IACR Communications in Cryptology
ISSN 3006-5496, Vol. 3, No. 1, 37 pages.

https://doi.org/10.62056/a0qj89krz

Check for updates

# Multi-Party Homomorphic Encryption with Dynamicity and Ciphertext Reusability

Jung Hee Cheon[1,2], Hyeongmin Choe[a,3], Seunghong Kim[4] and Yongdong Yeo[b,1]

[1] Seoul National University, Seoul, Korea
[2] CryptoLab Inc., Seoul, Korea
[3] University of Luxembourg, Esch-sur-Alzette, Luxembourg
[4] Samsung SDS, Seoul, Korea

**Abstract.** Homomorphic Encryption (HE) enables computation on encrypted data while preserving privacy. We explore its application in the *multi-party setting*, where data is stored in the cloud under several distinct keys. For $n$ parties, Multi-Key HE (MKHE) supports such scenarios but incurs $O(n)$ space and computational overhead, making it impractical for large-scale use. Conversely, Multi-Party HE (MPHE) achieves constant $O(1)$ overhead but is typically limited by a static group structure: ciphertexts are traditionally tied to a fixed set of parties, which poses challenges for dynamically joining new members or reusing existing ciphertexts for different party sets.

To address these limitations, we first construct a Dynamic MPHE (dMPHE) scheme that allows new parties to join, while the original parties are not required to remain online. Our construction bridges the gap between existing MPHE and MKHE frameworks while achieving superior efficiency compared to prior dynamic MPHE attempts. Building on this, we introduce Reusable Dynamic MPHE (rdMPHE), a new primitive that simultaneously supports dynamicity and ciphertext reusability. We implement both schemes based on the RLWE assumption. Our analyzes and experiments demonstrate that rdMPHE maintains $O(1)$ efficiency while overcoming the scalability and static constraints of previous MKHE and MPHE. To support open research, our code has been made publicly available.

**Keywords:** Multi-Party Homomorphic Encryption · Multi-Key Homomorphic Encryption · Data Lake · Cloud Computing

## 1 Introduction

Homomorphic Encryption (HE) [Gen09, BGV12, Bra12, FV12, CKKS17, DM15, CGGI16] enables computing over the encrypted ciphertexts without decryption. Thanks to this property, HE is widely adopted for outsourcing computations to servers while preserving user privacy. One of the possible applications is Privacy-Preserving Machine Learning (PPML) [GBDL+16, LLL+22, KG23, LLK+23].

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 can allow training and inferencing on private data while eliminating concerns about privacy. In a privacy-preserving manner, providing an inference result

E-mail: jhcheon@snu.ac.kr (Jung Hee Cheon), hyeongmin.choe@uni.lu (Hyeongmin Choe), yongdong@snu.ac.kr (Yongdong Yeo)
[a] Corresponding author.
[b] Primary contact for implementation and source code.

This work is licensed under a "CC BY 4.0" license.
Received: 2026-02-03 Accepted: 2026-04-20

on personal data, sometimes referred to as ML as a Service (MLaaS), can be securely implemented by using HE and is widely studied from theory [BOP06, YY14, GBDL+16] to practice [CdWM+17, BMMP18, AAM17, JKLS18, KSK+18]. On the other hand, PPML training requires accumulating and aggregating data from various users, and thus privacy should be preserved in a multi-party setting, securely against not only the computing server, but also other participating parties.

In scenarios like Federated Learning (FL), multiple parties participate in the model training locally with their own data, and iteratively by a fraction of the whole parties. That is, the training process is divided into several phases in which a subset of users is invited to contribute their local data. To this end, users should be able to dynamically join the training, while the data from each user should not be revealed to the server or other users, making HE in the multi-party setting desirable.

Another realistic requirement for HE in the multi-party setting can be found in the privacy-preserving Digital Asset Management (DAM) scenario [LAHH22]. This scenario is similar to the concept of a *Data Lake* [WA15, CSN+14, TSRC15, QHV16], where big data are accumulated from different resources. The multiple calls of data for different purposes should not require the parties to re-encrypt and send data to the server multiple times, but only once. It is natural to extend this data-centric computing scenario to the multi-party setting, where the users store their encrypted data on the server under their own secret keys.

Currently, there are two primary approaches in the multi-party HE setting: Multi-Key HE (MKHE) [LTV12, BP16, CCS19, KLSW24] and Multi-Party HE (MPHE) [AJL+12, MTBH21, MBH23, Par21]. MPHE has a rigid user structure that must be pre-determined. Thus, most MPHEs are efficient in a static setting, but lack *dynamicity* and *ciphertext reusability*, which we define as follows:

- Dynamicity: The ability to add new parties into an active computation group without re-executing the setup phase or restarting the entire homomorphic process.
- Ciphertext Reusability: The ability to utilize individually encrypted data across different collaborative sessions. For instance, a ciphertext $\mathsf{ct}$ of a party $A$ can be switched for joint computation among $\{A, B\}$ and later the $\mathsf{ct}$ is reused for another set $\{A, C, D\}$, without requiring the data owner to perform redundant re-encryptions.

On the other hand, MKHE provides these features, but is inefficient, as its complexity grows linearly to quadratically, with the number of parties $n$. Although recent work [CDL+21, LZC+24] introduced some flexibility, these schemes often require all existing data providers to remain online during the join process, which is infeasible in realistic network environments.

In this work, we address these challenges by focusing on dynamicity in MPHE. We first propose Dynamic MPHE (dMPHE), which allows new parties to join a computation group without requiring the original parties to be online. Building upon this, we introduce Reusable Dynamic MPHE (rdMPHE), a framework that bridges the gap between MKHE and MPHE by enabling both dynamicity and ciphertext reusability. Beyond merely bridging MKHE and MPHE, our framework can also be understood as a more flexible form of distributed key management in practical HE-based multi-party scenarios: data encrypted under individual secrets can be later reused under a common secret via key switching, while new parties can join ongoing computations through key updates without requiring existing participants to be online.

**Real-World Scenarios and Applications.** To address these practical limitations, we consider a data-centric computing framework that supports both dynamicity and reusability. Our proposed rdMPHE framework, illustrated in Figure 1, is designed to enable the following scenarios: (1) data owners encrypt data under their own keys and store

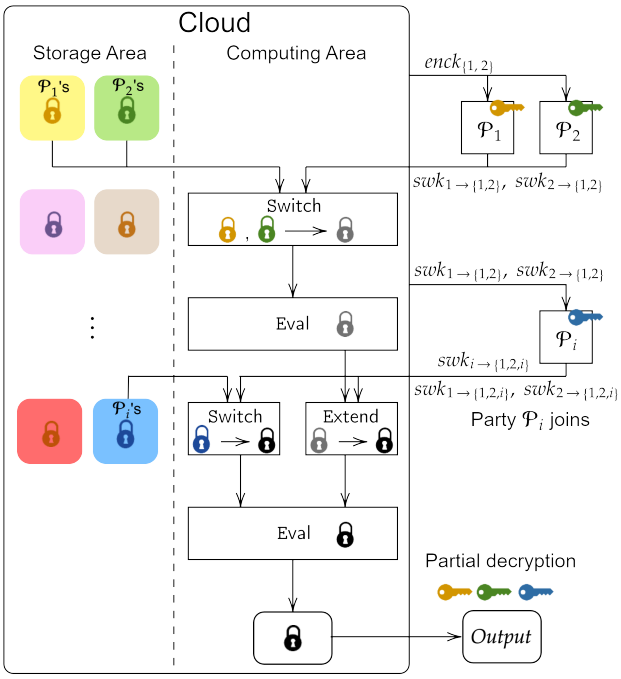



Figure 1: Real-world scenario using rdMPHE.

them in a cloud; (2) when a set of parties agree on a computation, they provide switching keys to switch individual ciphertexts to a common HE ciphertext; (3) homomorphic evaluation is performed under the joint key; (4) results are obtained via joint decryption.

This scenario is particularly beneficial for Privacy-Preserving Machine Learning, where a server trains a model over encrypted data stored in the cloud. New data contributed by additional parties can be seamlessly incorporated to improve the model without affecting performance. Similarly, in a Privacy-Preserving Data Lake, each data owner independently encrypts raw data (such as logs or images) and uploads it. The Data Lake then provides this encrypted data for diverse computations without requiring re-encryption or coordination with all existing providers.[1]

## 1.1 Our Contributions

In this work, we bridge the gap between MKHE and MPHE by introducing dMPHE for dynamicity and rdMPHE for reusability. Our main contributions are as follows:

- We construct an efficient dMPHE scheme in Section 3 with error analysis and complete security proof. Our construction utilizes a "Plus 1" trick to allow the joining parties to operate independently, ensuring that the original parties can remain offline during the extension phase.
- We define rdMPHE as a new primitive, providing formal correctness and game-based semantic security definitions in Section 4. These definitions specifically account for the complexities of dynamic joins and ciphertext reusability without re-encryption, offering a rigorous framework for data-centric applications. We also construct an efficient rdMPHE scheme upon the dMPHE scheme, providing error analysis and a complete security proof.

[1]We refer to Section 4.1.1 for further details.

- We implement the dMPHE and rdMPHE schemes based on CKKS [CKKS17] and B/FV [Bra12, FV12] in `Go` and provide extensive benchmarks in Section 5. We compare our schemes with the implementations of MPHE and MKHE in previous works, demonstrating that our rdMPHE successfully maintains the efficiency of MPHE evaluation $O(1)$ while offering the dynamicity and reusability previously only available in the more costly MKHE frameworks. In addition, we analyze the noise behavior in a practical application.

**Technical Overview.** The fundamental difficulty in simultaneously achieving dynamicity and reusability arises from the intrinsic trade-off between the efficiency of MPHE and the flexibility of MKHE. Although RLWE-based HE inherently supports both addition and multiplication, extending these capabilities to a multi-party setting typically leads either to a rigid, non-adaptive structure (as in MPHE) or to ciphertext sizes that grow linearly (as in MKHE). In addition, a hybrid method called Multi-Group Homomorphic Encryption (MGHE) has been introduced, which combines MPHE and MKHE to exploit the trade-offs between these two models [KLSW24]. It does so by partitioning users into multiple groups and employing MPHE within each group. Nonetheless, this construction does not offer a fundamental resolution to the underlying issue.

In contrast, our framework overcomes this shortcoming by enabling a composable and bidirectional conversions, thereby extending the earlier work of Park [Par21]. In [Par21], the proposed conversions were either one-way or resulted in ciphertexts with non-standard formats. By comparison, our dMPHE construction is explicitly designed to be fully composable. It can be viewed as an efficient mechanism for transforming:

$$\begin{aligned} \mathrm{MPHE}_{n} &\to \mathrm{MGHE}_{n,k_1} &&\to \mathrm{MPHE}_{n+k_1} \\ &\to \mathrm{MGHE}_{n+k_1,k_2} &&\to \mathrm{MPHE}_{n+k_1+k_2} \\ &\to \quad \dots &&\to \mathrm{MPHE}_{n+k_1+\cdots+k_\ell} \ , \end{aligned}$$

for some integer $n, k_1, \dots, k_\ell > 0$, where the subscript denotes the number of parties for MPHE and the number of parties in each group for MGHE. This ensures that the output of an extension process is a valid MPHE ciphertext that can serve as the input for further evaluations or additional joins, allowing the system to scale indefinitely.

To achieve this without requiring the original $n$ parties to remain online, we utilize the *plus 1 trick* (using an auxiliary modulus $P$), which transforms a ciphertext encrypting zero by adding 1 to its $\mathbf{a}$-component, thereby obtaining an encryption of the secret vector $\mathbf{s}$. Our goal is to generate a switching key $\mathsf{swk}_{[n]\to[n+k]}$ that encrypts the original secret key $(\mathbf{s}_{[n]}, 1)$ under the new joint secret key $(\mathbf{s}_{[n+k]}, 1)$.

1. We start by encrypting 0 with the common public key $\mathsf{pk}_{[n]}$ in a higher modulus $PQ$ to get $(\mathbf{a}, \mathbf{b})$.
2. By adding $P$ to the first component, we obtain $(\mathbf{a}' = \mathbf{a} + P, \mathbf{b})$, which effectively encrypts $P \cdot \mathbf{s}_{[n]}$ under $\mathbf{s}_{[n]}$.
3. The new joining parties (either individually or as a group) then add their RLWE samples $\mathbf{a}'\mathbf{s}_j + \mathbf{e}_j$ to the second component $\mathbf{b}$. This results in a switching key that satisfies $\mathbf{a}'\mathbf{s}_{[n+k]} + \mathbf{b}' \approx P \cdot \mathbf{s}_{[n]}$.

Crucially, although the original parties are offline, their authority is preserved because the *honestly generated* switching key only key-switches the ciphertexts under a secret key $(\mathbf{s}_{[n]}, 1)$ to a secret key $(\mathbf{s}_{[n+k]}, 1)$, so that it is decryptable only when the original parties secret keys are involved in the decryption oracle.

While our dMPHE construction provides dynamicity, our rdMPHE construction extends this to support ciphertext reusability across different computation sessions. A major

technical hurdle is updating the existing switch keys $\mathsf{swk}_{i\to[n]}$ to $\mathsf{swk}_{i\to[n+k]}$ when a new group evolves. A naive "double key-switching" would require an additional auxiliary modulus (e.g., modulus $R$ for $\mathcal{R}_{PQR}$), wasting the available modulus margin.

Instead, we employ the *key-switching technique with* $\{\mathsf{sk}_j\}_{[n+k]\setminus[n]}$, the secret keys of newly joined parties, where $\mathsf{sk}_j = (\mathbf{s}_j, 1)$. We update the switching keys by directly adding RLWE samples with respect to each secret key of $\{n+1, \cdots, n+k\}$. This approach avoids the need for a third modulus, maximizing parameter efficiency while allowing data owners to encrypt their data once and reuse it as the computation domain grows dynamically, while its security is guaranteed by the (multi-secret) RLWE assumption.

**Correctness and Semantic Security.** Every extension and switching operation inherently introduces a small amount of noise. We mitigate this through a formal error analysis. As detailed in Sections 3.2 and 4.2, we ensure that the newly introduced noise remains minor, which is experimentally validated in Section 5. In addition, we formally analyze the security of the schemes in Sections 3.3 and 4.3, formally bounding the impact of the extension keys and the switching keys.

**Applicability to DM/CGGI.** While our technique focuses on RLWE-based schemes such as CKKS, B/FV, and BGV [BGV12], one might consider its applicability to DM/CGGI-type FHE schemes [DM15, CGGI17]. However, these schemes typically require gate bootstrapping keys that encrypt each bit of the secret key, making a direct and efficient extension a non-trivial challenge; we leave such adaptation as an interesting direction for future work.

**Code Availability.** Our dMPHE and rdMPHE implementation is publicly available at https://github.com/YongdongYEO/rdMPHE.

## 1.2 Related Works

The landscape of HE in the multi-party setting has evolved through various primitives, each addressing different aspects of efficiency and flexibility. In this subsection, we review these primitives and analyze the technical gaps that motivate our work.

### 1.2.1 Multi-party HE Primitives

**Multi-Key Homomorphic Encryption (MKHE).** The concept of homomorphic encryption for multiple users was first realized by Lopez-Alt et al. [LTV12] through an NTRU-based MKHE scheme, which was later extended to various GSW-based constructions. To improve practical efficiency, Brakerski and Perlman [BP16] proposed an LWE-based MKHE with significantly shorter ciphertexts, and Chen et al. [CCS19] introduced a multi-key variant of CGGI scheme for faster gate-wise operations. The primary advantage of MKHE lies in its unrestricted flexibility, as it allows parties to independently encrypt data and join or leave a computation without any prior coordination.

**Multi-Party Homomorphic Encryption (MPHE).** In parallel, MPHE was developed to achieve higher efficiency in collaborative settings. Following the first BGV-based MPHE by Asharov et al. [AJL+12], Mouchet et al. [MTBH21] provided a more practical framework based on the B/FV scheme, which was recently enhanced with a $t$-out-of-$n$ threshold access structure [MBH23]. Further optimizations by Kwak et al. [KLSW24] enabled single-round relinearization and communication-free key generation, making MPHE a preferred choice for large-scale fixed-party computations. However, despite its efficiency, MPHE typically operates under a static participant model, posing challenges for applications requiring

dynamic party management. On the other hand, Park et al. [PR23] introduced the notion of a homomorphic indicator, which enables the construction of a blind rotation procedure with respect to the aggregated secret (i.e., the sum of individual secret shares). Leveraging this technique, they developed a multiparty variant of the CGGI bootstrapping scheme.

**Hybrid and Dynamic Extensions.** To bridge the gap between the flexibility of MKHE and the efficiency of MPHE, several hybrid approaches have emerged. Aloufi et al. [AHWC21] and Park [Par21] proposed structures where MPHE is employed within sub-groups and MKHE is used between them, a notion later formalized as Multi-Group Homomorphic Encryption (MGHE) by Kwak et al. [KLSW24]. More recently, works by Chen et al. [CDL+21] and Liu et al. [LZC+24] have attempted to introduce dynamicity directly into MPHE, exploring ways to allow new parties to join existing sessions under specific constraints. As we discuss in the following section, however, these extensions often introduce significant communication overhead or require strict online availability of all participants.

### 1.2.2 Limitations and Motivations

Despite these advancements, existing schemes face significant trade-offs in terms of compactness, dynamicity, and reusability, as summarized in Table 1.

Table 1: Comparison of the existing RLWE-based HE in the multi-party setting: MPHE [MTBH21, MBH23], MKHE [KKL+23], MGHE [KLSW24], Liu et al. [LZC+24], and our proposed rdMPHE. The ciphertext reusability, dynamicity, ciphertext and public key compactness, and security against semi-honest adversaries are shown.

| | Reusability | Dynamicity | Ctxt Cpt. | Public Key Cpt. |
|---|---|---|---|---|
| MPHE [MTBH21, MBH23] | ✗ | ✗ | ✓ | ✓ |
| MKHE [KKL+23] | ✓ | ✓ | ✗ | ✓ |
| MGHE [KLSW24] | ▲ (limited) | ✓ | ✗ | ✓ |
| Liu et al. [LZC+24] | ✓ | ▲ (bounded) | ✓ | ✗ |
| dMPHE (ours) | ✗ | ✓ | ✓ | ✓ |
| rdMPHE (ours) | ✓ | ✓ | ✓ | ✓ |

**Lack of Compactness in MKHE and MGHE.** In MKHE construction of [KKL+23], the ciphertext size and computational cost grow linearly with the number of participating parties, making it non-compact. Similarly, MGHE [KLSW24] suffers from linear growth relative to the number of groups. Furthermore, if the composition of MPHE subgroups changes, previously encrypted ciphertexts cannot be reused, limiting its flexibility in dynamic environments.

**Static Nature and Online Constraints of MPHE.** Conventional MPHE constructions [MBH23, KLSW24] require a fixed set of parties during key generation, preventing new parties from joining post-computation. While simple extensions could allow joining by interacting with original parties, this demands that all parties remain online, a prohibitive requirement for large-scale applications like Federated Learning. In contrast, our dMPHE construction allows new parties to join independently (via the "Plus-1" trick[2]) without requiring the original parties to be online, while still preserving their authority as their participation remains mandatory for the final decryption. Liu et al. [LZC+24] attempted

[2]Transforming an RLWE sample $\mathsf{ct} = (\mathbf{a}, \mathbf{b})$ into $\mathsf{ct}' = (\mathbf{a}+1, \mathbf{b})$, while adding appropriate randomness, so that for $\mathsf{sk} = (\mathbf{s}, 1)$, $\langle \mathsf{ct}', \mathsf{sk} \rangle = \langle \mathsf{ct}, \mathsf{sk} \rangle + \mathbf{s}$ is achieved.

to solve this by pre-sharing $\mathcal{O}(|I|)$ common reference strings (CRS), but this introduces significant storage overhead and lacks experimental validation.

**Limited Flexibility in Hybrids.** Schemes by Park et al. [Par21] and Chen et al. [CDL+21] support converting MPHE to MKHE for flexible participation. However, this conversion is one-way; once converted, the system permanently loses the efficiency of MPHE and inherits the non-compactness of MKHE. Moreover, these procedures require all parties to be online simultaneously to perform the conversion.

**Efficiency of Data Reusability.** Existing approaches to data reusability often rely on transciphering, which requires homomorphically evaluating complex decryption circuits. This is computationally expensive, often exceeding 150 seconds even on optimized systems [NHY+25]. Our rdMPHE framework addresses these gaps by enabling dynamic joining and ciphertext reusability through efficient key-switching and additions, maintaining both compactness and practical performance.

# 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 (secret) polynomials $\{\mathbf{s}_i\}_{i\in I}$, we denote by $\mathbf{s}_I$ the sum $\sum_{i\in I}\mathbf{s}_i$. All logarithms are taken to the base 2.

Throughout the paper, we denote the secret key as $\mathsf{sk} = (\mathbf{s}, 1)$, where $\mathbf{s}$ is the secret vector or polynomial. The secret and error distributions are denoted as $\chi_{\mathsf{sk}}$ (for the component $\mathbf{s}$) and $\chi_{\mathsf{err}}$, 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 widely used case.

We use $n$ to denote the number of parties involved in a computation, unless additional parties join at some later stage. Each party is labeled by $\mathcal{P}_i$ for $i \in [n]$, and any index set $I$ is assumed to satisfy $I \subset [n]$.

## 2.2 Assumptions

**Ring Learning with Errors.** Let $\chi$ 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 $\mathrm{DG}_\sigma$ be a discrete gaussian distribution of variance $\sigma$ over $\mathcal{R}_Q$ and $\mathrm{DG}_\sigma$ samples $N$ polynomial coefficients independently. Throughout this paper, we use $\mathrm{DG}_\sigma$ for the error distribution $\chi_{\mathrm{err}}$. The Ring Learning with Errors (RLWE) assumption associated with the parameter $(N, Q, \chi_{\mathsf{sk}}, \chi_{\mathrm{err}})$ is that for given polynomially many samples of either $(\mathbf{a}, \mathbf{b})$ or $(\mathbf{a}, -\mathbf{as}+\mathbf{e})$ are computationally indistinguishable, where $\mathbf{a}, \mathbf{b} \leftarrow \mathcal{R}_Q$, $\mathbf{s} \leftarrow \chi_{\mathsf{sk}}$, and $\mathbf{e} \leftarrow \chi_{\mathrm{err}}$. The multi-secret variant (MS-RLWE) extends the standard assumption to $k > 1$ independent secrets $\mathbf{s}_i \leftarrow \chi_{\mathrm{sk}}$ that share a common $\mathbf{a}$-component. An MS-RLWE sample is defined as a $(k+1)$-tuple $(\mathbf{a}, \mathbf{b}_1, \ldots, \mathbf{b}_k) \in \mathcal{R}_Q^{k+1}$, where $\mathbf{b}_i = -\mathbf{as}_i + \mathbf{e}_i$ with $\mathbf{e}_i \leftarrow \chi_{\mathrm{err}}$. Under this assumption, such a tuple is computationally indistinguishable from a uniform distribution over $\mathcal{R}_Q^{k+1}$.

**Circular Security Assumption.** An encryption scheme is said to be circular secure if it remains semantically secure even when the adversary is given an encrypted cycle of (partial) secret keys (e.g., $\mathtt{Enc}_{\mathsf{sk}}(\mathsf{sk})$ or $(\mathtt{Enc}_{\mathsf{sk}_1}(\mathsf{sk}_2), \mathtt{Enc}_{\mathsf{sk}_2}(\mathsf{sk}_1))$). In the context of our RLWE-based MPHE, we assume that the distribution of $(\mathbf{a}, -\mathbf{as} + \mathbf{e} + P \cdot \mathbf{s}) \in R_{PQ}^2$, an

RLWE sample added with the secret vector multiplied by a constant, is computationally indistinguishable from the uniform distribution over $R_{PQ}^2$. This assumption can be extended to the key-switching keys using the gadget decompositions introduced below.

## 2.3 Key Switching in HE.

We briefly introduce the key switching technique in the RLWE-based HE schemes, which will be heavily utilized in our constructions.

Key switching is a homomorphic operation that transforms a ciphertext encrypting a plaintext $\mathbf{m}$ under a secret key into a ciphertext encrypting the same plaintext but under a different secret key. To key-switch a ciphertext $\mathsf{ct} = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$, without decrypting it, one requires a public key-switching key $\mathsf{swk} = (\boldsymbol{\alpha}, \boldsymbol{\beta}) \in \mathcal{R}_{PQ}^2$. 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 $\boldsymbol{\alpha} \in \mathcal{R}_Q$, we let $\boldsymbol{\beta} + \boldsymbol{\alpha}\mathbf{s}' = P \cdot \mathbf{s} + \boldsymbol{\epsilon} \in \mathcal{R}_{PQ}$, for some error $\boldsymbol{\epsilon} \in \mathcal{R}$. A ciphertext $\mathsf{ct} = (\mathbf{a}, \mathbf{b})$ also has a similar structure, having $\mathbf{b} + \mathbf{as} = \Delta \cdot \mathbf{m} + \mathbf{e} \in \mathcal{R}_Q$ for some error $\mathbf{e} \in \mathcal{R}$. The key-switching procedure outputs a ciphertext $\mathsf{ct}' = (\mathbf{a}', \mathbf{b}') := (0, \mathbf{b}) + \left\lfloor P^{-1} \cdot \mathbf{a} \cdot (\boldsymbol{\alpha}, \boldsymbol{\beta}) \right\rceil \in \mathcal{R}_Q^2$, which satisfies

$$\begin{aligned}\mathbf{b}' + \mathbf{a}'\mathbf{s}' &= \mathbf{b} + \left\lfloor P^{-1} \cdot \mathbf{a}\beta \right\rceil + \left\lfloor P^{-1} \cdot \mathbf{a}\alpha \right\rceil \mathbf{s}' \\ &= \mathbf{b} + P^{-1} \cdot \mathbf{a} \cdot (\beta + \alpha\mathbf{s}') + \mathbf{e}_{\mathrm{rnd}} \\ &= \mathbf{b} + \mathbf{as} + P^{-1} \cdot \mathbf{a}\boldsymbol{\epsilon} + \mathbf{e}_{\mathrm{rnd}} = \Delta \cdot \mathbf{m} + \mathbf{e}',\end{aligned}$$

where $\mathbf{e}' = \mathbf{e} + P^{-1} \cdot \mathbf{a}\boldsymbol{\epsilon} + \mathbf{e}_{\mathrm{rnd}}$ for some rounding error $\mathbf{e}_{\mathrm{rnd}} = \mathbf{e}_\alpha + \mathbf{e}_\beta \mathbf{s}'$ and $\mathbf{e}_\alpha, \mathbf{e}_\beta$ having coefficients in $[-0.5, 0.5]$. If the Hamming weight of $\mathbf{s}'$ is set sufficiently small and $P$ is set sufficiently large, then we have a small bound for error $\mathbf{e}'$, and thus $\mathsf{ct}'$ becomes a ciphertext that encrypts 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 above techniques can be extended to the case when using the gadget decompositions, which dramatically decreases $P$ and increases the possible maximum $Q$,[3] allowing a deeper depth of homomorphic computations. The high-level idea is to encrypt $P \cdot G_j \cdot \mathbf{s}'$ instead of $P \cdot \mathbf{s}'$ for some gadget vector $\vec{G} = (G_1, \cdots, G_d)$, which satisfies $\langle \vec{G}, h(\mathbf{a}) \rangle = \mathbf{a}$ for any $\mathbf{a} \in \mathcal{R}_Q$, for some function $h : \mathcal{R}_Q \to \mathcal{R}_Q^d$.

## 2.4 Homomorphic Encryption in the Multi-Party Setting

Below, we recap the formalisms of HE and its multi-party variants, namely the MPHE and the MKHE, following the terminology defined in [KLSW24]. However, we also note that the MPHE schemes are sometimes referred to as *Compact MKHE* due to their compact ciphertext sizes.

**Multi-Key Homomorphic Encryption.** We begin with the definitions for MKHE.

**Definition 1** (Multi-Key Homomorphic Encryption [LTV12])**.** A Multi-Key Homomorphic Encryption $\mathsf{MKHE}$ is a tuple of efficient algorithms $(\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Eval}, \mathtt{Dec})$ run by multiple parties $\{\mathcal{P}_i\}_{i \in I}$ with the following specifications:

- $\underline{\mathtt{KeyGen}(1^\lambda)}$: for inputs a security parameter $1^\lambda$, run by $\mathcal{P}_i$, outputs a secret key $\mathsf{sk}_i$ and a public key $\mathsf{pk}_i$.

[3] Depending on the choice of $\lambda$, $N$, $\chi_{\mathsf{sk}}$, and $\chi_{\mathrm{err}}$, the maximum possible $PQ$ can be estimated via Lattice-estimator [APS15].

- $\underline{\mathtt{Enc}_{\mathsf{pk}_I}(m)}$: for inputs a set of public keys $\mathsf{pk}_I = \{\mathsf{pk}_i\}_{i\in I}$ and a plaintext $m \in \mathcal{M}$, and outputs a ciphertext $\mathsf{ct}_I$.
- $\underline{\mathtt{Eval}_{\mathsf{pk}_I}(C, (\mathsf{ct}_I^1, \cdots, \mathsf{ct}_I^k))}$: for inputs a set of public keys $\mathsf{pk}_I = \{\mathsf{pk}_i\}_{i\in I}$, a circuit $C : \mathcal{M}^k \to \mathcal{M}$ for some integer $k > 0$, and a tuple of ciphertexts $\mathsf{ct}_I^1, \ldots, \mathsf{ct}_I^k$, it outputs a ciphertext $\mathsf{ct}_I$.
- $\underline{\mathtt{Dec}_{\{\mathsf{sk}_i\}_{i\in I}}(\mathsf{ct}_I)}$: for inputs a set of secret keys $\{\mathsf{sk}_i\}_{i\in I}$ and a ciphertext $\mathsf{ct}_I$, and outputs a plaintext $m$.

One of the known properties of HE is the *composability*, which means that the unit homomorphic operations can be composed into a homomorphic circuit and guaranty its correctness up to a bound related to the circuit and the inputs. We refer to [Mic25] for a more formal treatment of composability, which is not the main target of this paper.

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 $\mu_i$, and are combined by any (possibly third-party) receiver.

**Multi-Party Homomorphic Encryption.** MPHE extends the MKHE framework by introducing a key aggregation algorithm. The primary distinction is that MPHE transforms a set of individual public keys into a single, compact common public key.

**Definition 2** (Multi-Party Homomorphic Encryption)**.** A Multi-Party Homomorphic Encryption MPHE is a tuple of efficient algorithms $(\mathtt{KeyGen}, \mathtt{CombKey}, \mathtt{Enc}, \mathtt{Eval}, \mathtt{Dec})$ run by multiple parties $\{\mathcal{P}_i\}_{i\in I}$ as follows:

- $\underline{\mathtt{KeyGen}(1^\lambda)}$: Run by each party $\mathcal{P}_i$, it outputs a secret key $\mathsf{sk}_i$ and a public key $\mathsf{pk}_i$.
- $\underline{\mathtt{CombKey}(\{\mathsf{pk}_i\}_{i\in I})}$: Taking a set of public keys $\{\mathsf{pk}_i\}_{i\in I}$ as input, it outputs a common public key $\mathsf{pk}_I$.
- $\underline{\mathtt{Enc}_{\mathsf{pk}_I}(m)}$: Taking the common public key $\mathsf{pk}_I$ and a plaintext $m \in \mathcal{M}$ as input, it outputs a ciphertext $\mathsf{ct}_I$ whose size is independent of $|I|$.
- $\underline{\mathtt{Eval}_{\mathsf{pk}_I}(C, (\mathsf{ct}_I^1, \ldots, \mathsf{ct}_I^k))}$: Taking the common public key $\mathsf{pk}_I$, a circuit $C$, and a tuple of ciphertexts encrypted under $\mathsf{pk}_I$, it outputs an evaluated ciphertext $\mathsf{ct}_I$.
- $\underline{\mathtt{Dec}_{\{\mathsf{sk}_i\}_{i\in I}}(\mathsf{ct}_I)}$: A distributed protocol where parties $\{\mathcal{P}_i\}_{i\in I}$ use their secret keys $\mathsf{sk}_i$ to collectively decrypt $\mathsf{ct}_I$ and output $m$.

MPHE achieves compactness by ensuring that $\mathsf{ct}_I$ has the same structure and size as a single-key HE ciphertext. The Dec algorithm is typically implemented as a multi-round protocol consisting of partial decryptions and a final aggregation step. We also note that MKHE can be viewed as a special case of MPHE when CombKey is an identity, i.e., $\mathsf{pk}_I = \{\mathsf{pk}_i\}_{i\in I}$.

The notions of correctness and semantic security are formally defined as follows.

**Definition 3** (Correctness of MPHE, adapted from [KLSW24])**.** Let $\mathsf{pk}_I \leftarrow \mathtt{CombKey}(\{\mathsf{pk}_i\}_{i\in I})$ be the common public key generated by a set of parties $\{\mathcal{P}_i\}_{i\in I}$. For $j \in \{1, \ldots, k\}$, let $\mathsf{ct}_{I,j} \leftarrow \mathtt{Enc}_{\mathsf{pk}_I}(m_j)$ be an encryption of a message $m_j \in \mathcal{M}$. The MPHE scheme is said to be correct if for any circuit $C : \mathcal{M}^k \to \mathcal{M}$ whose depth is bounded by the scheme's parameters, the following holds with overwhelming probability in the security parameter $\lambda$:

$$\mathtt{Dec}\left(\{\mathsf{sk}_i\}_{i\in I}; \mathtt{Eval}_{\mathsf{pk}_I}(C, (\mathsf{ct}_{I,1}, \ldots, \mathsf{ct}_{I,k}))\right) = C(m_1, \ldots, m_k) \ .$$

**Definition 4** (Semantic Security of MPHE in the Semi-honest Model, adapted from [KLSW24])**.** Let $I = \{\mathcal{P}_i\}_{1\le i\le n}$ be a set of parties and let $A \subsetneq I$ be an index set of adversarial parties and $H = I \setminus A$. An MPHE scheme $\Pi = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval}, \mathtt{CombKey})$ is said to be secure if the advantage of $\mathcal{A}$ in the following game is negligible in the security parameter $\lambda$, for any PPT adversary $\mathcal{A}$:

- The challenger executes $\mathtt{KeyGen}(1^\lambda)$ for each $\mathcal{P}_i$, for $1 \le i \le n$, and sends the public keys $\mathsf{pk}_1, \cdots, \mathsf{pk}_n$ and secret keys $\{\mathsf{sk}_i : i \in A\}$ to the adversary.
- The adversary chooses messages $m_0, m_1 \in \mathcal{M}$ and send it to the challenger.
- The challenger samples a random bit $b \in \{0, 1\}$ and sends $\mathtt{Enc}_{\mathsf{pk}_I}(m_b)$ back to the adversary.
- The adversary $\mathcal{A}$ outputs a bit $b'$. The advantage is defined as $|\Pr[b = b'] - \frac{1}{2}|$.

**Dynamic Multi-Party Homomorphic Encryption.** Below, we recap the Dynamic MPHE (dMPHE) with additional properties, which originates from the Dynamic MKHE (dMKHE) of [CDL+21]. At a high level, the dynamicity allows new parties to join during homomorphic computation.

**Definition 5** (Dynamic Multi-Party Homomorphic Encryption [CDL+21])**.** A Dynamic Multi-Party Homomorphic Encryption $\mathsf{dMPHE}$ is an MPHE additionally with efficient algorithms $\mathtt{ExtKeyGen}$ and $\mathtt{Extend}$ with the following specifications:

- $\underline{\mathtt{ExtKeyGen}_{I\to I'}(\{\mathsf{pk}_i\}_{i\in I'}, \{\mathsf{sk}_i\}_{i\in I'})}$: run by $\{\mathcal{P}_i\}_{i\in I'}$, takes as inputs two index sets $I \subsetneq I'$, a set of public keys $\{\mathsf{pk}_i\}_{i\in I'}$, and a set of secret keys $\{\mathsf{sk}_i\}_{i\in I'}$, it outputs a public extension key $\mathsf{extk}_{I\to I'}$.
- $\underline{\mathtt{Extend}(\mathsf{extk}_{I\to I'}, \{\mathsf{pk}_i\}_{i\in I'}, \mathsf{ct}_I)}$: for given inputs a public extension key $\mathsf{extk}_{I\to I'}$, a set of public keys $\{\mathsf{pk}_i\}_{i\in I'}$, and a ciphertext $\mathsf{ct}_I$ encrypted under $\{\mathsf{sk}_i\}_{i\in I}$, it outputs a ciphertext $\mathsf{ct}_{I'}$ encrypted under $\{\mathsf{sk}_i\}_{i\in I'}$.

We then give the formal correctness and security definitions of dMPHE below.

**Definition 6** (Correctness of dMPHE, adapted from [CDL+21])**.** Let $I_1 \subsetneq I_2 \subsetneq \cdots \subsetneq I_k$ be a sequence of sets of parties and $\mathsf{pk}_{I_\ell} \leftarrow \mathtt{CombKey}(\{\mathsf{pk}_i\}_{i\in I_\ell})$ be the common public key for each set $I_\ell$. For any $1 \le \ell \le k$ and $j \in \{1, \ldots, m\}$, let $\mathsf{ct}_{I_\ell,j}$ be a ciphertext encrypting $m_j$ under the joint key $\mathsf{sk}_{I_\ell}$, where $\mathsf{ct}_{I_\ell,j}$ is obtained via one of the following: i) fresh encryption $\mathsf{ct}_{I_\ell,j} \leftarrow \mathtt{Enc}_{\mathsf{pk}_{I_\ell}}(m_j)$, or ii) recursive extension $\mathsf{ct}_{I_\ell,j} \leftarrow \mathtt{Extend}(\mathsf{extk}_{I_{\ell-1}\to I_\ell}, \{\mathsf{pk}_i\}_{i\in I_\ell}, \mathsf{ct}_{I_{\ell-1},j})$ where $\mathsf{ct}_{I_{\ell-1},j}$ is a valid ciphertext under $\mathsf{sk}_{I_{\ell-1}}$. Then, the dMPHE scheme is said to be correct if for any index $1 \le \ell \le k$, any circuit $C : \mathcal{M}^m \to \mathcal{M}$ whose depth is bounded by the scheme's parameters, and any messages $m_1, \ldots, m_m \in \mathcal{M}$, the following holds with overwhelming probability in the security parameter $\lambda$:

$$\mathtt{Dec}\left(\{\mathsf{sk}_i\}_{i\in I_\ell}; \mathtt{Eval}_{\mathsf{pk}_{I_\ell}}(C, (\mathsf{ct}_{I_\ell,1}, \ldots, \mathsf{ct}_{I_\ell,m}))\right) = C(m_1, \ldots, m_m) \ .$$

**Definition 7** (Semantic Security of dMPHE in the Semi-honest Model, adapted from [CDL+21])**.** Let $I_1 \subsetneq I_2 \subsetneq \cdots \subsetneq I_k = \{\mathcal{P}_i\}_{1\le i\le n}$ be a sequence of set of parties. Let $A \subset I_k$ be a set of adversarial parties, and $H = I_k \setminus A$. A dMPHE scheme $\Pi = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval}, \mathtt{CombKey}, \mathtt{ExtKeyGen}, \mathtt{Extend})$ is said to be secure if the advantage of $\mathcal{A}$ in the following game is negligible in the security parameter $\lambda$, for any PPT adversary $\mathcal{A}$:

- The challenger executes $\mathtt{KeyGen}(1^\lambda)$ for each $\mathcal{P}_i$, for $1 \le i \le n$, and sends the public keys $\mathsf{pk}_1, \cdots, \mathsf{pk}_n$ and secret keys $\{\mathsf{sk}_i : i \in A\}$ to the adversary.

- The challenger also executes $\mathtt{ExtKeyGen}_{I_i \to I_{i+1}}(\{\mathsf{pk}_i\}_{i\in I_i}, \{\mathsf{sk}_i\}_{i\in I_{i+1}})$ for all $i = 1, \cdots, k-1$, and sends the extension keys $\mathsf{extk}_{I_i \to I_{i+1}}$ to the adversary.
- The adversary chooses messages $m_0, m_1 \in \mathcal{M}$ and an index $1 \le \ell \le k$, where $I_\ell \not\subset A$, then send them back to the challenger.
- The challenger samples a random bit $b \in \{0, 1\}$ and sends $\mathtt{Enc}_{\mathsf{pk}_{I_\ell}}(m_b)$ back to the adversary.
- The adversary $\mathcal{A}$ outputs a bit $b'$. The advantage is defined as $|\Pr[b = b'] - \frac{1}{2}|$.

# 3 Dynamic MPHE Scheme

In this section, we introduce our dMPHE construction, that serves as a building block for constructing an rdMPHE scheme in the latter section. We first recap the state-of-the-art MPHE scheme in Section 3.1, which becomes an underlying structure of our dMPHE in Section 3.2. We discuss its dynamicity and correctness with error analysis. Lastly, we provide its security proof in Section 3.3.

## 3.1 Base MPHE Scheme

Our base MPHE is identical to the MPHE scheme from [MTBH21], but incorporates a non-interactive relinearization key generation technique from [KLSW24]. Our scheme is based on the CRS model, i.e., all parties have access to the same random string. Our base MPHE scheme can be described as follows.

- $\mathsf{MPHE}.\mathtt{KeyGen}(1^\lambda; \mathtt{crs})$: For given inputs, each party $\mathcal{P}_i$ samples small polynomials for a secret key $\mathsf{sk}_i = (\mathbf{s}_i, 1)$ and an auxiliary secret $\mathbf{r}_i$ from $\chi_{\mathsf{sk}}(\in \mathcal{R})$. A public key $\mathsf{pk}_i$ is also generated, which consists of encryption key $\mathsf{enck}_i$, rotation keys $\mathsf{autk}_{\sigma,i}$ (with respect to each automorphism $\sigma$), and relinearization key $\mathsf{rlk}_i$. Each key is generated based on common polynomials $\mathbf{a}_* \in \mathcal{R}_*$ induced from $\mathtt{crs}$ and errors $\mathbf{e}_*$ sampled from $\chi_{\mathrm{err}}$ as follows.
  - $\mathsf{enck}_i = (\mathbf{a}_{\mathsf{enck}}, \mathbf{b}_{\mathsf{enck},i} = -\mathbf{a}_{\mathsf{enck}}\mathbf{s}_i + \mathbf{e}_{\mathsf{enck},i}) \in \mathcal{R}_Q^2$,
  - $\mathsf{autk}_{\sigma,i} = (\mathbf{a}_\sigma, \mathbf{b}_{\sigma,i} = -\mathbf{a}_\sigma\mathbf{s}_i + P \cdot \sigma(\mathbf{s}_i) + \mathbf{e}_{\sigma,i}) \in \mathcal{R}_{PQ}^2$,
  - $\mathsf{rlk}_i = (\mathsf{rlk}_{i,0}, \mathsf{rlk}_{i,1}, \mathsf{rlk}_{i,2}) \in \mathcal{R}_{PQ}^3$ for $\mathsf{rlk}_{i,0} = -\mathbf{a}_{\mathsf{rlk},0}\mathbf{s}_i + \mathbf{e}_{\mathsf{rlk},i,0}$, $\mathsf{rlk}_{i,1} = -\mathbf{a}_{\mathsf{rlk},0}\mathbf{r}_i + P \cdot \mathbf{s}_i + \mathbf{e}_{\mathsf{rlk},i,1}$, and $\mathsf{rlk}_{i,2} = -\mathbf{a}_{\mathsf{rlk},1}\mathbf{s}_i - P \cdot \mathbf{r}_i + \mathbf{e}_{\mathsf{rlk},i,2}$.
- $\mathsf{MPHE}.\mathtt{CombKey}(\{\mathsf{pk}_i\}_{i\in I})$: For given inputs, a common public key $\mathsf{pk}_I$ for a set of parties $I$ is generated by summing the partial public keys: It consists of a common encryption key $\mathsf{enck}_I = (\mathbf{a}_{\mathsf{enck}}, \sum_{i\in I} \mathbf{b}_{\mathsf{enck},i}) \in \mathcal{R}_Q^2$, common rotation keys $\mathsf{autk}_{\sigma,I} = (\mathbf{a}_\sigma, \sum_{i\in I} \mathbf{b}_{\sigma,i}) \in \mathcal{R}_{PQ}^2$, and a common relinearization key $\mathsf{rlk}_I = (\mathsf{rlk}_{I,0}, \mathsf{rlk}_{I,1}, \mathsf{rlk}_{I,2}) = \sum_{i\in I} \mathsf{rlk}_i \in \mathcal{R}_{PQ}^3$.
- $\mathsf{MPHE}.\mathtt{Enc}(\mathsf{enck}_I, \mathbf{m})$: For given a message $\mathbf{m} \in \mathcal{R}$ and the common encryption key $\mathsf{enck}_I$, it outputs $\mathsf{ct} = \mathbf{u} \cdot \mathsf{enck}_I + (\mathbf{e}_0, \Delta\mathbf{m} + \mathbf{e}_1) \in \mathcal{R}_Q^2$, where $\mathbf{u} \leftarrow \mathcal{R}$ and $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \chi_{\mathrm{err}}$.
- $\mathsf{MPHE}.\mathtt{Dec}(\{\mathsf{sk}_i\}_{i\in I}, \mathsf{ct})$: For given a ciphertext $\mathsf{ct} = (\mathbf{c}_0, \mathbf{c}_1) \in \mathcal{R}_Q^2$ and secret keys $\mathsf{sk}_i = (\mathbf{s}_i, 1)$ of parties $\mathcal{P}_i$ for $i \in I$, the decryption is done in a distributed way:
  - $\mathtt{PartDec}(\mathsf{sk}_i, \mathsf{ct})$ : Each party $\mathcal{P}_i$ generates partial decryption $\boldsymbol{\mu}_i = \mathbf{c}_0\mathbf{s}_i + \mathbf{e}_i$ of $\mathsf{ct}$, where $\mathbf{e}_i \leftarrow \chi_{\mathrm{smudge}}$.
  - $\mathtt{Aggregate}(\{\boldsymbol{\mu}_i\}_{i\in I}, \mathsf{ct})$ : An aggregator (one among the parties or a third party) aggregates the partial decryptions and outputs $\lfloor (\sum_{i\in I} \boldsymbol{\mu}_i + \mathbf{c}_1)/\Delta \rceil \in \mathcal{R}$.

We note that $\chi_{\text{smudge}}$ be an error distribution for a secure distributed decryption [LMSS22, CSS+22], where the standard deviation is set $\lambda/2$ bit larger than that of the ciphertext error to smudge the error, not to reveal the secret, even when $|I|-1$ parties collude.

For the sake of simplicity, we denote $\mathbf{s}_I = \sum_{i\in I} \mathbf{s}_i$. Also, we sometimes denote the keys as $\mathsf{swk}_{\mathbf{s}\to\mathbf{s}'} = (\mathbf{a}, \mathbf{b} = -\mathbf{a}\mathbf{s}' + P\cdot\mathbf{s} + \mathbf{e}) \in \mathcal{R}_{PQ}^2$ for some small error $\mathbf{e}$ if there is no confusion, e.g., $\mathsf{enck}_i$ can be written as $\mathsf{swk}_{0\to\mathbf{s}_i}$; and $\mathsf{autk}_{\sigma,I}$ can be written as $\mathsf{swk}_{\sigma(\mathbf{s}_I)\to\mathbf{s}_I}$.

For homomorphic evaluation, we recall homomorphic addition, multiplication, and automorphism (such as rotating the message polynomials). For ciphertexts $\mathsf{ct} = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$ and $\mathsf{ct}' = (\mathbf{a}', \mathbf{b}') \in \mathcal{R}_Q^2$ encrypting plaintexts $\mathbf{m}_0$ and $\mathbf{m}_1$ in $\mathcal{R}$ under $\{\mathsf{sk}_i\}_{i\in I}$, homomorphic addition, multiplication, key switching, and automorphisms can be done as follows.

- $\underline{\mathsf{MPHE.Add}(\mathsf{ct}_I, \mathsf{ct}_I')}$: For given inputs, it outputs $\mathsf{ct}_I + \mathsf{ct}_I' \in \mathcal{R}$, encrypting $\mathbf{m}_0 + \mathbf{m}_1$.
- $\underline{\mathsf{MPHE.Mult}_{\mathsf{rlk}_I}(\mathsf{ct}_I, \mathsf{ct}_I')}$: For given two ciphertexts, it computes as follows.
  1. It first tensor and obtain $\mathbf{d} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2) = (\mathbf{b}\mathbf{b}', \mathbf{a}\mathbf{b}' + \mathbf{a}'\mathbf{b}, \mathbf{a}\mathbf{a}') \in \mathcal{R}_Q^3$.
  2. It then relinearizes by first computing compute $\mathbf{d}_2' = \lfloor P^{-1}\cdot(\mathbf{d}_2 \cdot \mathsf{rlk}_0)\rceil \in \mathcal{R}_Q$, then computing
  $$(\mathbf{d}_1', \mathbf{d}_0') = \lfloor P^{-1}\cdot(\mathbf{d}_2' \cdot (\mathbf{a}_{\mathsf{rlk},1}, \mathsf{rlk}_2) + \mathbf{d}_2 \cdot (\mathsf{rlk}_1, 0))\rceil \in \mathcal{R}_Q^2 \ .$$
  3. It outputs $\lfloor \Delta^{-1}\cdot(\mathbf{d}_1' + \mathbf{d}_1, \mathbf{d}_0' + \mathbf{d}_0)\rceil \in \mathcal{R}_{Q/\Delta}^2$, encrypting $\mathbf{m}_0 \cdot \mathbf{m}_1$.
- $\underline{\mathsf{MPHE.KS}(\mathsf{swk}_{\mathbf{s}\to\mathbf{s}'}, \mathsf{ct})}$: For given a ciphertext $\mathsf{ct} = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$, it outputs $(0, \mathbf{b}) + \lfloor P^{-1}\cdot\mathbf{a}\cdot\mathsf{swk}_{\mathbf{s}\to\mathbf{s}'}\rceil \in \mathcal{R}_Q^2$, encrypting $\sigma(\mathbf{m})$.
- $\underline{\mathsf{MPHE.Aut}_{\mathsf{autk}_{\sigma,I}}(\mathsf{ct})}$: For given a ciphertext $\mathsf{ct} = (\mathbf{a}, \mathbf{b}) \in \mathcal{R}_Q^2$, it first computes $\mathsf{ct}' = (\sigma(\mathbf{a}), \sigma(\mathbf{b}))$. It then outputs $\mathsf{MPHE.KS}(\mathsf{swk}_{\sigma(\mathbf{s}_I)\to\mathbf{s}_I} = \mathsf{autk}_{\sigma,I}, \mathsf{ct}') \in \mathcal{R}_Q^2$.

**Correctness and Semantic Security of MPHE Scheme.** For the correctness and semantic security of the above MPHE scheme, we refer to [KLSW24, Sections 4], the correctness and semantic security analysis for their MGHE construction. It applies directly to the above-defined MPHE scheme when the number of groups in MGHE is 1. In a high-level view, the correctness of multiplication holds since

$$\begin{aligned}
\mathbf{d}_2\cdot\mathbf{s}_I^2 &\approx P^{-1}\cdot\mathbf{d}_2\cdot(P\cdot\mathbf{s}_I)\cdot\mathbf{s}_I \approx P^{-1}\cdot\mathbf{d}_2\cdot(\mathbf{a}_{\mathsf{rlk},0}\mathbf{r}_I + \mathsf{rlk}_1)\cdot\mathbf{s}_I \\
&\approx P^{-1}\cdot[\mathbf{d}_2\cdot(\mathbf{a}_{\mathsf{rlk},0}\cdot\mathbf{s}_I)\cdot\mathbf{r}_I + \mathbf{d}_2\cdot\mathsf{rlk}_1\cdot\mathbf{s}_I] \\
&\approx P^{-1}\cdot\left[P^{-1}\cdot\mathbf{d}_2\cdot\mathsf{rlk}_0\cdot(P\mathbf{r}_I) + \mathbf{d}_2\cdot\mathsf{rlk}_1\cdot\mathbf{s}_I\right] \\
&\approx P^{-1}\cdot[\mathbf{d}_2'\cdot(\mathbf{a}_{\mathsf{rlk},1}\cdot\mathbf{s}_I + \mathsf{rlk}_2) + \mathbf{d}_2\cdot\mathsf{rlk}_1\cdot\mathbf{s}_I] \approx \mathbf{d}_1'\mathbf{s}_I + \mathbf{d}_0'.
\end{aligned}$$

For semantic security, we recall the following lemma from [KLSW24].

**Lemma 1** (Semantic Security of the Base MPHE scheme [KLSW24, Lemma 1])**.** *The MPHE scheme described above is semantically secure under the RLWE assumption and the circular security assumption with respect to the given parameters.*

*Proof.* The proof follows the structure of the security analysis in [KLSW24], adapted to our notation where the number of groups is fixed to one ($k = 0$).

The semantic security of the MPHE scheme is proven through a sequence of hybrid games. Let $A \subset I$ be the set of parties corrupted by the adversary, and $H = I \setminus A$ be the set of honest parties, where $H \neq \emptyset$.

- Game 0: This is the real-world security game. The challenger generates all keys according to the $\mathsf{MPHE.KeyGen}$ algorithm.

- Game 1: Identical to Game 0, except that for each honest party $i \in H$, the components of the public key $\mathsf{pk}_i = (\mathsf{enck}_i, \mathsf{autk}_{\sigma,i}, \mathsf{rlk}_i)$ are replaced with uniform random samples from their respective domains ($\mathcal{R}_Q^2$ and $\mathcal{R}_{PQ}^2, \mathcal{R}_{PQ}^3$).
- Game 2: Identical to Game 1, but the challenge ciphertext is an encryption of 0 instead of $m_b$.

**Indistinguishability of Game 0 and Game 1.** For an honest party $i \in H$, the public key components are:

$$\begin{aligned}\mathsf{enck}_i &= (\mathbf{a}_{\mathsf{enck}}, \mathbf{b}_{\mathsf{enck},i} = -\mathbf{a}_{\mathsf{enck}}\mathbf{s}_i + \mathbf{e}_{\mathsf{enck},i}),\\ \mathsf{autk}_{\sigma,i} &= (\mathbf{a}_\sigma, \mathbf{b}_{\sigma,i} = -\mathbf{a}_\sigma \mathbf{s}_i + P \cdot \sigma(\mathbf{s}_i) + \mathbf{e}_{\sigma,i}),\\ \mathsf{rlk}_i &= (\mathsf{rlk}_{i,0}, \mathsf{rlk}_{i,1}, \mathsf{rlk}_{i,2}) \ ,\end{aligned}$$

involving secret $\mathbf{s}_i$ and auxiliary secret $\mathbf{r}_i$. Under the RLWE assumption, the encryption key $\mathsf{enck}_i$ is indistinguishable from a uniform distribution over $\mathcal{R}_{PQ}^2$. For $\mathsf{autk}_{\sigma,i}$ and $\mathsf{rlk}_i$, these can be viewed as circular encryptions (e.g., encryption of $\mathbf{s}_i$ under $\mathbf{r}_i$ and vice versa). By invoking the circular security assumption, these keys are also computationally indistinguishable from uniform random samples. Thus, Game 0 and Game 1 are computationally indistinguishable.

**Indistinguishability of Game 1 and Game 2.** In Game 1, the joint encryption key $\mathsf{enck}_I$ is computed as:

$$\mathsf{enck}_I = (\mathbf{a}_{\mathsf{enck}}, \sum_{i\in A} \mathbf{b}_{\mathsf{enck},i} + \sum_{i\in H} \mathbf{b}_{\mathsf{enck},i}) \ .$$

Since $H$ is non-empty and each $\mathbf{b}_{\mathsf{enck},i}$ for $i \in H$ is sampled uniformly at random in Game 1, the sum $\sum_{i\in I} \mathbf{b}_{\mathsf{enck},i}$ is also uniformly distributed over $\mathcal{R}_Q$. Therefore, the encryption key $\mathsf{enck}_I$ is a uniform random sample in $\mathcal{R}_Q^2$. Under the RLWE assumption, an encryption of any message $m_b$ is indistinguishable from an encryption of 0 (which itself is indistinguishable from a uniform distribution over $\mathcal{R}_Q^2$). Hence, Game 1 and Game 2 are indistinguishable. Since the adversary's advantage in Game 2 is zero, the advantage in Game 0 must be negligible. □

## 3.2 Dynamic MPHE Construction

In this section, we introduce dynamicity into the base MPHE scheme. Unlike [CDL+21], we highlight that our construction allows dynamicity to be achieved without requiring the previous parties in $I$ to remain online during the extension. That is, the original parties can remain *offline* and will only participate in the decryption phase. We prove the game-based semantic security of our dMPHE construction by reducing its security to the base MPHE's semantic security.

Our dMPHE shares the main structures with the algorithms in Section 3.1, but is modified a bit to introduce dynamicity:

- The (common) encryption key is now defined in $\mathcal{R}_{PQ}^2$, instead of $\mathcal{R}_Q^2$. Consequently, the encryptions can be defined in modulus $PQ$, but homomorphic operations are only allowed in modulus below $Q$;
- The two algorithms `ExtKeyGen` and `Extend` are defined as follows. For the index sets $I \subsetneq I'$, we define
  - $\mathsf{dMPHE.ExtKeyGen}(\mathsf{enck}_I, \{\mathsf{sk}_i\}_{i\in I'\setminus I})$: For given encryption key for $I$, it first computes $(\mathbf{a}, \mathbf{b}) = \mathtt{Enc}_{\mathsf{enck}_I}(0) \in \mathcal{R}_{PQ}^2$, then outputs $\mathsf{extk}_{I\to I'} = (\mathbf{a} + P, \mathbf{b}) - \sum_{i\in I'\setminus I}(0, (\mathbf{a}+P)\cdot \mathbf{s}_i + \mathbf{e}_i)$, where $\mathbf{e}_i \leftarrow \chi_{\mathrm{err}}$.

– $\underline{\mathsf{dMPHE.Extend}\ (\mathsf{extk}_{I\to I'}, \mathsf{ct}_I)}$: For given extension key and a ciphertext with respect to $I$, it outputs $\mathsf{MPHE.KS}(\mathsf{extk}_{I\to I'}, \mathsf{ct}_I)$, an extended ciphertext for $I'$.

We note that for the base MPHE construction that employs gadget decomposition (e.g. RNS-based or digit-based decompositions [CHK+19, HPS19]), a subtle modification allows `ExtKeyGen` to function in an analogous manner. Specifically, the subtraction by $P$ is replaced by the subtraction by $PG_j$, where $G_j$ denotes the corresponding gadget vectors.

We also note that our dMPHE construction relies on a relaxed definition for `ExtKeyGen`, which does not need the secret keys of the parties in $I$. That is, all the public keys $\{\mathsf{pk}_i\}_{i\in I'}$ and the secret keys $\{\mathsf{sk}_i\}_{i\in I'\setminus I}$ of only the parties that join are input into the `ExtKeyGen` algorithm. Also, `ExtKeyGen` is done by the challenger in the semi-honest model, as part of the key generation, which is a natural extension of the usual MPHE use-cases, where the whole computation (including the `SetUp` and `KeyGen`) should be re-executed from scratch.

**Joining of New Parties.** Assume a scenario where a new set of parties (of index $I'$) joins the existing parties (of index $I$). To facilitate subsequent computations among the expanded set of parties of index $I'$, the joining parties must first generate the extension key $\mathsf{extk}_{I\to I'}$ via `ExtKeyGen`. Furthermore, the public keys and the ciphertexts must be updated to facilitate subsequent computation among the parties in $I'$. We summarize the required actions by the joining parties and the entity that evaluates the homomorphic computation:

- Extension Key Generation ($\mathsf{extk}_{I\to I'}$, by Joining Parties): The set of joining parties $\{\mathcal{P}_i\}_{i\in I'\setminus I}$ executes the `extkeygen` algorithm. This process takes the public keys of the parties in $I$ and the secret keys of the joining parties as inputs, and outputs the extension key $\mathsf{extk}_{I\to I'}$.
- Public Key Update ($\mathsf{pk}_I \to \mathsf{pk}_{I'}$): The common public key $\mathsf{pk}_I$ (including common encryption, automorphism, and relinearization keys) is updated to $\mathsf{pk}_{I'}$ by applying `CombKey` to the individual public keys of all parties in $I'$. As all inputs are public, this procedure can be performed by any entity (e.g., the evaluator).
- Ciphertext Extension ($\{\mathsf{ct}_{I,j}\} \to \{\mathsf{ct}_{I',j}\}$): Ciphertexts encrypted under the secret key $\mathsf{sk}_I$ (whether freshly encrypted or resulting from homomorphic evaluations by the parties in $I$) must be transitioned to the ciphertexts encrypted under the new secret key $\mathsf{sk}_{I'}$ to enable further computation with the new public keys. This extension is achieved by applying `Extend` to the ciphertexts using the extension key $\mathsf{extk}_{I\to I'}$. Since the extension key is public, this extension can also be performed by any entity, such as the evaluator.

We note that the extension keys and the public keys need to be generated honestly for its correctness and security.

**Dynamicity: Passive Drop-out vs. Proactive Quitting.** We adopt a notion of dynamicity similar to [CDL+21], but we further clarify our scope by distinguishing between *passive dropout* and *proactive quitting*.

First, regarding passive drop-out (where a party unexpectedly goes offline), our construction naturally supports this during the setup and evaluation phases. Since existing parties are not required to participate when new parties join, the temporary absence of a party does not halt the system's progress. However, in a strict $n$-out-of-$n$ threshold setting, there is currently no solution for drop-outs during the *decryption phase*. If any party (in the access structure) denies participation or remains offline, the decryption process cannot be completed. This ensures that no information is leaked without the explicit consent of all stakeholders.

Second, one may consider a proactive quitting scenario where a participant voluntarily leaves and seeks to be removed from the access structure. At first glance, this could be handled by executing `PartDec` on the common public keys and applying an `extend`-like procedure to all relevant ciphertexts to remove the leaving party's secret components. However, this approach introduces several nontrivial security concerns:

- Forced Removal vs. Security: The most compelling application of a drop-out mechanism would be the forced removal of parties who have gone offline; however, such a mechanism is inherently insecure. If it were possible to remove a party without their active participation, the remaining parties could potentially apply `extend`-like procedures to any individual input ciphertext and decrypt it without the owner's consent.
- Trust and Collusion: In a quitting scenario where a party proactively updates ciphertexts via `PartDec` before leaving, they essentially grant the remaining parties full authority over those data. This raises a critical trust issue: the quitter must be certain that at least one honest party remains among the group to prevent collusion and unauthorized access to the now "partially decrypted" ciphertexts.

Therefore, supporting such "quitting" is only feasible if a leaving party can securely delegate its authority to a guaranteed honest party. Consequently, we focus on the *join-only dynamic setting* to maintain a robust security guarantee. This approach ensures that while the system accommodates new participants, the integrity of the original access structure remains intact.

### 3.2.1 Correctness of Dynamic MPHE Scheme.

To prove the correctness of our dMPHE construction, it suffices to show that the extended ciphertext and the original ciphertext are decrypt to the same plaintext, with respect to the set of secret keys $\{\mathsf{sk}_i\}_{i\in I'}$ and $\{\mathsf{sk}_i\}_{i\in I}$, respectively. Once the parties have joined and the public keys and ciphertexts are updated, the resulting public keys exhibit errors of exactly the same form and distribution as those in a base MPHE construction for $I'$ parties. Consequently, all subsequent homomorphic evaluations on the extended ciphertexts can be analyzed as in the base MPHE scheme.

Let $(\mathbf{a}, \mathbf{b}) = \mathtt{Enc}_{\mathsf{enck}_I}(0)$ and the error $\mathbf{e}_i$ is defined as in `ExtKeyGen` algorithm. Let $\mathbf{e}_{\mathsf{enc}} = \mathbf{a}\mathbf{s}_I + \mathbf{b} \in \mathcal{R}_{PQ}$, $\mathbf{a}' = \mathbf{a} + P \in \mathcal{R}_{PQ}$, and $\mathsf{extk}_{I\to I'} = (\mathbf{a}', \mathbf{b}'') \in \mathcal{R}_{PQ}^2$ where $\mathbf{b}'' = \mathbf{b} - \sum_{i\in I'\setminus I}(\mathbf{a}'\mathbf{s}_i + \mathbf{e}_i)$. Then, we have

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

for $\mathbf{e}_{\mathsf{extk}} = \mathbf{e}_{\mathsf{enc}} - \sum_{i\in I'\setminus I}\mathbf{e}_i$. That is, $\mathsf{extk}_{I\to I'}$ is a form of a key-switching key $\mathsf{swk}_{\mathbf{s}_I\to\mathbf{s}_{I'}}$, and thus the key-switching operation concludes the correctness proof unless the aggregated error is too large.

In Lemma 2, we formally analyze the error after ciphertext extension with the extension key `extk`, in the case of CKKS-based dMPHE constructions. The lemma implies that the error introduced by the ciphertext extension (`Extend`) can be managed as small as usual key-switching operations in MPHE with $I'$ number of parties. In addition, the experimental results in Section 5 highlight their practical impact on the error, during homomorphic computation. We note that the lemma can be seamlessly extended to B/FV-based constructions, and further to BGV-based constructions.

**Lemma 2** (Error Analysis for Extended dMPHE Ciphertexts)**.** *Consider the above dMPHE scheme constructed upon a CKKS-based MPHE scheme. Assume that the secret coefficients are sampled from a ternary distribution taking values in* $\{0, \pm 1\}$ *with probabilities* $1/2$ *and* $1/4$ *for* $0$ *and* $\pm 1$*, respectively, and that the error coefficients are sampled from a discrete Gaussian distribution with standard deviation* $\sigma$*. Assume further that the canonical embedding norm of the base MPHE key-switching error is bounded by the sum of the multiplication-induced error*[4] *and the rounding error with overwhelming probability in the security parameter* $\lambda$*, as* $B_{\mathrm{KS}}^{\mathsf{MPHE}} = B_{\mathrm{mul}}^{\mathsf{MPHE}} + B_{\mathrm{rnd}}^{\mathsf{MPHE}}$*, where* $B_{\mathrm{mul}}^{\mathsf{MPHE}} = (8/\sqrt{3}) \cdot P^{-1} q_\ell \sigma N \sqrt{|I'|}$ *and* $B_{\mathrm{rnd}}^{\mathsf{MPHE}} = 3q_\ell \sqrt{N/3} + 8q_\ell N \sqrt{|I'|/6}$*, as analyzed in [CKKS17].*

*Then, for a dMPHE ciphertext extended from* $I$ *to* $I'$*, the extension error is bounded by* $B_{\mathrm{ext}} = B_{\mathrm{mul}} + B_{\mathrm{rnd}}$ *with overwhelming probability in the security parameter* $\lambda$*, where*

$$B_{\mathrm{mul}} = \sqrt{(|I'|N + |I' \setminus I| + 1)/|I'|} \cdot B_{\mathrm{mul}}^{\mathsf{MPHE}} \quad \text{and} \quad B_{\mathrm{rnd}} = B_{\mathrm{rnd}}^{\mathsf{MPHE}} \ .$$

*Proof.* We adopt a heuristic approach as also utilized in [GHS12,CS16]. For high-probability bounds, we use $6\sigma$ for a variable with variance $\sigma^2$, and $16\sigma_1\sigma_2$ for the product of two Gaussian variables with variances $\sigma_1^2$ and $\sigma_2^2$. All error bounds are measured with respect to the canonical norm in the CKKS message space [CKKS17].

Let $\mathsf{ct} = (\mathbf{a}, \mathbf{b})$ be a ciphertext with respect to the party set $I$, and consider when it is key-switched with $\mathsf{extk}_{I \to I'}$. Recall that $(0, \mathbf{b}) + \left\lfloor P^{-1} \cdot \mathbf{a} \cdot \mathsf{extk}_{I \to I'} \right\rceil$ contains an error $\mathbf{e}_{\mathrm{mul}}$ from $P^{-1} \cdot \mathbf{a}\mathbf{e}_{\mathsf{extk}}$ and a rounding error $\mathbf{e}_{\mathrm{rnd}}$, where $\mathbf{e}_{\mathsf{extk}} = \langle \mathsf{extk}_{I \to I'}, \mathsf{sk} \rangle - P \cdot s_I$. Note that the rounding error $\mathbf{e}_{\mathrm{rnd}}$ remains the same as the usual MPHE rounding error, since it depends only on the common secret key $\mathsf{sk}_{I'}$. Observe that the error of $\mathsf{extk}_{I \to I'}$ is represented as

$$\mathbf{e}_{\mathsf{extk}} = \mathbf{v} \sum_{i \in I} \mathbf{e}_i + \mathbf{e}_1 \sum_{i \in I} \mathbf{s}_i + \sum_{i \in I' \setminus I} \mathbf{e}_i + \mathbf{e}_2 \ ,$$

where $\mathbf{v}, \mathbf{s}_i \leftarrow \mathrm{Ter}(0.5), \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_i \leftarrow \chi_{\mathrm{err}}$. The variance of the $\mathbf{e}_{\mathsf{extk}}$ is

$$\mathrm{Var}(\mathbf{e}_{\mathsf{extk}}) = |I|N\sigma^2/2 + |I|N\sigma^2/2 + |I' \setminus I|\sigma^2 + \sigma^2 \ ,$$

and $\mathbf{a}$ follows a uniform random distribution over the ciphertext modulus $q_\ell$. Hence, the high-probability canonical norm bound for the error $\mathbf{e}_{\mathrm{mul}}$ is

$$\begin{aligned} B_{\mathrm{mul}} = \|\mathbf{e}_{\mathrm{mul}}\|_\infty^{\mathrm{can}} &= P^{-1} \cdot 16N \sqrt{q_\ell^2/12} \sqrt{|I|N\sigma^2 + (|I' \setminus I| + 1)\sigma^2} \\ &= P^{-1} \cdot (8/\sqrt{3}) \cdot \sigma q_\ell N \sqrt{|I'|N + |I' \setminus I| + 1}. \end{aligned}$$

Since the rounding of the key-switching is performed exactly the same as usual MPHE key-switching rounding over $\mathbb{Z}_{PQ}$, the rounding error bound $B_{\mathrm{rnd}}$ is the same as MPHE. □

Consequently, with the assumption on the magnitude of $\sigma$ that $P^{-1} \cdot \sigma \cdot N^{1.5} = O(1)$, the asymptotic magnitude of the multiplication-induced part becomes

$$O(P^{-1} q_\ell \sigma N^{1.5} \sqrt{|I'|}) = O(q_\ell \sqrt{|I'|}) \ ,$$

whereas the rounding error part has an asymptotic magnitude of $O(N q_\ell \sqrt{|I'|})$. Hence, the impact on the error remains small enough.

Since our dMPHE construction differs from the base MPHE scheme only in the extension-related algorithms, the remaining error analysis, and consequently the correctness established by Lemma 2, remains consistent with the base MPHE scheme for $I'$ parties, even after additional parties join.

[4] Error induced from the multiplication between the key-switching key and the ciphertext component.

### 3.3 Semantic Security of Dynamic MPHE Construction

In this section, we prove the semantic security of our dMPHE construction. In a high level, it suffices to show that the newly introduced features do not leak any further information in conjunction with the original transcripts, such as the ciphertexts or the public keys.

First of all, the extension key is generated from $(\mathbf{a}, \mathbf{b})$, an encryption of 0, which is basically an RLWE sample with respect to a secret vector $\mathbf{s}_I$. One can generate another RLWE sample $(\mathbf{a}+P, \mathbf{b}' = -(\mathbf{a}+P)\mathbf{s}_{I'\setminus I} + \mathbf{e}_{I'\setminus I})$, but with respect to a secret vector $\mathbf{s}_{I'\setminus I} = \sum_{i\in I'\setminus I} \mathbf{s}_i$, which is independent of $\mathbf{s}_I$. Thus, we have the following (multi-secret) RLWE sample with $(0, 0, -P\mathbf{s}_{I'\setminus I})$ added:

$$(\mathbf{a}, -\mathbf{a}\mathbf{s}_I + \mathbf{e}, -\mathbf{a}\mathbf{s}_{I'\setminus I} + \mathbf{e}_{I'\setminus I} - P\mathbf{s}_{I'\setminus I}) \ ,$$

which is computationally indistinguishable from a random triple $(\mathbf{a}, \mathbf{b}, \mathbf{b}') \leftarrow_{\$} \mathcal{R}_{PQ}^3$ under the circular security assumption. The honestly generated extension key $\mathsf{extk}$ is then $(\mathbf{a}+P, \mathbf{b}+\mathbf{b}')$, whose joint distribution with $\mathbf{b}$ can still be viewed as a random tuple in $\mathcal{R}_{PQ}^3$, i.e., the distribution of $(\mathbf{a}+P, \mathbf{b}, \mathbf{b}+\mathbf{b}')$.

Secondly, the updated public keys $\mathsf{pk}_{I'}$ are generated solely with the individual public keys $\mathsf{pk}_i$, which is analogous to the public keys of the base MPHE with $I'$ parties.

Finally, since the ciphertext extensions are performed solely using the former ciphertexts and the public extension keys $\mathsf{extk}$ without using any further secret information, the extended ciphertexts remain secure if the former ciphertexts and the public extension keys are secure.

Before we give a formal security proof in Theorem 1, we first give Lemma 3 as a preparation. The lemma is introduced primarily for the sake of technical completeness in our reduction in Theorem 1, and is a straightforward adaptation of the proof of Lemma 1. Essentially, it demonstrates that the semantic security of the MPHE scheme remains intact even when the challenge ciphertext in the indistinguishability game is encrypted under a partial public key $\mathsf{pk}_\ell := \mathtt{CombKey}(\{\mathsf{pk}_i\}_{1\le i\le \ell})$ instead of the (full) common public key $\mathsf{pk}_I = \mathtt{CombKey}(\{\mathsf{pk}_i\}_{1\le i\le n})$, provided that there is at least one honest party among the parties $\{\mathcal{P}_i\}_{1\le i\le \ell}$.

**Lemma 3.** *The MPHE scheme described in Section 3.1 is partially semantically secure under the RLWE assumption and the circular security assumption with respect to the given parameters. Here, partially semantic security denotes the indistinguishability of encryptions under the partial joint public key $\mathsf{pk}_\ell$ for any $1 \le \ell \le n$, associated with a subset of parties that includes at least one honest party.*

*Proof.* We follow the logic in the proof of Lemma 1, with the only modification being the target public key used in the challenge phase.

We apply the sequence of hybrid games defined in the proof of Lemma 1 with only a single modification: in the challenge phase, the challenger provides a ciphertext $\mathsf{ct}_{I_\ell}$ encrypted under the partial joint public key $\mathsf{pk}_\ell := \mathsf{MPHE.CombKey}(\{\mathsf{pk}_i\}_{1\le i\le \ell})$ instead of the full joint public key $\mathsf{pk}_I$ (or more concretely, with the encryption key $\mathsf{enck}_\ell = (\mathbf{a}_{\mathsf{enck}}, \sum_{i\in I_\ell} \mathbf{b}_{\mathsf{enck},i})$ instead of $\mathsf{enck}_I$).

The indistinguishability between Game 0 and Game 1 remains identical, as it relies on the RLWE and circular security of the individual public keys $\mathsf{pk}_i$ for honest parties $i \in H$, which is independent of which joint key is used for encryption.

In Game 1, since all honest parties' encryption keys $\mathsf{enck}_i$ are replaced with uniform random samples, the partial joint encryption key $\mathsf{enck}_\ell = (\mathbf{a}_{\mathsf{enck}}, \sum_{i=1}^{\ell} \mathbf{b}_{\mathsf{enck},i})$ is also distributed uniformly at random, provided that $I_\ell$ contains at least one honest party ($I_\ell \cap H \neq \emptyset$). Consequently, the transition to Game 2 (encrypting 0 instead of $m_b$) holds under the RLWE assumption, as the challenge ciphertext $\mathsf{ct}_{I_\ell}$ is encrypted under a public key that is computationally indistinguishable from a uniform random sample. □

**Theorem 1** (Semantic Security of the dMPHE Construction)**.** *The dMPHE scheme described in Section 3.2 is semantically secure under the RLWE assumption and the circular security assumption with respect to the given parameters.*

*Proof.* We adopt the notation from Definition 7 and define a sequence of hybrid games as:

- $G_0$: The original security game as defined in Definition 7.
- $G_1$: Similar to $G_0$, except that the challenger samples $\mathsf{extk}_{I_i \to I_{i+1}}$ uniformly at random from $\mathcal{R}^2_{PQ}$ if $(I_{i+1} \setminus I_i) \cap H \neq \emptyset$, instead of running the algorithm $\mathtt{ExtKeyGen}$.
- $G_2$: Similar to $G_1$, except that the challenger now acts as a reduction (as an 'internal MPHE adversary') that interacts with an 'internal MPHE challenger' for the partial semantic security of the base MPHE scheme (as defined in Lemma 3). Specifically, the challenger operates as follows:
  - The challenger requests the internal MPHE challenger to execute $\mathsf{MPHE}.\mathtt{KeyGen}$ for each party $\mathcal{P}_i$, where $1 \leq i \leq n$. Instead of performing $\mathsf{dMPHE}.\mathtt{KeyGen}$, it passes the resulting public keys of all parties and the secret keys of the adversaries (parties in $A$) to the adversary.
  - The challenger executes $\mathsf{dMPHE}.\mathtt{ExtKeyGen}_{I_i \to I_{i+1}}$ and provides the resulting $\mathsf{extk}_{I_i \to I_{i+1}}$ to the adversary if $(I_{i+1} \setminus I_i) \cap H = \emptyset$. Otherwise, it provides a uniformly random tuple in $\mathcal{R}^2_{PQ}$.
  - Upon receiving the chosen messages and the index $\ell$ from the adversary, the challenger forwards the messages to the internal MPHE challenger. The internal challenger selects a bit $b$ and returns a ciphertext encrypted under the MPHE public key $\mathsf{pk}_{I_\ell}$, which the challenger then forwards to the adversary.
  - When the adversary outputs a bit $b'$, the challenger passes this bit back to the internal MPHE challenger.

**Indistinguishability between $G_0$ and $G_1$.** In $G_0$, let us denote a extension key $\mathsf{extk}_{I_i \to I_{i+1}} = (\mathbf{a}' = \mathbf{a} + P, \mathbf{b}'' = \mathbf{b} + \mathbf{b}')$ obtained from $\mathtt{ExtKeyGen}(\mathsf{enck}_{I_i}, \{\mathbf{s}_i\}_{i \in I_{i+1} \setminus I_i})$, where $(\mathbf{a}, \mathbf{b} = -\mathbf{a}\mathbf{s}_{I_i} + \mathbf{e}) \leftarrow \mathtt{Enc}_{\mathsf{enck}_{I_i}}(0)$ and $\mathbf{b}' = -\sum_{i \in I_{i+1} \setminus I_i} (\mathbf{a}'\mathbf{s}_i + \mathbf{e}_i)$ for some $\mathbf{e}$ and $\mathbf{e}_i$'s. Then, the triple $(\mathbf{a}, \mathbf{b}, \mathbf{b}')$ can be decomposed as follows:

$$\begin{aligned}(\mathbf{a}, -\mathbf{a}\mathbf{s}_{I_i \cap H} + \mathbf{e}, -\mathbf{a}\mathbf{s}_{(I_{i+1} \setminus I_i) \cap H} + \mathbf{e}_{I_{i+1} \setminus I_i}) + (0, 0, -P \cdot \mathbf{s}_{(I_{i+1} \setminus I_i) \cap H}) \\ + (0, -\mathbf{a}\mathbf{s}_{I_i \cap A}, -\mathbf{a}\mathbf{s}_{(I_{i+1} \setminus I_i) \cap A} - P \cdot \mathbf{s}_{(I_{i+1} \setminus I_i) \cap A}) \ .\end{aligned}$$

In the case when $I_i \cap H \neq \emptyset$ and $(I_{i+1} \setminus I_i) \cap H \neq \emptyset$, the first summand constitutes a multi-secret RLWE sample in $\mathcal{R}^3_{PQ}$ with respect to the honest parties' secret vectors $\mathbf{s}_{I_i \cap H}$ and $\mathbf{s}_{(I_{i+1} \setminus I_i) \cap H}$ (unless $I_i \cap H = \emptyset$). The second summand consists of the honest parties' secret vector multiplied by the constant $P$. Under the circular security assumption, the sum of these first two summands is indistinguishable from a triple chosen uniformly at random from $\mathcal{R}^3_{PQ}$. Since the final summand consists of values computable by the adversary, when added to the uniform random triple, the joint distribution $(\mathbf{a}, \mathbf{b}, \mathbf{b}')$ is indistinguishable from a uniform random triple. Consequently, the extension keys $(\mathbf{a} + P, \mathbf{b} + \mathbf{b}')$ themselves are also indistinguishable from uniform random tuples. We note that the extension keys serve as inputs to the $\mathtt{Extend}$ algorithm alongside the ciphertexts, effectively acting as a key-switching key $\mathsf{swk}_{I_i \to I_{i+1}}$. Given that $I_i \cap H \neq \emptyset$ and $(I_{i+1} \setminus I_i) \cap H \neq \emptyset$, performing a key-switching operation with a uniform random sample instead of a valid key-switching key does not lead to any non-negligible computational advantage for the adversary, since $(\mathsf{ct}_{I_i}, \mathtt{Extend}(\mathsf{extk}_{I_i \to I_{i+1}}, \mathsf{ct}_{I_i}), \mathsf{extk}_{I_i \to I_{i+1}})$ and $(\mathsf{ct}_{I_i}, \mathtt{Extend}((\mathbf{a}^\$, \mathbf{b}^\$), \mathsf{ct}_{I_i}), (\mathbf{a}^\$, \mathbf{b}^\$))$ are computationally indistinguishable where $(\mathbf{a}^\$, \mathbf{b}^\$) \leftarrow_\$ \mathcal{R}^2_{PQ}$. Thus, replacing the honest extension keys with uniform random samples does not affect the adversary's success probability between the games $G_0$ and $G_1$.

In the case when $I_i \cap H = \emptyset$ and $(I_{i+1} \setminus I_i) \cap H \neq \emptyset$, the extension key components $(\mathbf{a}', \mathbf{b}'')$ can be decomposed as follows:

$$(\mathbf{a}, -\mathbf{as}_{(I_{i+1}\setminus I_i)\cap H} + \mathbf{e}_{I_{i+1}\setminus I_i} + \mathbf{e}) + (0, -P \cdot \mathbf{s}_{(I_{i+1}\setminus I_i)\cap H}) \\ +(P, -\mathbf{as}_{I_{i+1}\cap A} - P \cdot \mathbf{s}_{(I_{i+1}\setminus I_i)\cap A}) \ .$$

The first term constitutes an RLWE sample with respect to the secret keys of the honest parties, and the second term consists of the honest parties' secret vector multiplied by a constant. Consequently, under the circular security assumption, the sum of these first two terms is indistinguishable from a uniform random tuple to the adversary. The third term is entirely computable by the adversary; thus, when added, the resulting tuple $(\mathbf{a}', \mathbf{b}'')$ remains indistinguishable from a uniform random tuple. As in the previous case, this leads to a computational distinguishability between $G_0$ and $G_1$.

**Indistinguishability between $G_1$ and $G_2$.** The primary difference between $G_1$ and $G_2$ is that the $G_2$ challenger acts as a reduction to the *partial* semantic security of the base MPHE scheme. We argue that the adversary's view in both games is identical.

First, the key generation process is the same in both games; thus, all public keys and the secret keys for parties in $A$ are indistinguishable.

Second, the extension keys are identically generated. In both $G_1$ and $G_2$, extension keys $\mathsf{extk}_{I_i \to I_{i+1}}$ for $(I_{i+1} \setminus I_i) \cap H \neq \emptyset$ are sampled uniformly at random. Otherwise, we have $(I_{i+1} \setminus I_i) \subseteq A$, thus, the extension keys are generated using only the public keys and the secret keys of the adversaries. Since the challenger in $G_2$ (acting as an internal MPHE adversary) possesses the same secret information as the adversary in $G_1$ and $G_2$, the extension keys are indistuinguishable in the two games.

Third, the challenge ciphertext in $G_2$ is generated by the internal MPHE challenger using $\mathsf{pk}_{I_\ell}$ (specifically, the encryption key $\mathsf{enck}_{I_\ell}$). From the adversary's perspective, this ciphertext is generated identically to the challenge ciphertext in $G_1$, as all the operations in both games are identically defined.

**Advantage of the adversary in $G_2$.** Finally, as $G_2$ is designed as a reduction, any success in the dMPHE security game is thus directly translated into a success against the underlying MPHE scheme's *partial* semantic security. Thanks to Lemma 3, the latter is negligible under the RLWE and circular security assumptions. Thus, the advantage of the adversary in $G_2$ is also negligible. This concludes the proof. □

# 4 Dynamic MPHE with Reusable Ciphertexts

In this section, we introduce our Reusable Dynamic MPHE (rdMPHE) scheme, which extends the functionality of dMPHE to support practical cloud computing scenarios. While dMPHE provides the foundational mechanism for parties to join dynamically, rdMPHE further enables ciphertext reusability, allowing individually encrypted data to be seamlessly integrated into joint computations without redundant re-encryption or transmission.

We begin by formally defining the rdMPHE framework and discussing the concept of homomorphic computation with reusable ciphertexts, and provide a concrete construction of the rdMPHE scheme, including its noise analysis and proof of semantic security. In addition, we provide an asymptotic comparison between our rdMPHE and other existing multi-party encryption solutions for ciphertext reusing scenario, highlighting its efficiency in terms of computation and communication cost.

## 4.1 Homomorphic Computation with Ciphertext Reusability

In real-world applications, we consider a cloud computing scenario where multiple parties aim to jointly execute computations on their private data while ensuring that no information

other than the final result is leaked. In such a scenario, it is desirable for the ciphertexts stored in the cloud to be directly compatible with joint computations. This eliminates the need for parties to re-encrypt messages for every session or to decrypt and then re-encrypt them under a specific key for each joint task. We refer to this property as *ciphertext reusability*.

Furthermore, we consider a dynamic setting that allows parties to join during the computation at any time, contributing their data as additional inputs. To address these requirements, we define Reusable Dynamic Multi-Party Homomorphic Encryption (rdMPHE) as follows.

**Definition 8** (Reusable Dynamic Multi-Party Homomorphic Encryption)**.** A Reusable Dynamic Multi-Party Homomorphic Encryption $\mathsf{rdMPHE}$ is a tuple of efficient algorithms $(\mathtt{KeyGen}, \mathtt{CombKey}, \mathtt{Enc}, \mathtt{SwKeyGen}, \mathtt{Switch}, \mathtt{Eval}, \mathtt{Dec}, \mathtt{ExtKeyGen}, \mathtt{Extend}, \mathtt{ExtendSwk})$ run by multiple parties $\{\mathcal{P}_i\}_{i\in I}$ for any index set $I$. The algorithms $\mathtt{KeyGen}, \mathtt{CombKey}, \mathtt{Enc}, \mathtt{Eval}, \mathtt{Dec}$, $\mathtt{ExtKeyGen}$, and $\mathtt{Extend}$ are defined identically to those in $\mathsf{dMPHE}$. The $\mathtt{SwKeyGen}$, $\mathtt{Switch}$, and $\mathtt{ExtendSwk}$ algorithms are defined as follows:

- $\underline{\mathtt{SwKeyGen}(\mathsf{pk}_I, \mathsf{sk}_i)}$: Run by a party $\mathcal{P}_i$, it takes as inputs a public key $\mathsf{pk}_I$ and a secret key $\mathsf{sk}_i$ where $i \in I$, and outputs a public switching key $\mathsf{swk}_{i\to I}$.
- $\underline{\mathtt{Switch}(\mathsf{swk}_{i\to I}, \mathsf{ct}_i)}$: It takes as inputs a public switching key $\mathsf{swk}_{i\to I}$ and a ciphertext $\mathsf{ct}_i$ encrypted under $\mathsf{sk}_i$. It outputs a ciphertext $\mathsf{ct}_I$, which encrypts the same message as $\mathsf{ct}_i$ but under the joint key $\mathsf{sk}_I$.
- $\underline{\mathtt{ExtendSwk}(\mathsf{swk}_{i\to I}, \{\mathsf{sk}_j\}_{j\in I'\setminus I})}$: It takes as inputs a public switching key $\mathsf{swk}_{i\to I}$ and a set of secret keys $\{\mathsf{sk}_j\}_{j\in I'\setminus I}$ of the joining parties. It outputs an updated switching key $\mathsf{swk}_{i\to I'}$, which allows switching a ciphertext $\mathsf{ct}_i$ into $\mathsf{ct}_{I'}$.

The rdMPHE framework combines the advantages of both MPHE and MKHE: it features compact ciphertexts and compact common public keys, as in MPHE, while also allowing ciphertexts to be initially encrypted with each party's individual key and subsequently used for joint computation, as in MKHE. Ciphertexts encrypted under an individual party's key are key-switched into a common key corresponding to a set of parties tailored to the specific application. Consequently, the ciphertexts can be reused across different homomorphic computations involving various sets of participating parties, which significantly reduces the communication overhead associated with re-transmitting encrypted data.

The correctness and semantic security of rdMPHE are defined analogously to those of dMPHE, with the extension that ciphertexts can be freshly encrypted, evaluated, $\mathtt{Extend}$-ed, or $\mathtt{Switch}$-ed. We give their formal definitions below.

**Definition 9** (Correctness of rdMPHE)**.** Let $I_1 \subsetneq I_2 \subsetneq \cdots \subsetneq I_k$ be a sequence of sets of parties, and let $\mathsf{pk}_{I_\ell} \leftarrow \mathtt{CombKey}(\{\mathsf{pk}_i\}_{i\in I_\ell})$ be the common public key for each set $I_\ell$. For any $1 \le \ell \le k$ and $j \in \{1, \ldots, m\}$, let $\mathsf{ct}_{I_\ell,j}$ be a ciphertext encrypting $m_j$ under the joint key $\mathsf{sk}_{I_\ell}$, where $\mathsf{ct}_{I_\ell,j}$ is obtained via one of the following, possibly applied recursively: i) (Fresh Encryption) $\mathsf{ct}_{I_\ell,j} \leftarrow \mathtt{Enc}_{\mathsf{pk}_{I_\ell}}(m_j)$, ii) (Switching) $\mathsf{ct}_{I_\ell,j} \leftarrow \mathtt{Switch}(\mathsf{swk}_{i\to I_\ell}, \mathsf{ct}_i)$ where $\mathsf{ct}_i \leftarrow \mathtt{Enc}_{\mathsf{pk}_i}(m_j)$ for some $i \in I_\ell$, or iii) (Extension) $\mathsf{ct}_{I_\ell,j} \leftarrow \mathtt{Extend}(\mathsf{extk}_{I_{\ell-1}\to I_\ell}, \{\mathsf{pk}_i\}_{i\in I_\ell}, \mathsf{ct}_{I_{\ell-1},j})$ where $\mathsf{ct}_{I_{\ell-1},j}$ is a valid ciphertext under $\mathsf{sk}_{I_{\ell-1}}$, where the $\mathsf{swk}$ and $\mathsf{extk}$ are honestly generated by $\mathtt{SwKeyGen}, \mathtt{ExtendSwk}$, and $\mathtt{ExtKeyGen}$.

Then, the rdMPHE scheme is said to be correct if for any index $1 \le \ell \le k$, any circuit $C : \mathcal{M}^m \to \mathcal{M}$ whose depth is bounded by the scheme's parameters, and any messages $m_1, \ldots, m_m \in \mathcal{M}$, the following holds with overwhelming probability in the security parameter $\lambda$:

$$\mathtt{Dec}\left(\{\mathsf{sk}_i\}_{i\in I_\ell}; \mathtt{Eval}_{\mathsf{pk}_{I_\ell}}(C, (\mathsf{ct}_{I_\ell,1}, \ldots, \mathsf{ct}_{I_\ell,m}))\right) = C(m_1, \ldots, m_m) \ .$$

**Definition 10** (Semantic Security of rdMPHE in the Semi-honest Model)**.** Let $I_1 \subsetneq I_2 \subsetneq \cdots \subsetneq I_k = \{\mathcal{P}_i\}_{1\le i\le n}$ be a sequence of a set of parties. Let $A \subset I_k$ be a set of adversarial parties, and $H = I_k \setminus A$. An rdMPHE scheme $\Pi = (\texttt{KeyGen}, \texttt{CombKey}, \texttt{Enc}, \texttt{SwKeyGen}, \texttt{Switch}, \texttt{Eval}, \texttt{Dec}, \texttt{ExtKeyGen}, \texttt{Extend})$ is said to be secure if the advantage of $\mathcal{A}$ in the following game is negligible in the security parameter $\lambda$, for any PPT adversary $\mathcal{A}$:

- The challenger executes $\texttt{KeyGen}(1^\lambda)$ for each $\mathcal{P}_i$, for $1 \le i \le n$, and sends the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ and secret keys $\{\mathsf{sk}_i : i \in A\}$ to the adversary.
- The challenger executes $\texttt{ExtKeyGen}_{I_i \to I_{i+1}}(\{\mathsf{pk}_i\}_{i\in I_i}, \{\mathsf{sk}_i\}_{i\in I_{i+1}})$ for all $i = 1, \ldots, k-1$, and sends the extension keys $\mathsf{extk}_{I_i \to I_{i+1}}$ to the adversary.
- The challenger executes $\texttt{SwKeyGen}(\mathsf{pk}_{I_j}, \mathsf{sk}_i)$ and generates $\mathsf{extk}_{i\to I_j}$ if $i \in I_j \setminus I_{j-1}$, where $I_0 := \emptyset$. The challenger then recursively executes $\texttt{ExtendSwk}(\mathsf{swk}_{i\to I_j}, \{\mathsf{sk}_m\}_{m\in I_{j+1}\setminus I_j})$, $\cdots$, $\texttt{ExtendSwk}(\mathsf{swk}_{i\to I_{k-1}}, \{\mathsf{sk}_m\}_{m\in I_k\setminus I_{k-1}})$ to extend the switching keys. The challenger then sends the (extended) switching keys $\{\mathsf{swk}_{i\to I_j}, \cdots, \mathsf{swk}_{i\to I_k}\}$ for every $1 \le i \le n$.
- The adversary chooses messages $m_0, m_1 \in \mathcal{M}$ and an index $1 \le \ell \le k$, where $I_\ell \cap H \neq \emptyset$, then sends them back to the challenger; The challenger samples a random bit $b \in \{0, 1\}$ and sends $\mathsf{ct}^* \leftarrow \texttt{Enc}_{\mathsf{pk}_{I_\ell}}(m_b)$ back to the adversary.
- The adversary $\mathcal{A}$ outputs a bit $b'$. The advantage is defined as $|\Pr[b = b'] - \frac{1}{2}|$.

### 4.1.1 Real-World Scenarios

The strength of rdMPHE lies in its data-centric structure, which enables the reusability of encrypted data. In addition, rdMPHE has advantages in both the lightness of a single-key HE and the flexibility of MKHE, with minimal communication prior to evaluation. The following protocol illustrates how an rdMPHE scheme can be effectively applied in practice.

- Data Accumulation: Data providers (parties) execute `KeyGen` to generate their secret and public keys. They then run `Enc` to encrypt their data under their individual keys. The encrypted data, along with the corresponding public keys, is uploaded to the cloud. Each ciphertext is uploaded only once and can be reused for multiple independent computations.
- Set Up: For a specific computation, a set of participating parties $I$ is identified. The server then runs `CombKey` to generate a common public key $\mathsf{pk}_I$, which is shared with the participants or broadcast. Each participant then runs `SwKeyGen` to generate their partial switching key, granting permission to include their data in the computation.
- Computation: The server executes `Switch` to transform the input ciphertexts from individual keys to the common joint key $\mathsf{sk}_I$. Subsequently, the server performs the desired homomorphic evaluations via `Eval` using the common public keys.
- Join: On demand, a new data provider can join at any time. The joining parties execute `ExtKeyGen` to generate extension keys and `SwKeyGen` to generate their switching keys, and `ExtendSwk` to generate the existing parties' switching keys. The server then executes `CombKey` and `Extend` to generate a new common key for the expanded set of parties and to extend the existing ciphertexts to the new joint key, respectively.
- Decryption: Upon completion, the resulting ciphertexts are sent to the participating parties. They jointly perform `Dec`, and the decrypted result is sent to the designated receiver.

We list some of the useful privacy-preserving applications below.

**Privacy-Preserving Machine Learning.** Privacy-preserving machine learning (PPML) training is a promising application of the proposed scenario, as it benefits from ciphertext reusability and dynamic participation. In this setting, the server trains a model over encrypted data stored in the cloud, where each data set belongs to different parties and is encrypted under their individual keys. New data contributed by additional parties can be seamlessly incorporated to improve the model without degrading the performance of the homomorphic computations. Once training is complete, the model can be jointly decrypted with the consent of all participants.

In federated learning, each user or group independently trains a local model, and the server aggregates the encrypted updates to build a global model. This approach preserves privacy while enabling incremental improvements as new contributors join over time. The capability to repeatedly use ciphertexts in several training iterations and to flexibly add new data providers aligns well with the inherent features of rdMPHE.

Previous PPML schemes, including federated learning frameworks [KSK+18, LLK+23, XLG+23, KCSPS23], typically required that all data providers be fixed in advance or suffered from low efficiency in homomorphic evaluation. In contrast, rdMPHE supports dynamic participation without causing communication costs that grow with the number of parties.

**Privacy-Preserving Digital Asset Management.** Another representative application of rdMPHE is a privacy-preserving Data Lake [WA15, CSN+14, TSRC15, QHV16]. In this setting, each data owner independently encrypts their raw data–such as logs, images, or product metadata–without any pre-processing or transformation and uploads them to the Data Lake. The Data Lake serves as a central repository and can later provide this encrypted data for diverse computations, including quality assessment, auditing, or media asset indexing. Unlike conventional MKHE or MPHE, our approach enables efficient homomorphic computations when the computing server incorporates data from new parties, without requiring re-encryption or coordination with all existing data providers. This design simplifies key management and reduces the operational burden of maintaining consistent encryption across all participants.

## 4.2 Reusable Dynamic MPHE Scheme

We now extend our dMPHE construction into rdMPHE, as a family of MKHE by allowing messages to be encrypted via individual party's key, then key-switching them into ciphertexts under a common key. This can be viewed as an MKHE to dMPHE conversion, which is an extension of Park's conversion [Par21]. While maintaining dynamicity and ciphertext reusability, our rdMPHE construction enjoys the compactness of MPHE schemes in terms of both ciphertext sizes and homomorphic operation complexity.

Our rdMPHE, which shares the main structures with the dMPHE scheme but is modified to introduce reusability–the `KeyGen`, `CombKey`, `Enc`, `Eval`, `Dec`, `ExtKeyGen`, and `Extend` algorithms are identical to those of dMPHE. We further provide the three algorithms `SwKeyGen`, `Switch`, and `ExtendSwk` as an ad-hoc extension to our dMPHE scheme, which allows switching the ciphertexts encrypted under a party's key to under a common key. Note, the `ExtendSwk` algorithm allows extending the switching keys when new parties join.

- $\underline{\texttt{SwKeyGen}(\mathsf{enck}_I, \mathsf{sk}_i)}$: If $i \in I$, it outputs $\mathsf{swk}_{i\to I} := \mathsf{Enc}_{\mathsf{enck}_I}(P \cdot \mathbf{s}_i) \in \mathcal{R}^2_{PQ}$;
- $\underline{\texttt{Switch}(\mathsf{swk}_{i\to I}, \mathsf{ct}_i)}$: For $\mathsf{ct}_i \in \mathcal{R}^2_Q$, it outputs $\mathsf{dMPHE.KS}(\mathsf{swk}_{i\to I}, \mathsf{ct}_i)$;
- $\underline{\texttt{ExtendSwk}(\mathsf{swk}_{i\to I}, \{\mathsf{sk}_j\}_{j\in I'\setminus I})}$: For $\mathsf{swk}_{i\to I} = (\mathbf{a}, \mathbf{b})$, it updates the switching key by generating $\mathsf{swk}_{i\to I'} := \mathsf{swk}_{i\to I} - \sum_{j\in I'\setminus I}(0, \mathbf{a}\mathbf{s}_j + \mathbf{e}_j)$, where $\mathbf{e}_j \leftarrow \chi_{\mathrm{err}}$.

We note that the switching keys $\mathsf{swk}_{i\to I'}$ for $i \in I$ might not need to be updated, if the latter computation does not require the ciphertexts of the parties in $I$.

#### 4.2.1 Correctness of Reusable Dynamic MPHE Scheme.

For the correctness of the above rdMPHE construction, it suffices to show that the `Switch` algorithm works correctly with the switching keys generated from `SwKeyGen`, and with the switching keys generated from `ExtKeyGen`.

First, let $\mathsf{swk}_{i\to I} = (\mathbf{a}_i, \mathbf{b}_i)$, 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 small error $\mathbf{e}_i$ and hence the `Switch`, which is basically the key-switching operation, works correctly if the error term is relatively smaller than $\Delta/2$. 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.

Secondly, we demonstrate that the `Switch` algorithm operates correctly when new parties join with `ExtKeyGen`. The switching keys are updated from $\mathsf{swk}_{i\to I}$ to $\mathsf{swk}_{i\to I'}$ during `ExtKeyGen`. We assume that the switching key $\mathsf{swk}_{i\to I}$ 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 $\mathsf{swk}_{i\to I} = (\mathbf{a}_i, \mathbf{b}_i)$, $\mathsf{swk}_{i\to I'} = (\mathbf{a}_i, \mathbf{b}'_i)$, $\mathbf{e}'_i = \mathbf{e}_i - \mathbf{e}_{I'\setminus I}$, 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 huge.

In Lemma 4, we formally analyze the error after `Switch` operations, in the case of CKKS-based dMPHE constructions. The lemma implies that the error introduced by `Switch` can be managed as small as the usual key-switching operations in MPHE with the same number of parties. In addition, the experimental results in Section 5 highlight their practical impact on the error, during homomorphic computation. We note that the lemma can be seamlessly extended to B/FV-based constructions, and further to BGV-based constructions.

**Lemma 4** (Error Analysis for Switched rdMPHE Ciphertexts)**.** *Consider the above rdMPHE scheme constructed upon a CKKS-based dMPHE scheme. Let us borrow assumptions on the secret and error coefficients, as well as the canonical embedding norm bound of the base MPHE key-switching error. Then, for an rdMPHE ciphertext switched from $i$ to $I$, the switch error is bounded by $B_{\mathrm{swk}} = B_{\mathrm{mul}} + B_{\mathrm{rnd}}$ with overwhelming probability in the security parameter $\lambda$, where*

$$B_{\mathrm{mul}} = \begin{cases}\sqrt{N + 1/|I|}\cdot B^{\mathsf{MPHE}}_{\mathrm{mul}} & \text{when } \mathsf{swk} \leftarrow \texttt{SwKeyGen}\ ,\\ \sqrt{N + (1+|I'\setminus I|)/|I|}\cdot B^{\mathsf{MPHE}}_{\mathrm{mul}} & \text{when } \mathsf{swk} \leftarrow \texttt{ExtendSwk}\ ,\end{cases}$$

*and $B_{\mathrm{rnd}} = B^{\mathsf{MPHE}}_{\mathrm{rnd}}$.*

*Proof.* Let $\mathsf{ct} = (\mathbf{a}, \mathbf{b})$ be a ciphertext with respect to the party $P_i$, and consider when it is key-switched with $\mathsf{swk}_i$. Recall that $(0, \mathbf{b}) + \lfloor P^{-1}\cdot\mathbf{a}\cdot\mathsf{swk}_i\rceil$ contains an error $\mathbf{e}_{\mathrm{mul}}$ from $P^{-1}\cdot\mathbf{a}\mathbf{e}_{\mathsf{swk}}$ and a rounding error $\mathbf{e}_{\mathrm{rnd}}$, where $\mathbf{e}_{\mathsf{swk}} = \langle\mathsf{swk}_i, \mathsf{sk}\rangle - P\cdot s_i$. Note that the rounding error $\mathbf{e}_{\mathrm{rnd}}$ remains the same as the usual MPHE rounding error, since it depends only on the common secret key $\mathsf{sk}_{I'}$. Observe that the error of $\mathsf{swk}_i$ is represented as

$$\mathbf{e}_{\mathsf{swk}} = \begin{cases}\mathbf{e}_1\sum_{i\in I}\mathbf{s}_i + \mathbf{v}\sum_{i\in I}\mathbf{e}_i + \mathbf{e}_2 & \text{for } \mathsf{swk}_i \text{ from } \texttt{SwKeyGen}\ ,\\ \mathbf{e}_1\sum_{i\in I}\mathbf{s}_i + \mathbf{v}\sum_{i\in I}\mathbf{e}_i + \mathbf{e}_2 + \sum_{j\in I'\setminus I}\mathbf{e}_j & \text{for } \mathsf{swk}_i \text{ from } \texttt{ExtendSwk}\ ,\end{cases}$$

where $\mathbf{v}, \mathbf{s}_i \leftarrow \mathrm{Ter}(0.5), \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_i, \mathbf{e}_j \leftarrow \chi_{\mathrm{err}}$. The variance of the $\mathbf{e}_{\mathsf{swk}}$ is

$$\mathrm{Var}(\mathbf{e}_{\mathsf{swk}}) = \begin{cases}|I|N\sigma^2/2 + |I|N\sigma^2/2 + \sigma^2 & \text{and}\\ |I|N\sigma^2/2 + |I|N\sigma^2/2 + \sigma^2 + |I'\setminus I|\sigma^2, & \text{respectively,}\end{cases}$$

and $\mathbf{a}$ follows a uniform random distribution over the ciphertext modulus $q_\ell$. Hence, the high-probability canonical norm bound for the error $\mathbf{e}_{\mathrm{mul}}$ when using $\mathsf{swk}_i$ obtained from $\mathtt{SwKeyGen}$ is

$$\begin{aligned} B_{\mathrm{mul}}^{\mathtt{SwKeyGen}} = \|\mathbf{e}_{\mathrm{mul}}\|_\infty^{\mathrm{can}} &= P^{-1} \cdot 16N\sqrt{q_\ell^2/12}\sqrt{|I|N\sigma^2 + \sigma^2} \\ &= P^{-1} \cdot (8/\sqrt{3}) \cdot \sigma q_\ell N \sqrt{|I|N + 1}, \end{aligned}$$

and the same bound for the error $\mathbf{e}_{\mathrm{mul}}$ when using $\mathsf{swk}_i$ obtained from $\mathtt{ExtendSwk}$ is

$$\begin{aligned} B_{\mathrm{mul}}^{\mathtt{ExtendSwk}} = \|\mathbf{e}_{\mathrm{mul}}\|_\infty^{\mathrm{can}} &= P^{-1} \cdot 16N\sqrt{q_\ell^2/12}\sqrt{|I|N\sigma^2 + \sigma^2 + |I' \setminus I|\sigma^2} \\ &= P^{-1} \cdot (8/\sqrt{3}) \cdot \sigma q_\ell N \sqrt{|I|N + 1 + |I' \setminus I|}. \end{aligned}$$

Since the rounding of the key-switching is performed exactly the same as usual MPHE key-switching rounding over $\mathbb{Z}_{PQ}$, the rounding error bound $B_{\mathrm{rnd}}$ is the same as MPHE. □

Consequently, with the assumption on the magnitude of $\sigma$ that $P^{-1} \cdot \sigma \cdot N^{1.5} = O(1)$, the asymptotic magnitudes of the multiplication-induced parts of the two cases become the same.

$$O(P^{-1} q_\ell \sigma N^{1.5} \sqrt{|I|}) = O(q_\ell \sqrt{|I|}) \ ,$$

whereas the rounding error part has an asymptotic magnitude of $O(N q_\ell \sqrt{|I|})$. Hence, the impact on the error remains small enough.

As a side note, if we apply an automorphism to a ciphertext before or after some party joins, we can fuse the automorphism with the $\mathtt{Extend}$ algorithms for such ciphertexts, reducing half the cost, since they are basically the key-switching algorithm.

### 4.3 Semantic Security of Reusable Dynamic MPHE Scheme

We now prove the sematic security of our rdMPHE construction in Theorem 2. The security relies on the fact that the added switching keys and the modified extension process do not leak any additional information about the honest parties' secret keys beyond what is already present in the underlying dMPHE.

**Theorem 2** (Semantic Security of the rdMPHE Construction)**.** *The rdMPHE scheme described in Section 4.2 is semantically secure under the RLWE assumption and the circular security assumption, provided that the underlying dMPHE scheme is semantically secure.*

*Proof.* We adopt the notation from Definition 10 and define a sequence of hybrid games as:

- $G_0$: The original semantic security game for rdMPHE as defined in Definition 10.
- $G_j$ for $1 \le j \le k$: Similar to $G_{j-1}$, except that the $\mathbf{b}$-parts of the switching keys $\mathsf{swk}_{i \to I_{k+1-j}}$ are replaced by uniform random polynomials from $\mathcal{R}_{PQ}$ if $i \in I_{k-j} \cap H$; and the switching key $\mathsf{swk}_{i \to I_{k+1-j}}$ are replaced by uniform random tuple from $\mathcal{R}_{PQ}^2$ for $i \in (I_{k+1-j} \setminus I_{k-j}) \cap H$.
- $G_{k+1}$: Similar to $G_k$, except that the challenger now acts as a reduction (as an 'internal dMPHE adversary') that interacts with an 'internal dMPHE challenger' for the semantic security of the base dMPHE scheme. Specifically, it operates as follows:
  - For $\mathtt{KeyGen}$ and $\mathtt{ExtKeyGen}$, it asks to the internal dMPHE challenger who runs the $\mathsf{dMPHE}.\mathtt{KeyGen}$ and $\mathsf{dMPHE}.\mathtt{ExtKeyGen}$. The public keys, adversaries' secret keys, and the extension keys are obtained, and sent to the adversary.
  - For $\mathtt{SwKeyGen}$ and $\mathtt{ExtendSwk}$, the challenger sends the uniform random tuples (if $i \in H \cap I_j$ and $(I_j \setminus I_{j-1}) \cap H \neq \emptyset$) or the recursively generated (otherwise) $\mathsf{swk}_{i \to I_j}$, as in $G_k$.

– Upon receiving the messages and $\ell$, the challenger passes them to the internal challenger, and sends the challenge ciphertext obtained from the internal challenger, which is a dMPHE ciphertext encrypted using $\mathsf{pk}_{I_\ell}$.

**Indistinguishability between $G_{j-1}$ and $G_j$ for $1 \le j \le k$.** The difference between $G_{j-1}$ and $G_j$ lies in the switching keys $\mathsf{swk}_{i \to I_{k+1-j}}$ for all $i \in H \cap I_{k+1-j}$. For $i \in H \cap (I_{k+1-j} \setminus I_{k-j})$, the switching key is an encryption of $P \cdot \mathbf{s}_i$ with respect to $\mathsf{enck}_{I_{k+1-j}}$, and all the switching keys of $i$ can be induced from the adversaries' secret key. Hence, it is indistinguishable when replaced by a random tuple in $\mathcal{R}_{PQ}^2$. For $i \in H \cap I_{k-j}$, we have multiple switching keys that cannot be induced by the adversary, hence their joint distribution should be considered:

$$(\mathbf{a}, \mathbf{b}_m := -\mathbf{a}\mathbf{s}_{I_m} + \mathbf{e}_m + P \cdot \mathbf{s}_i, \cdots, \mathbf{b}_{k-j}, \mathbf{b}_{k+1-j} := \mathbf{b}_{k-j} - \mathbf{a}\mathbf{s}_{J \cap H} - \mathbf{a}\mathbf{s}_{J \cap A} + \mathbf{e}_{k+1-j}) \ ,$$

for some $1 \le m \le k-j$, error $\mathbf{e}_m, \cdots, \mathbf{e}_{k+1-j}$, and $J := I_{k+1-j} \setminus I_{k-j}$. By subtracting consecutive terms and eliminating adversary-inducible terms, we have $(\mathbf{a}, -\mathbf{a} \cdot \mathbf{s}_{I_m \cap H} + \mathbf{e}_m + P \cdot \mathbf{s}_i, \cdots, -\mathbf{a} \cdot \mathbf{s}_{J \cap H} + \mathbf{e}_{k+1-j})$. Since (at least) the secret keys and errors used in the last term are independent from others, it is computationally indistinguishable when the last term is replaced by a uniform random sample from the (multi-secret) RLWE assumption. Therefore, the two games $G_{j-1}$ and $G_j$ are computationally indistinguishable to the adversary.

**Indistinguishability between $G_k$ and $G_{k+1}$.** In $G_k$, all switching keys associated with honest parties that involve at least one honest party's secret key during generation or update have been replaced by uniform random tuples. The remaining switching keys are those that can be perfectly simulated using only the adversarial secret keys $\{\mathsf{sk}_i\}_{i \in A}$ and the public transcripts. Therefore, the challenger in $G_{k+1}$ can perfectly simulate the adversary's view in $G_k$ using the information provided by the internal dMPHE challenger. The challenge ciphertext and extension keys are distributed identically in both games.

**Advantage of the adversary in $G_{k+1}$.** Since the rdMPHE challenger (as an internal dMPHE adversary) only uses the information that the internal dMPHE challenger gives, any success of the rdMPHE adversary is directly translated into a success against the underlying dMPHE security game. By Theorem 1, the latter is negligible under the RLWE and circular security assumptions, it follows that the advantage in $G_{k+1}, \cdots, G_0$, are also negligible. This concludes the proof. □

**Extension to Malicious Setting.** Extending rdMPHE to a malicious setting involves integrating Zero Knowledge (ZK) proofs or Non-Interactive Zero Knowledge (NIZK) proofs for `KeyGen` (including `ExtKeyGen` and `SwKeyGen`) and `Enc`, consistent with standard MPHE frameworks [AJL+12, HLSS25]. Additionally, ZK proofs are required for `ExtendSwk` where secret keys are involved. Beyond these steps, all homomorphic evaluations are performed on public values such as the ciphertexts, public keys, and the circuits, and are thus publicly verifiable.

## 4.4 Complexity Analysis.

In this subsection, we provide a comprehensive asymptotic analysis of the computational and communication complexities for rdMPHE. We compare our framework with existing multi-party HE primitives MPHE, MKHE, and MGHE, focusing on their scalability and efficiency in large-scale dynamic networks.

**Unit Operation and Asymptotic Baseline.** To establish a rigorous baseline, we define the complexities of core cryptographic operations. We denote $N$ as the ring degree,

$Q$ as the ciphertext modulus, $P$ as the auxiliary modulus, and $d$ as the gadget rank for switching keys ($P \approx Q^{1/d}$, hence $d \log PQ \approx (d+1) \log Q$).

*Computational Unit Costs.* The dominant factor in most operations is the Number Theoretic Transform (NTT), with a complexity of $\mathcal{O}(N \log N \log PQ)$.

- Key Generation and Encryption: A public key generation involves $\mathcal{O}(d)$ Hadamard multiplications, running in $\mathcal{O}(dN \log PQ)$. A single encryption takes $\mathcal{O}(N \log Q)$ or $\mathcal{O}(dN \log PQ)$ depending on the resulting modulus.
- Key-Switching and Rotation: These algorithms involve $\mathcal{O}(d)$ ModUps, $\mathcal{O}(1)$ ModDown, and $\mathcal{O}(d)$ Hadamard multiplications in $\mathcal{R}_{PQ}$, resulting in $\mathcal{O}(dN \log N \log PQ)$.
- Homomorphic Evaluations: An addition takes $\mathcal{O}(N \log Q)$. A multiplication is dominated by the key-switching cost, thus yielding $\mathcal{O}(dN \log N \log PQ)$.

As summarized in Table 2, rdMPHE maintains the inherent efficiency of standard MPHE. While MKHE and MGHE suffer from costs that scale linearly with the number of parties ($n$) or groups ($n_g$), rdMPHE preserves compact ciphertexts and constant-time homomorphic operations.

Table 2: Computational complexities of unit homomorphic operations. The common factor $N \log Q$ is omitted.

| | Ctxt Size | Homomorphic Operations | | |
|---|---|---|---|---|
| | | Add. | Mult. | Aut. |
| MPHE [MBH23] | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(d \log N)$ | $\mathcal{O}(d \log N)$ |
| MKHE [KKL+23] | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(nd \log N)$ | $\mathcal{O}(nd \log N)$ |
| MGHE [KLSW24] | $\mathcal{O}(n_g)$ | $\mathcal{O}(n_g)$ | $\mathcal{O}(n_g d \log N)$ | $\mathcal{O}(n_g d \log N)$ |
| rdMPHE (ours) | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(d \log N)$ | $\mathcal{O}(d \log N)$ |

**Efficiency in Reusable Ciphertext Scenarios.** We analyze a scenario requiring joint computations for $M$ independent sessions, each involving $k$ ciphertexts. Let $C_{\text{single}}$ be the circuit complexity, and for normalized comparison, let $c_{\text{single}} = C_{\text{single}}/(N \log Q)$.

*Computational Cost Analysis.* The primary advantage of rdMPHE lies in its ability to reuse ciphertexts while maintaining computational efficiency.

- MPHE [MTBH21, MBH23]: Lacks reusability; thus, encryption must be performed for each session, taking $\mathcal{O}(MNk \log Q)$. The total complexity is $\mathcal{O}((Mc_{\text{single}} + Mk + d)N \log Q)$.
- MKHE [KKL+23]: Although it supports reusability, every evaluation is $\mathcal{O}(n)$ times slower due to the $n$-component ciphertext structure, resulting in $\mathcal{O}((Mc_{\text{single}}n + k + d)N \log Q)$.
- rdMPHE: Ciphertexts are encrypted once ($\mathcal{O}(k \cdot N \log Q)$). For the preparation phase, key aggregation involves only additions, and switching key generation takes $\mathcal{O}(dN \log PQ)$. Crucially, the key-switching in rdMPHE requires $\mathcal{O}(N \log N \log PQ)$ since the $\mathcal{O}(d)$ ModUp is performed once and amortized across $M$ sessions, yielding a total cost of $\mathcal{O}((Mc_{\text{single}} + k \log N + d)N \log Q)$.

*Communication Cost Analysis.* We assume a public key size of $\mathcal{O}(dN \log Q)$ and a standard ciphertext size of $\mathcal{O}(N \log Q)$.

- MPHE [MTBH21,MBH23]: Requires $\mathcal{O}(MkN \log Q)$ as ciphertexts are re-transmitted for each session. Over $M$ computations, the total cost is $\mathcal{O}((Mk + nd)N \log Q)$ over 3 rounds.

- MKHE [KKL+23]: Ciphertexts are transmitted once but are non-compact ($\mathcal{O}(n)$). The total cost is $\mathcal{O}((k + Mn + nd)N \log Q)$, achieved in 1 round.
- rdMPHE: Ciphertexts are transmitted once and remain compact ($\mathcal{O}(k)$). Reusability is enabled by transmitting switching keys ($\mathcal{O}(MndN \log PQ)$), resulting in a total cost of $\mathcal{O}((k + Mdn)N \log Q)$ over 3 rounds.

Table 3: Asymptotic analysis for the ciphertext reusing scenario across $M$ computations. The common factor $N \log Q$ is omitted.

| | Reusables | Computation | Comm. | Rnd. | Dyn. |
|---|---|---|---|---|---|
| MPHE [MBH23] | $\mathsf{pk}_i$ | $\mathcal{O}(Mc_{\text{single}} + Mk)$ | $\mathcal{O}(Mk + n)$ | 3 | ✗ |
| MKHE [KKL+23] | $\mathsf{pk}_i, \mathsf{ct}_i$ | $\mathcal{O}(nMc_{\text{single}} + k)$ | $\mathcal{O}(k + Mn)$ | 1 | ✓ |
| rdMPHE (ours) | $\mathsf{pk}_i, \mathsf{ct}_i$ | $\mathcal{O}(Mc_{\text{single}} + k \log N)$ | $\mathcal{O}(k + Mn)$ | 3 | ✓ |

*Note on MGHE and Dynamic Stability.* For MGHE [KLSW24], complexity varies based on the ciphertext and group structures. In the worst case, complexities align with MKHE (replacing $n$ with group count $n_g$). In the best case, structured ciphertexts may reduce complexity to MPHE levels. However, rdMPHE provides a stable framework that uniquely bridges the lightweight nature of MPHE with the long-term flexibility of MKHE, supporting efficient dynamic updates without the prohibitive overhead of re-running setup phases or expanding ciphertext dimensions.

# 5 Experimental Results

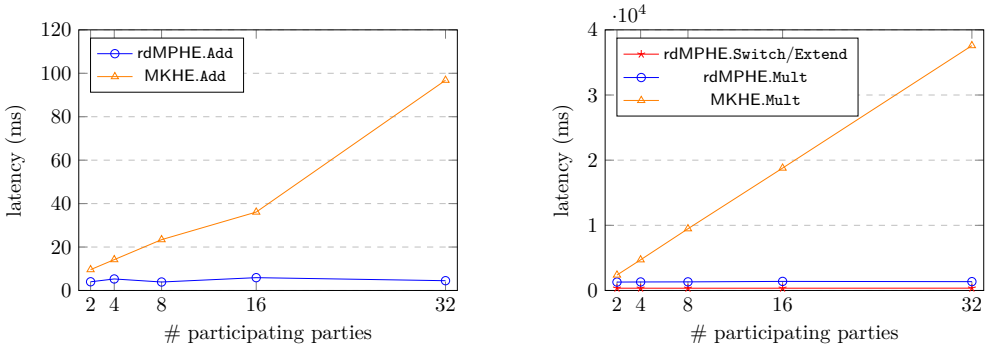
In this section, we present the implementation results of our dMPHE and rdMPHE schemes and compare them with prior solutions, such as MPHE and MKHE. We exclude MGHE from this comparison, as a fair apple-to-apple evaluation is difficult due to MGHE's heavy dependence on its predefined group structure. We implemented both the dMPHE and the rdMPHE schemes[5] based on the implementation of the CKKS-and B/FV-based MGHE scheme of [KLSW24, SNU24], built on Lattigo [Tun21], version 2.3.0.

For comparison, we modified their MKHE implementation by adding the joining functionality in an ad-hoc manner to evaluate dynamic capabilities. Furthermore, we introduce an example application of our rdMPHE scheme, the element-wise vector product, to demonstrate the utility of its dynamic property.

**Experimental Setup.** All our experiments are performed on an Intel Xeon Silver 4114 CPU at 2.2GHz with 256GB of RAM running Linux. Our benchmark is performed with the CKKS scheme [CKKS17] and B/FV scheme [Bra12, FV12]. We mainly use two sets of HE parameters of $(N, \lceil \log PQ \rceil \approx \lceil \log P \rceil + \lceil \log Q \rceil, d) = (2^{15}, 880 \approx 120 + 757, 14)$ and $(2^{14}, 439 \approx 120 + 313, 6)$, where $N$ is the ring dimension, $P$ and $Q$ are the auxiliary and the maximum ciphertext moduli, respectively, and $d$ is the gadget rank used for the key-switching keys.[6] We set the secret key distribution $\chi_{\mathsf{sk}}$ as a ternary distribution, sampling from $\{-1, 0, 1\}$ with probability 1/4, 1/2, and 1/4, respectively. The errors are sampled from a centered normal distribution with a standard deviation of 3.2. All parameter sets are chosen following the prior works [MTBH21, KLSW24], achieving 128 bits of RLWE hardness using the Lattice estimator [APS15]. All the numbers are averaged from 10 repetitions.

[5] Our code is available at https://github.com/YongdongYEO/rdMPHE.

[6] We denote the modulus size $\lceil \log PQ \rceil$ of our two parameter sets as 880 and 439, following [KLSW24], and note that our parameter choices exactly coincide with those used in their work.




**(a)** Addition. **(b)** Multiplication and key-switching.

Figure 2: Latency Comparison of MKHE [KLSW24] and rdMPHE. The latencies for (a) homomorphic addition, and (b) multiplication, `Switch`, and `Extend` of rdMPHE and multiplication of MKHE are given over the varying numbers of participating parties, based on a B/FV-based scheme with a parameter set ($N$=$2^{15}$, $\lceil \log PQ \rceil = 880$, $d = 14$).

## 5.1 Microbenchmark: Homomorphic Operations

In Figure 2, we compare the latencies of homomorphic operations of our rdMPHE and the MKHE construction of [KLSW24]. It shows the latency of each unit operation, in the number of participating parties.

To compare the latencies in the joining scenarios, as both schemes have dynamicity and reusability, we provide a detailed benchmark in Table 4. We measured the latency of `Add`, `Mult`, `Rot` for the number of parties $2, 4, 8, 16, 32$. 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 a single key-switching can perform.

We note that for each operation, the timings of rdMPHE should be almost the same as the timings of MPHE and dMPHE. All the evaluations (`Add`, `Mult`, `Rot`) and `Swi/Ext` (or `Switch/Extend`) are performed with ciphertexts at the top level.

Table 4: Benchmark result of `Add`, `Mult`, `Rot` between our Reusable Dynamic Multi-Party Homomorphic Encryption (rdMPHE) and Multi-Key Homomorphic Encryption (MKHE) from [KLSW24], for each party number $n = 2, 4, 8, 16, 32$. All the latencies are given in milliseconds.

| Type ($N$, $\lceil \log PQ \rceil$) | $n$ | 2 | | 4 | | 8 | | 16 | | 32 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | rdMPHE (ours) | MKHE [KLSW24] | rdMPHE (ours) | MKHE [KLSW24] | rdMPHE (ours) | MKHE [KLSW24] | rdMPHE (ours) | MKHE [KLSW24] | rdMPHE (ours) | MKHE [KLSW24] |
| B/FV ($2^{14}$, 439) | `Add` | 0.6 | 1.5 | 1.0 | 2.7 | 0.7 | 5.5 | 1.0 | 8.8 | 1.0 | 17.5 |
| | `Mult` | 150 | 275 | 150 | 539 | 151 | 1031 | 152 | 2112 | 163 | 4330 |
| | `Rot` | 42 | 83 | 44 | 170 | 43 | 335 | 43 | 677 | 44 | 1377 |
| CKKS ($2^{14}$, 439) | `Add` | 0.8 | 1.8 | 1.0 | 2.3 | 1.6 | 4.5 | 0.7 | 10.4 | 0.9 | 18.8 |
| | `Mult` | 69 | 129 | 70 | 248 | 71 | 504 | 73 | 1014 | 73 | 2016 |
| | `Rot` | 41 | 81 | 41 | 166 | 42 | 338 | 42 | 678 | 41 | 1342 |
| B/FV ($2^{15}$, 880) | `Add` | 4.0 | 9.6 | 5.3 | 14.2 | 3.9 | 23.4 | 5.9 | 36.1 | 4.5 | 96.8 |
| | `Mult` | 1289 | 2368 | 1307 | 4708 | 1320 | 9473 | 1393 | 18785 | 1353 | 37581 |
| | `Rot` | 373 | 721 | 369 | 1464 | 369 | 2954 | 370 | 5921 | 366 | 11624 |
| CKKS ($2^{15}$, 880) | `Add` | 5.6 | 9.0 | 4.2 | 12.8 | 4.2 | 25.6 | 5.1 | 41.9 | 4.0 | 90.6 |
| | `Mult` | 585 | 1097 | 577 | 2254 | 566 | 4078 | 561 | 8623 | 680 | 17643 |
| | `Rot` | 356 | 712 | 362 | 1486 | 354 | 2850 | 354 | 5882 | 376 | 11792 |

The overall benchmark results shown in Figure 2 and Table 4 demonstrate that our proposed dMPHE/rdMPHE consistently outperforms MKHE in all evaluated homomorphic computations. As analyzed in Section 4.4, dMPHE/rdMPHE maintains $O(1)$ computational complexity relative to the number of parties, while the complexity of MKHE increases linearly. Notably, the latencies for core operations such as `Rot`, `Switch`, and `Extend`

are dominated by a single-key key-switching (`KS`) operation, resulting in performance comparable to standard MPHE and dMPHE.

## 5.2 Application: Element-wise Vector Product

To evaluate the practical benefits of dMPHE and rdMPHE, we conducted an experiment with an element-wise vector product scenario where an initial group of 32 parties start homomorphic computations, then eventually, 32 additional parties join $(32 + 32)$.

Table 5: Profiled latency (in seconds) and error for element-wise vector product for 32 parties and extra 32 parties using MPHE/MKHE [KLSW24] and our dMPHE/rdMPHE. The latency and resulting error of the vector product evaluation before and after the additional parties join are denoted as (`Eval` 1, Error 1) and (`Eval` 2, Error 2), respectively. The errors are reported together with their standard deviations ($\pm$) from 10 repetitions.

| operations & errors | B/FV | | | | CKKS | | | |
|---|---|---|---|---|---|---|---|---|
| | MPHE [KLSW24] | dMPHE (ours) | rdMPHE (ours) | MKHE [KLSW24] | MPHE [KLSW24] | dMPHE (ours) | rdMPHE (ours) | MKHE [KLSW24] |
| `KeyGen` | - | 92 | 122 | 100 | - | 49 | 79 | 56 |
| `Switch` | - | - | 11 | - | - | - | 11 | - |
| `Eval` 1 | - | 41 | 41 | 97 | - | 20 | 20 | 42 |
| Error 1 | - | $208 \pm 3.4$ | $209 \pm 4.2$ | $195 \pm 1.3$ | - | $-30 \pm 0.1$ | $-27 \pm 0.7$ | $-33 \pm 0.1$ |
| `Extend` | $161^*$ | 70 | 98 | 67 | $97^*$ | 46 | 74 | 42 |
| `Eval` 2 | 81 | 42 | 42 | 134 | 40 | 21 | 20 | 52 |
| Error 2 | $248 \pm 5.8$ | $251 \pm 5.0$ | $250 \pm 5.0$ | $233 \pm 1.3$ | $-29.2 \pm 0.2$ | $-29 \pm 0.1$ | $-27 \pm 0.7$ | $-33 \pm 0.1$ |

Table 5 summarizes the latencies measured before and after the join, denoted as `Eval` 1 and `Eval` 2, respectively. For standard MPHE, which lacks dynamicity, results are reported in `Extend` with an asterisk ($^*$) and `Eval` 2 for direct comparison, as its setup can only begin after all 64 parties are fixed.

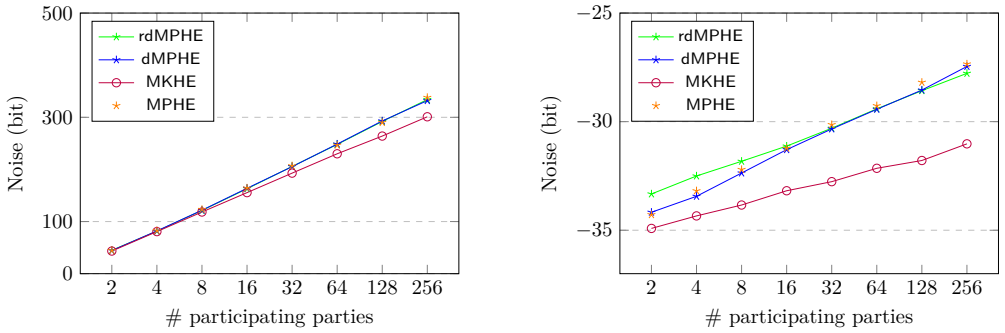
The results show that our schemes significantly outperform MKHE in computational efficiency as the number of parties increases. Unlike standard MPHE, which must remain idle until the final group is determined, dMPHE and rdMPHE provide superior flexibility by allowing immediate evaluation among the initial group (`Eval` 1). Furthermore, the overhead for scaling to 64 parties (`Eval` 2) remains minimal.

### 5.2.1 Error Analysis

In Table 5, we also summarize the noise measured before and after the join, denoted as Error 1 and Error 2, respectively. For B/FV, we tracked the $\ell_\infty$-norm of the noise polynomial, while for CKKS, we measured the error in the decoded message.

In Figure 3, we depict how noise accumulates in our vector product experiments. We successively doubled the number of participants, starting from 2 to $4 = 2 + 2$, then to $8 = 4 + 4$, and continuing in this manner up to $256 = 128 + 128$, recording the error after each evaluation.

The results confirm that while MPHE-style schemes (including ours) incur higher public-key noise compared to MKHE, the additional error introduced by our `Extend` and `Switch` procedures is negligible. As shown in Figure 3, the noise levels of dMPHE and rdMPHE closely track those of standard MPHE across both B/FV and CKKS. This demonstrates that our dynamic framework maintains the same magnitude of noise as the underlying MPHE, ensuring the reliability of homomorphic operations even as the group of participants scales. This confirms that while MPHE-based schemes generally exhibit higher noise growth than MKHE, the overhead from our dynamic extension and switching remains negligible, closely following the baseline MPHE noise budget.




**(a)** B/FV Scheme **(b)** CKKS Scheme

Figure 3: Comparison of error growths of element-wise vector product with MPHE, MKHE, and our rdMPHE/dMPHE. The number of parties starts with 2 and the number of parties increases with extra parties join ($2, 4, \cdots, 32$ at each step). The errors are measured after extra parties join and element-wise vector product with inputs of joined parties. The values are based on the parameter set ($N$=$2^{15}$, $\lceil \log PQ \rceil = 880$, $d = 14$).

### 5.2.2 Large-scale Benchmarks: 512 Parties

To further evaluate the scalability of our rdMPHE, we performed benchmarks with a significantly larger number of participating parties. Due to their extensive size, the full-scale benchmarks for the parameters set with $N = 2^{14}$, covering up to 512 parties, are provided in Tables 6 and 7. These results remain consistent with the efficiency and error trends discussed in the preceding sections.

## 6 Conclusion and Future Work

In this paper, we introduced dMPHE/rdMPHE, a versatile and compact multi-party homomorphic encryption framework that bridges the gap between the efficiency of MPHE and the flexibility of MKHE. By enabling dynamic party joining and ciphertext reusability without the prohibitive overhead of full setup re-runs or ciphertext expansion, our scheme provides a practical solution for large-scale, long-term cloud computations. While our construction achieves significant improvements in communication and computational costs, several directions remain for future exploration.

**Drop-out Case and Threshold Structure.** Throughout the paper, we assume that all participating parties in the access structure take part in the final partial decryption procedure. Such an $n$-out-of-$n$ setting implies that these parties could have decrypted the data regardless, which motivates the need for a threshold access structure (see Section 3.2).

In this case, however, the threshold access structure must also be updated during the Extend (joining) phase. For example, a $t$-out-of-$n$ structure should be updated to a $(t + k)$-out-of-$(n + k)$ structure when $k$ new parties join, in order to prevent the new parties (or any coalition of them with only $t - 1$ original parties) from decrypting on their own. Updating such structures requires coordinated efforts from both the original and the joining parties. For instance, share-resharing techniques (e.g., [MBH23]) require the participation of $t$ original parties together with the $k$ joining parties, typically assuming the availability of precomputed polynomials of degree $t + k$.

**Bootstrapping.** In our scheme, bootstrapping can be handled efficiently in an interactive manner following [MTBH21]. Although non-interactive solutions are available for

Table 6: Profiled latency (in seconds) and error for element-wise vector product using MPHE/MKHE [KLSW24] and our dMPHE/rdMPHE for up to $32 + 32$ parties. The latency and resulting error of the vector product evaluation before and after the additional parties join are denoted as (`Eval` 1, Error 1) and (`Eval` 2, Error 2), respectively. The errors are reported together with their standard deviations from 10 repetitions.

| #Parties | operations & errors | B/FV | | | | CKKS | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | MPHE [KLSW24] | dMPHE (ours) | rdMPHE (ours) | MKHE [KLSW24] | MPHE [KLSW24] | dMPHE (ours) | rdMPHE (ours) | MKHE [KLSW24] |
| 2+2 | `KeyGen` | - | 28 | 31 | 29 | - | 8 | 10 | 8 |
| | `Switch` | - | - | 1 | - | - | - | 1 | - |
| | `Eval` 1 | - | 1 | 1 | 1 | - | 1 | 1 | 1 |
| | Error 1 | - | 44 | 45 | 43 | - | -34 | -33 | -35 |
| | (std) | (-) | (1.0) | (1.3) | (0.5) | (-) | (0.1) | (0.7) | (0.2) |
| | `Extend` | 36 | 6 | 8 | 4 | 11 | 4 | 6 | 3 |
| | `Eval` 2 | 4 | 3 | 3 | 4 | 2 | 1 | 1 | 2 |
| | Error 2 | 83 | 82 | 83 | 80 | -33 | -33 | -32 | -35 |
| | (std) | (1.6) | (1.7) | (1.5) | (0.7) | (0.2) | (0.3) | (0.6) | (0.1) |
| 4+4 | `KeyGen` | - | 31 | 36 | 34 | - | 11 | 15 | 11 |
| | `Switch` | - | - | 1 | - | - | - | 1 | - |
| | `Eval` 1 | - | 4 | 4 | 5 | - | 2 | 2 | 3 |
| | Error 1 | - | 83 | 84 | 80 | - | -33 | -31 | -35 |
| | (std) | (-) | (2.1) | (2.4) | (0.7) | (-) | (0.2) | (0.7) | (0.1) |
| | `Extend` | 43 | 10 | 14 | 8 | 16 | 7 | 11 | 5 |
| | `Eval` 2 | 9 | 5 | 5 | 10 | 5 | 3 | 3 | 5 |
| | Error 2 | 122 | 121 | 122 | 117 | -32 | -32 | -31 | -34 |
| | (std) | (2.0) | (2.1) | (2.3) | (1.7) | (0.1) | (0.4) | (0.7) | (0.1) |
| 8+8 | `KeyGen` | - | 40 | 48 | 44 | - | 16 | 24 | 18 |
| | `Switch` | - | - | 3 | - | - | - | 3 | - |
| | `Eval` 1 | - | 9 | 9 | 15 | - | 5 | 5 | 7 |
| | Error 1 | - | 123 | 125 | 117 | - | -32 | -30 | -34 |
| | (std) | (-) | (3.1) | (1.7) | (1.9) | (-) | (0.1) | (1.2) | (0.1) |
| | `Extend` | 58 | 19 | 27 | 16 | 27 | 13 | 20 | 11 |
| | `Eval` 2 | 20 | 10 | 11 | 24 | 10 | 5 | 5 | 11 |
| | Error 2 | 162 | 164 | 164 | 155 | -31 | -31 | -30 | -34 |
| | (std) | (2.9) | (4.0) | (3.0) | (0.8) | (0.2) | (0.2) | (1.1) | (0.1) |
| 16+16 | `KeyGen` | - | 57 | 71 | 63 | - | 27 | 42 | 32 |
| | `Switch` | - | - | 5 | - | - | - | 5 | - |
| | `Eval` 1 | - | 20 | 20 | 39 | - | 10 | 10 | 19 |
| | Error 1 | - | 166 | 166 | 155 | - | -31 | -29 | -34 |
| | (std) | (-) | (3.3) | (2.5) | (1.8) | (-) | (0.1) | (0.9) | (0.1) |
| | `Extend` | 91 | 36 | 50 | 32 | 50 | 23 | 37 | 22 |
| | `Eval` 2 | 41 | 21 | 21 | 58 | 20 | 10 | 10 | 25 |
| | Error 2 | 207 | 207 | 207 | 195 | -30 | -30 | -28 | -33 |
| | (std) | (4.8) | (3.9) | (4.0) | (0.8) | (0.1) | (0.1) | (0.8) | (0.1) |
| 32+32 | `KeyGen` | - | 92 | 122 | 100 | - | 49 | 79 | 56 |
| | `Switch` | - | - | 11 | - | - | - | 11 | - |
| | `Eval` 1 | - | 41 | 41 | 97 | - | 20 | 20 | 42 |
| | Error 1 | - | 208 | 209 | 195 | - | -30 | -27 | -33 |
| | (std) | (-) | (3.4) | (4.2) | (1.3) | (-) | (0.1) | (0.7) | (0.1) |
| | `Extend` | 161 | 70 | 98 | 67 | 97 | 46 | 74 | 42 |
| | `Eval` 2 | 81 | 42 | 42 | 134 | 40 | 21 | 20 | 52 |
| | Error 2 | 248 | 251 | 250 | 233 | -29.2 | -29 | -27 | -33 |
| | (std) | (5.8) | (5.0) | (5.0) | (1.3) | (0.2) | (0.1) | (0.7) | (0.1) |

non-sparse or non-ternary secret keys (e.g., [BMTH21]), they may incur performance degradation compared to bootstrapping with sparse secrets. This limitation is inherent to all existing MPHE schemes unless a trusted third-party dealer is assumed. We also note that a recent work [MHP+25] investigates distributed key generation for sparse secrets; however, incorporating such techniques into our dynamic construction remains an interesting direction for future work.

Table 7: Profiled latency (in seconds) and error for element-wise vector product using MPHE/MKHE [KLSW24] and our dMPHE/rdMPHE for $64 + 64$ parties to $256 + 256$ parties. The latency and resulting error of the vector product evaluation before and after the additional parties join are denoted as (`Eval` 1, Error 1) and (`Eval` 2, Error 2), respectively. The values in the table are executed only once. For the experiment of $256 + 256$, MKHE did not finished due to the peak memory usage (denoted DNF).

| #Parties | operations & errors | B/FV | | | | CKKS | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | MPHE [KLSW24] | dMPHE (ours) | rdMPHE (ours) | MKHE [KLSW24] | MPHE [KLSW24] | dMPHE (ours) | rdMPHE (ours) | MKHE [KLSW24] |
| 64+64 | `KeyGen` | - | 175 | 235 | 183 | - | 95 | 166 | 104 |
| | `Switch` | - | - | 24 | - | - | - | 24 | - |
| | `Eval` 1 | - | 85 | 80 | 232 | - | 43 | 40 | 95 |
| | Error 1 | - | 245 | 253 | 234 | - | -29 | -24 | -33 |
| | `Extend` | 330 | 134 | 206 | 128 | 190 | 94 | 162 | 84 |
| | `Eval` 2 | 168 | 88 | 82 | 303 | 84 | 42 | 41 | 110 |
| | Error 2 | 286 | 290 | 298 | 274 | -28 | -29 | -24 | -32 |
| 128+128 | `KeyGen` | - | 300 | 469 | 340 | - | 180 | 336 | 208 |
| | `Switch` | - | - | 43 | - | - | - | 53 | - |
| | `Eval` 1 | - | 165 | 163 | 551 | - | 83 | 86 | 210 |
| | Error 1 | - | 292 | 284 | 274 | - | -28 | -24 | -32 |
| | `Extend` | 585 | 253 | 394 | 259 | 402 | 171 | 303 | 171 |
| | `Eval` 2 | 332 | 163 | 170 | 701 | 168 | 84 | 85 | 237 |
| | Error 2 | 338 | 340 | 333 | 314 | -27 | -27 | -24 | -32 |
| 256+256 | `KeyGen` | - | 584 | 860 | DNF | - | 387 | 630 | DNF |
| | `Switch` | - | - | 98 | DNF | - | - | 102 | DNF |
| | `Eval` 1 | - | 335 | 327 | DNF | - | 163 | 172 | DNF |
| | Error 1 | - | 324 | 336 | DNF | - | -27 | -22 | DNF |
| | `Extend` | 1151 | 509 | 816 | DNF | 756 | 342 | 640 | DNF |
| | `Eval` 2 | 669 | 348 | 341 | DNF | 336 | 163 | 161 | DNF |
| | Error 2 | 380 | 380 | 378 | DNF | -26 | -27 | -22 | DNF |

## Acknowledgments.

Part of this work was conducted while the second and third authors were at Seoul National University, and some results are included in the third author's Master's thesis at the same institution. The fourth author was financially supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (RS-2025-02217656). The authors thank Dr. Jai Hyun Park and Dr. Wonhee Cho for their insights and helpful discussions during the early stages of this work.

## References

[AAM17] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. Secure approximation of edit distance on genomic data. *BMC Medical Genomics*, 10:55–67, 2017. doi:10.1186/s12920-017-0279-9.

[AHWC21] Asma Aloufi, Peizhao Hu, Harry W. H. Wong, and Sherman S. M. Chow. Blindfolded evaluation of random forests with multi-key homomorphic encryption. *IEEE Transactions on Dependable and Secure Computing*, 18(4):1821–1835, 2021. doi:10.1109/TDSC.2019.2940020.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_29.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. doi:10.1515/jmc-2015-0016.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012. doi:10.1145/2090236.2090262.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 483–512. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96878-0_17.

[BMTH21] Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 587–617. Springer, Heidelberg, October 2021. doi:10.1007/978-3-030-77870-5_21.

[BOP06] M. Barni, C. Orlandi, and A. Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th Workshop on Multimedia and Security*, MM&Sec '06, pages 146–151, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1161366.1161393.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_8.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_50.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from TFHE. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 446–472. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-34621-8_16.

[CDL+21] Yuling Chen, Sen Dong, Tao Li, Yilei Wang, and Huiyu Zhou. Dynamic multi-key FHE in asymmetric key setting from LWE. *IEEE Transactions on Information Forensics and Security*, 16:5239–5249, 2021. doi:10.1109/TIFS.2021.3127023.

[CdWM+17] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Paper 2017/035, 2017. URL: https://eprint.iacr.org/2017/035.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*,

volume 10031 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53887-6_1.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 377–408. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70694-8_14.

[CHK+19] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J. Jacobson Jr:, editors, *SAC 2018*, volume 11349 of *LNCS*, pages 347–368. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-10970-7_16.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70694-8_15.

[CS16] Ana Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 325–340. Springer, Heidelberg, February / March 2016. doi:10.1007/978-3-319-29485-8_19.

[CSN+14] Mandy Chessell, Ferd Scheepers, Nhan Nguyen, Ruud van Kessel, and Ron van der Starre. Governing and managing big data for analytics and decision makers. *IBM Redguides for Business Leaders*, 252, 2014. URL: https://www.redbooks.ibm.com/abstracts/redp5120.html.

[CSS+22] Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. Efficient threshold FHE for privacy-preserving applications. Cryptology ePrint Archive, Paper 2022/1625, 2022. URL: https://eprint.iacr.org/2022/1625.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_24.

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

[GBDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 201–210, New York, USA, 20–22 Jun 2016. PMLR. URL: https://proceedings.mlr.press/v48/gilad-bachrach16.html.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. URL: crypto.stanford.edu/craig.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_49.

[HLSS25] Intak Hwang, Hyeonbum Lee, Jinyeong Seo, and Yongsoo Song. Practical zero-knowledge PIOP for maliciously secure multiparty homomorphic encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, pages 4049–4063, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3719027.3765229.

[HPS19] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 83–105. Springer, Heidelberg, March 2019. doi:10.1007/978-3-030-12612-4_5.

[JKLS18] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1209–1222. ACM Press, October 2018. doi:10.1145/3243734.3243837.

[KCSPS23] R. Hari Kishore, A. Chandra Sekhar, Pramoda Patro, and Debabrata Swain. 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)*, pages 1–6, 2023. doi:10.1109/ACCAI58221.2023.10200291.

[KG23] Dongwoo Kim and Cyril Guyot. Optimized privacy-preserving CNN inference with fully homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 18:2175–2187, 2023. doi:10.1109/TIFS.2023.3263631.

[KKL+23] Taechan Kim, Hyesun Kwak, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 726–740. ACM Press, November 2023. doi:10.1145/3576915.3623176.

[KLSW24] Hyesun Kwak, Dongwon Lee, Yongsoo Song, and Sameer Wagh. A general framework of homomorphic encryption for multiple parties with non-interactive key-aggregation. In *ACNS 2024*, pages 403–430, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-54773-7_16.

[KSK+18] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11:23–31, 2018. doi:10.1186/s12920-018-0401-7.

[LAHH22] Wei-Shan Lee, John A, Hsiu-Chun Hsu, and Pao-Ann Hsiung. SPChain: A smart and private blockchain-enabled framework for combining GDPR-compliant digital assets management with AI models. *IEEE Access*, 10:130424–130443, 2022. doi:10.1109/ACCESS.2022.3227969.

[LLK+23] Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. HETAL: efficient privacy-preserving transfer learning with homomorphic encryption. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023. URL: https://dl.acm.org/doi/10.5555/3618408.3619194.

[LLL+22] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 12403–12422. PMLR, 17–23 Jul 2022. URL: https://proceedings.mlr.press/v162/lee22e.html.

[LMSS22] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 560–589. Springer, Heidelberg, August 2022. doi:10.1007/978-3-031-15802-5_20.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012. doi:10.1145/2213977.2214086.

[LZC+24] Wenchao Liu, Tanping Zhou, Long Chen, Hongjian Yang, Jiang Han, and Xiaoyuan Yang. Round efficient privacy-preserving federated learning based on MKFHE. *Computer Standards & Interfaces*, 87, 2024. doi:10.1016/j.csi.2023.103773.

[MBH23] Christian Mouchet, Elliott Bertrand, and Jean-Pierre Hubaux. An efficient threshold access-structure for RLWE-based multiparty homomorphic encryption. *J. Cryptol.*, 36(2), 2023. doi:10.1007/s00145-023-09452-8.

[MHP+25] Seonhong Min, Guillaume Hanrot, Jai Hyun Park, Alain Passelègue, and Damien Stehlé. Distributed key generation for efficient threshold-CKKS. Cryptology ePrint Archive, Paper 2025/2057, 2025. URL: https://eprint.iacr.org/2025/2057.

[Mic25] Daniele Micciancio. Fully composable homomorphic encryption. *IACR Communications in Cryptology*, 2(1), 2025. doi:10.62056/ak5wl86bm.

[MTBH21] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *PoPETs*, 2021(4):291–311, October 2021. doi:10.2478/popets-2021-0071.

[NHY+25] Chao Niu, Zhicong Huang, Zhaomin Yang, Yi Chen, Liang Kong, Cheng Hong, and Tao Wei. XBOOT: Free-XOR gates for CKKS with applications to transciphering. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):118–144, Sep. 2025. URL: https://tches.iacr.org/index.php/TCHES/article/view/12406, doi:10.46586/tches.v2025.i4.118-144.

[Par21] Jeongeun Park. Homomorphic encryption for multiple users with less communications. *IEEE Access*, 9:135915–135926, 2021. doi:10.1109/ACCESS.2021.3117029.

[PR23] Jeongeun Park and Sergi Rovira. Efficient TFHE bootstrapping in the multiparty setting. *IEEE Access*, 11:118625–118638, 2023. doi:10.1109/ACCESS.2023.3326237.

[QHV16] Christoph Quix, Rihan Hai, and Ivan Vatov. GEMMS: A generic and extensible metadata management system for data lakes. In *CAiSE forum*, volume 129, 2016. URL: https://ceur-ws.org/Vol-1612/paper17.pdf.

[SNU24] SNUCP. SNU-MGHE. https://github.com/SNUCP/snu-mghe, January 2024. Commit: `295387e`.

[TSRC15] Ignacio G Terrizzano, Peter M Schwarz, Mary Roth, and John E Colino. Data Wrangling: The challenging journey from the wild to the lake. In *Conference on Innovative Data Systems Research*. Asilomar, 2015. URL: https://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper2.pdf.

[Tun21] Tune Insight. Lattigo v2.3.0. https://github.com/tuneinsight/lattigo, October 2021. Commit: `73d2269`.

[WA15] Coral Walker and Hassan Alrehamy. Personal data lake with data gravity pull. In *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, pages 160–167, 2015. doi:10.1109/BDCloud.2015.62.

[XLG+23] Guowen Xu, Guanlin Li, Shangwei Guo, Tianwei Zhang, and Hongwei Li. Secure decentralized image classification with multiparty homomorphic encryption. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(7):3185–3198, 2023. doi:10.1109/TCSVT.2023.3234278.

[YY14] Jiawei Yuan and Shucheng Yu. Privacy preserving back-propagation neural network learning made practical with cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):212–221, 2014. doi:10.1109/TPDS.2013.18.