978-3-662-53018-4_8
- [12] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-Preserving Classification on Deep Neural Network. Cryptology ePrint Archive, Report 2017/035. <https://eprint.iacr.org/2017/035>.
- [13] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Multi-Key Homomorphic Encryption from TFHE. In *ASIACRYPT 2019, Part II (LNCS, Vol. 11922)*, Steven D. Galbraith and Shihō Moriai (Eds.). Springer, Heidelberg, 446–472. https://doi.org/10.1007/978-3-030-34621-8_16
- [14] Yuling Chen, Sen Dong, Tao Li, Yilei Wang, and Huiyu Zhou. 2021. Dynamic Multi-Key FHE in Asymmetric Key Setting From LWE. *IEEE Transactions on Information Forensics and Security* 16 (2021), 5239–5249. <https://doi.org/10.1109/TIFS.2021.3127023>
- [15] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2019. A Full RNS Variant of Approximate Homomorphic Encryption. In *SAC 2018 (LNCS, Vol. 11349)*, Carlos Cid and Michael J. Jacobson Jr. (Eds.). Springer, Heidelberg, 347–368. https://doi.org/10.1007/978-3-030-10970-7_16
- [16] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT 2017*,

- Part I (LNCS, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Heidelberg, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15
- [17] Mandy Chessell, Ferd Scheepers, Nhan Nguyen, Ruud van Kessel, and Ron van der Starre. 2014. Governing and managing big data for analytics and decision makers. *IBM Redguides for Business Leaders* 252 (2014).
 - [18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *ASIACRYPT 2016, Part I (LNCS, Vol. 10031)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). Springer, Heidelberg, 3–33. https://doi.org/10.1007/978-3-662-53887-6_1
 - [19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* 33, 1 (Jan. 2020), 34–91. <https://doi.org/10.1007/s00145-019-09319-x>
 - [20] Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. 2022. Efficient Threshold FHE with Application to Real-Time Systems. *Cryptology ePrint Archive*, Report 2022/1625. <https://eprint.iacr.org/2022/1625>.
 - [21] cryptolabinc. 2023. HEaAn private AI: Homomorphic Encryption Library. Available at <https://hub.docker.com/r/cryptolabinc/heaan>. CryptoLab, Inc..
 - [22] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (New York, NY, USA) (*ICML '16*). JMLR.org, 201–210.
 - [23] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *EUROCRYPT 2015, Part I (LNCS, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 617–640. https://doi.org/10.1007/978-3-662-46800-5_24
 - [24] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *Cryptology ePrint Archive*, Report 2012/144. <https://eprint.iacr.org/2012/144>.
 - [25] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Ph. D. Dissertation, Stanford University. crypto.stanford.edu/craig.
 - [26] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 201–210. <https://proceedings.mlr.press/v48/giladbachrach16.html>
 - [27] Shai Halevi, Yuriy Polyakov, and Victor Shoup. 2019. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. In *CT-RSA 2019 (LNCS, Vol. 11405)*, Mitsuru Matsui (Ed.). Springer, Heidelberg, 83–105. https://doi.org/10.1007/978-3-030-12612-4_5
 - [28] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 1209–1222. <https://doi.org/10.1145/3243734.3243837>
 - [29] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics* 11 (2018), 23–31.
 - [30] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics* 11 (10 2018). <https://doi.org/10.1186/s12920-018-0401-7>
 - [31] Dongwoo Kim and Cyril Guyot. 2023. Optimized Privacy-Preserving CNN Inference With Fully Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* 18 (2023), 2175–2187. <https://doi.org/10.1109/TIFS.2023.3263631>
 - [32] Taechan Kim, Hyesun Kwak, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. 2023. Asymptotically Faster Multi-Key Homomorphic Encryption from Homomorphic Gadget Decomposition. In *ACM CCS 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM Press, 726–740. <https://doi.org/10.1145/3576915.3623176>
 - [33] R. Hari Kishore, A. Chandra Sekhar, Pramoda Patro, and Debabrata Swain. 2023. Multi-Key Privacy-Preserving Training and Classification using Supervised Machine Learning Techniques in Cloud Computing. In *2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, 1–6. <https://doi.org/10.1109/ACCAI58221.2023.10200291>
 - [34] Hyesun Kwak, Dongwon Lee, Yongsoo Song, and Sameer Wagh. 2021. A Unified Framework of Homomorphic Encryption for Multiple Parties with Non-Interactive Setup. *Cryptology ePrint Archive*, Report 2021/1412. <https://eprint.iacr.org/2021/1412>.
 - [35] Hyesun Kwak, Dongwon Lee, Yongsoo Song, and Sameer Wagh. 2024. A General Framework of Homomorphic Encryption for Multiple Parties with Non-interactive Key-Aggregation. In *Applied Cryptography and Network Security*, Christina Pöpper and Lejla Batina (Eds.). Springer Nature Switzerland, Cham, 403–430.
 - [36] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 12403–12422. <https://proceedings.mlr.press/v162/lee22e.html>
 - [37] Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. 2023. HETAL: efficient privacy-preserving transfer learning with homomorphic encryption. In *Proceedings of the 40th International Conference on Machine Learning* (<conf-loc>Honolulu</city>, <state>Hawaii</state>, <country>USA</country>, </conf-loc>) (*ICML '23*). JMLR.org, Article 786, 26 pages.
 - [38] Wei-Shan Lee, John A. Hsiu-Chun Hsu, and Pao-Ann Hsiung. 2022. SPChain: A Smart and Private Blockchain-Enabled Framework for Combining GDPR-Compliant Digital Assets Management With AI Models. *IEEE Access* 10 (2022), 130424–130443. <https://doi.org/10.1109/ACCESS.2022.3227969>
 - [39] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. 2022. Securing Approximate Homomorphic Encryption Using Differential Privacy. In *CRYPTO 2022, Part I (LNCS, Vol. 13507)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Heidelberg, 560–589. https://doi.org/10.1007/978-3-031-15802-5_20
 - [40] Wenchao Liu, Tanping Zhou, Long Chen, Hongjian Yang, Jiang Han, and Xiaoyuan Yang. 2024. Round efficient privacy-preserving federated learning based on MKFHE. *Computer Standards & Interfaces* 87 (2024), 103773. <https://doi.org/10.1016/j.csi.2023.103773>
 - [41] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 2012. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *44th ACM STOC*, Howard J. Karloff and Toniann Pitassi (Eds.). ACM Press, 1219–1234. <https://doi.org/10.1145/2213977.2214086>
 - [42] Christian Mouchet, Elliott Bertrand, and Jean-Pierre Hubaux. 2023. An Efficient Threshold Access-Structure for RLWE-Based Multiparty Homomorphic Encryption. *J. Cryptol.* 36, 2 (mar 2023), 20 pages. <https://doi.org/10.1007/s00145-023-09452-8>
 - [43] Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. 2021. Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies* 2021, 4 (2021), 291–311.
 - [44] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. 2021. Multiparty Homomorphic Encryption from Ring-Learning-with-Errors. *PoPETs* 2021, 4 (Oct. 2021), 291–311. <https://doi.org/10.2478/popets-2021-0071>
 - [45] Jeongeun Park. 2021. Homomorphic Encryption for Multiple Users with Less Communications. *IEEE Access* PP (2021), 1–1. <https://api.semanticscholar.org/CorpusID:237425395>
 - [46] Christoph Quix, Rihan Hai, and Ivan Vatrov. 2016. GEMMS: A Generic and Extensible Metadata Management System for Data Lakes.
 - [47] SNUUCP. 2023. SNU-MGHE. Available at <https://github.com/SNUUCP/snu-mghe.git>.
 - [48] Ignacio G. Terrizzano, Peter M. Schwarz, Mary Roth, and John E. Colino. 2015. Data Wrangling: The Challenging Journey from the Wild to the Lake. In *Conference on Innovative Data Systems Research*. <https://api.semanticscholar.org/CorpusID:17462093>
 - [49] Tuneinsight. 2021. Lattigo v2.3.0. Available at <https://github.com/lidsec/lattigo>.
 - [50] Coral Walker and Hassan Alrehamy. 2015. Personal Data Lake with Data Gravity Pull. In *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, 160–167. <https://doi.org/10.1109/BDCloud.2015.62>
 - [51] Guowen Xu, Guanlin Li, Shangwei Guo, Tianwei Zhang, and Hongwei Li. 2023. Secure Decentralized Image Classification With Multiparty Homomorphic Encryption. *IEEE Transactions on Circuits and Systems for Video Technology* 33, 7 (2023), 3185–3198. <https://doi.org/10.1109/TCSVT.2023.3234278>
 - [52] Jiawei Yuan and Shucheng Yu. 2014. Privacy Preserving Back-Propagation Neural Network Learning Made Practical with Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems* 25, 1 (2014), 212–221. <https://doi.org/10.1109/TPDS.2013.18>