Secure Multi-Key Homomorphic Encryption with Application to Privacy-Preserving Federated Learning

Jiahui Wu, Tiecheng Sun, *Member, IEEE,* Fucai Luo, Haiyan Wang, Weizhe Zhang, *Senior Member, IEEE*

***Abstract*—Multi-Key Homomorphic Encryption (MKHE), proposed by López-Alt et al. (STOC 2012), allows for performing arithmetic computations directly on ciphertexts encrypted under distinct keys. Subsequent works by Chen and Dai et al. (CCS 2019) and Kim and Song et al. (CCS 2023) extended this concept by proposing multi-key BFV/CKKS variants, referred to as the CDKS scheme. These variants incorporate asymptotically optimal techniques to facilitate secure computation across multiple data providers. In this paper, we identify a critical security vulnerability in the CDKS scheme when applied to multiparty secure computation tasks, such as privacy-preserving federated learning (PPFL). In particular, we show that CDKS may inadvertently leak plaintext information from one party to others. To mitigate this issue, we propose a new scheme, SMHE (Secure Multi-Key Homomorphic Encryption), which incorporates a novel masking mechanism into the multi-key BFV and CKKS frameworks to ensure that plaintexts remain confidential throughout the computation. We implement a PPFL application using SMHE and demonstrate that it provides significantly improved security with only a modest overhead in homomorphic evaluation. For instance, our PPFL model based on multi-key CKKS incurs less than a $2\times$ runtime and communication traffic increase compared to the CDKS-based PPFL model. The code is publicly available at https://github.com/JiahuiWu2022/SMHE.git.**

***Index Terms*—Multi-key homomorphic encryption, masking scheme, privacy protection, federated learning.**

I. Introduction

Homomorphic encryption (HE) is a cryptographic technique that enables computations to be performed directly on encrypted data, eliminating the need for decryption during the process. This property allows for the secure processing of sensitive data while preserving its confidentiality. For decades, constructing a fully homomorphic encryption (FHE) scheme, capable of supporting arbitrary computations on ciphertexts, remained an open problem until Gentry's groundbreaking work [1]. Since then, significant advancements have been achieved in the field, leading to various HE schemes such as BFV [2], [3], GSW [4], BGV [5], TFHE [6], and CKKS [7]. These schemes have expanded the practicality and efficiency of HE, making it a vital tool for secure computations in modern applications. One of the key features of HE is its ability to enable secure computation "on-the-fly," meaning that data owners are not required to be actively involved during the computation process. Instead, the evaluation can be carried out entirely by a public server. This makes HE particularly appealing for scenarios such as cloud computing, where data is processed remotely, and privacy preservation is of paramount importance.

In recent years, the demand for secure multiparty computation (MPC) protocols has surged, driven by applications such as federated learning [8]. MPC allows multiple parties to collaboratively evaluate a function or circuit on their private inputs without revealing any information beyond the final result. However, traditional single-key HE schemes are not well-suited for such multi-party settings. A major limitation arises when multiple data sources are involved, as standard HE schemes typically require all data to be encrypted under the same encryption key. This requirement grants the entity possessing the corresponding decryption key full access to the encrypted data, which raises issues regarding privacy and the reliance on centralized trust.

To address these limitations, researchers have extended the functionality of HE through approaches such as threshold homomorphic encryption (THE) [9]–[14] and multi-key homomorphic encryption (MKHE) [15]–[21]. THE allows decryption authority to be distributed among multiple parties, ensuring that no single entity possesses full access to the secret key. Similarly, MKHE enables computations on ciphertexts encrypted under different keys, thereby avoiding the need for a single shared encryption key and mitigating the risk of authority concentration. These extensions not only overcome the limitations of single-key HE but also integrate seamlessly with secure MPC protocols, preserving the inherent advantages of HE. Recognizing their potential, the National Institute of Standards and Technology (NIST) has highlighted these primitives as promising candidates for standardization in its recent call for multi-party threshold cryptographic schemes [22]. As a result, THE and MKHE have emerged as essential building blocks for advancing privacy-preserving technologies in collaborative and distributed environments.

In THE, multiple parties collaboratively generate a common public key, with the corresponding secret key shared among them through a secret-sharing mechanism. Although THE schemes generally achieve performance levels comparable to single-key HE and are typically more efficient than multi-key HE schemes, they suffer from a critical limitation: reliance on

This work is supported by The Key Program of the Joint Fund of the National Natural Science Foundation of China (Grant No. U22A2036).

Jiahui Wu, Tiecheng Sun, Haiyan Wang, and Weizhe Zhang are with the Department of New Networks, Pengcheng Laboratory, Shenzhen 518000, China; Fucai Luo is with the School of Computer Science and Technology, Zhejiang Gongshang University, Hangzhou, China (e-mail: wujh01@pcl.ac.cn/jiahuiwu2022@163.com; tiechengsun@126.com; lfucai@126.com; wanghy01@pcl.ac.cn; wzzhang@hit.edu.cn.); Corresponding author: Weizhe Zhang.

a static key access structure. In other words, all participants must be predetermined and fixed during the initial setup phase. In contrast, this work focuses on MKHE, which offers significant advantages in terms of flexibility and reduced interaction requirements. Specifically, MKHE schemes allow each participant to independently generate their own public-secret key pairs without needing knowledge of other parties' keys. This property enables operations on ciphertexts encrypted under different keys and allows computations to be performed in a public cloud environment without establishing a common public key. Moreover, recent advances in MKHE have introduced *full dynamism*, allowing computations to be performed on multi-key ciphertexts without predefined circuits. Arbitrary circuits can be evaluated in real time, and new participants or ciphertexts can be incorporated into ongoing evaluations at any stage. This flexibility facilitates the construction of SMC protocols on top of MKHE, leveraging its dynamic and adaptive nature [17]. The ability to seamlessly integrate new participants and ciphertexts on-the-fly is a key advantage, making MKHE particularly well-suited for applications such as federated learning, where the set of participants may change throughout the computation process.

While MKHE provides a flexible and dynamic framework, designing secure and efficient MKHE schemes is significantly more challenging than for other homomorphic encryption variants, due to stringent functional and security requirements. Since the pioneering work of López-Alt et al. [15], who introduced the first MKHE scheme based on NTRU, considerable efforts have been made to extend traditional (single-key) HE schemes into their multi-key counterparts [16]–[21], [23], [24]. Nonetheless, achieving both strong security guarantees and practical efficiency in MKHE remains a substantial challenge. Early MKHE constructions often suffered from high computational and communication overhead, limiting their practicality in real-world deployments. Recent advancements [19]–[21] have proposed improved designs with significantly enhanced asymptotic and concrete efficiency, representing the current state-of-the-art in MKHE. However, despite these improvements, previously overlooked security vulnerabilities have emerged, making these MKHE schemes potentially less secure than their single-key HE counterparts.

A. Challenges and Contributions

Recent MKHE schemes, particularly the CDKS constructions [20], [21], have significantly improved practicality. However, they exhibit security vulnerabilities in real-world multiparty computation scenarios. To address this, we revisit the security assumptions of CDKS-style constructions and identify several critical challenges. Our main contributions are summarized below.

Challenge 1: Inadequate Security Guarantees in CDKS for MPC. The CDKS schemes [20], [21] lack adequate security guarantees in MPC settings. In these schemes, ciphertexts from different parties are expanded into a unified format embedding all parties' identity information for joint evaluation. Our analysis reveals that this expansion and evaluation process may inadvertently leak plaintext data. A detailed examination of this vulnerability, rooted in the ciphertext expansion mechanism, is provided in Section IV-B.

Our Solution: We propose a secure multi-key HE framework named SMHE. Our approach introduces a new ciphertext expansion and evaluation method by leveraging masking primitives, originally developed in GSW-based MKHE [16], [17], and we adapt them to the polynomial-based structures of BFV and CKKS.

At a high level, a masking refers to an encryption of a random value that is added to a fresh ciphertext. Since the masking is itself a semantically secure ciphertext, it reveals no information about the random value it encodes. This encrypted randomness serves as a one-time pad to conceal the participant's plaintext during ciphertext expansion and intermediate homomorphic evaluation. This effectively mitigates the attack we later demonstrate against CDKS, in which adversaries exploit partial decryption results to infer individual plaintexts.

The core challenge lies in efficiently constructing such encrypted masks that can be correctly removed during final decryption to preserve result correctness, without undermining the inherent dynamism of the MKHE framework. In GSW-based MKHE schemes [16], [17], the masking scheme allows each party P to produce auxiliary information U during encryption. This information, published together with the ciphertext C, leaks no information about the underlying plaintext and is used to support joint computation. When a second participant P' joins with public key pk', the auxiliary information U and pk' can be used to construct a masking matrix X satisfying

$$\begin{aligned}(sk, sk')\begin{pmatrix} C & X \\ 0 & C \end{pmatrix} &= (skC, skX + sk'C) \\ \approx (\mu \cdot sk\Theta, \mu \cdot sk'\Theta) &= \mu(sk, sk')\Theta,\end{aligned} \tag{1}$$

where sk and sk' are the secret keys of the two parties, μ is the plaintext of the party P, and Θ is a fixed public matrix. Define the concatenated secret key $\overline{sk} = (sk, sk')$ and the expanded ciphertext $\overline{C} = \begin{pmatrix} C & X \\ 0 & C \end{pmatrix}$. Then, using GSW decryption, the plaintext is recovered as $\mu\overline{sk}\Theta \cdot \Theta^{-1}W \approx \mu$, where W is a fixed GSW decryption vector. Similarly, for party P', with auxiliary information U' and ciphertext C', the expansion yields: $\overline{sk}\begin{pmatrix} C' & 0 \\ X' & C' \end{pmatrix} = (skC' + sk'X', sk'C') \approx \mu' \cdot \overline{sk}\Theta$, where X' is constructed from U' and pk, and μ' is the plaintext of P'. The homomorphic addition of the two parties' expanded ciphertexts is given by: $\overline{C}+\overline{C}' = \begin{pmatrix} C+C' & X \\ X' & C+C' \end{pmatrix}$, and its decryption yields:

$$\begin{aligned}&\overline{sk}\begin{pmatrix} C+C' & X \\ X' & C+C' \end{pmatrix} \cdot \Theta^{-1}W \\ &= \big(sk(C+C') + sk'X', skX + sk'(C+C')\big) \cdot \Theta^{-1}W \\ &\approx (\mu+\mu')\overline{sk}\Theta \cdot \Theta^{-1}W = \mu + \mu'.\end{aligned}$$

This ensures the correctness of homomorphic addition. Homomorphic multiplication can be handled similarly, as shown in [16], [17]. By using this construction, ciphertexts encrypted under different keys are transformed into a unified expansion

format, enabling efficient homomorphic operations in a dynamic multi-key setting. This allows each party to encrypt data independently while still enabling collaborative computation, without predetermining key structures or decryption.

Challenge 2: While the masking scheme proposed in [16], [17] effectively ensures the security and the correctness of GSW-based MKHE schemes, it cannot be directly applied to CKKS or BFV schemes for the following reasons:

R1: GSW ciphertexts are represented as matrices $C \in \mathbb{Z}_Q^{n\times m}$, whereas CKKS/BFV ciphertexts are polynomial pairs $ct = (c_0, c_1) \in R_Q^2$. This structural difference makes it challenging to construct a multi-key CKKS/BFV joint decryption equation of the form

$$\langle sk, (ct + cx)\rangle + \langle sk', ct\rangle \approx \mu \cdot \theta,$$

(where cx is a masking with the same form as ct constructed by a auxiliary information we denote as Γ, and θ is a constant term), analogous to the GSW multi-key decryption equation $\big(sk(C + X) + sk'C\big) \cdot \Theta^{-1}W \approx \mu \cdot \overline{sk} \cdot W = \mu$.

R2: The GSW masking scheme requires large extended ciphertexts for correct demasking, with each single-bit encryption resulting in a ciphertext of size $O(N^2)$, quadratic in the number of parties. This leads to significant computational and communication overhead. Therefore, it is necessary to design a new masking scheme for CKKS/BFV that ensures the expanded ciphertext size scales linearly with N, thereby improving both computational and space efficiency.

Our Solution: To address the challenges outlined in R1 and R2, we propose a novel masking scheme specifically designed for CKKS/BFV. To ensure the correctness of the joint decryption of the masked ciphertext (implicitly implying the unmasking operation followed by decryption), we add an additional "demasking" ciphertext to the masked ciphertext. Specifically, we construct a masking ciphertext $cx = (x_0, x_1)$ and a "demasking" ciphertext $cz = (z_0, z_1)$, where cx contains a random mask r. The condition $\langle sk, cx\rangle + \langle sk', cz\rangle \approx 0$ must hold, and the correct joint decryption is then given by:

$$\begin{aligned}&\langle sk, cx\rangle + \langle sk', cz\rangle + \langle sk, ct\rangle \approx \mu\\ \Rightarrow\ &\langle (1, s, s'), (z_0 + x_0 + c_0, x_1 + c_1, z_1)\rangle \triangleq \langle \overline{sk}, \overline{ct}\rangle \approx \mu\end{aligned}$$

where $\overline{sk} = (1, s, s')$ is the concatenated secret key and $\overline{ct} = (z_0 + x_0 + c_0, x_1 + c_1, z_1) \triangleq (\bar{c}_0, \bar{c}_1, \bar{c}_2)$ is the expanded ciphertext. Notably, the size of our expanded ciphertext $\overline{ct}$ is linear with the number of participants. The remaining task is to construct the equation

$$\langle sk, cx\rangle + \langle sk', cz\rangle \approx 0. \tag{2}$$

Taking the CKKS scheme as an example (which is similar to the BFV scheme), encryption of a plaintext μ is expressed as $ct \leftarrow \texttt{Encrypt}(pk, \mu) = w \cdot pk + (\mu + e_0, e_1) \pmod Q$, where w is a random polynomial, $sk = (1, s)$ is the secret key, $pk = (b = -as + e, a)$ is the public key, a is a polynomial in a given ring R_Q, and e, e_0, e_1 represent small errors. The construction of Eq. (2) is then described as follows.

Let cz be a ciphertext representing zero. It then follows that:

$$\langle sk', cz\rangle = \langle (1, s'), r\cdot pk + (e_0, e_1)\rangle = r(b - b') + e_z \pmod Q,$$

where r is a random polynomial, $b' = -as' + e'$, and $cz \leftarrow \texttt{Encrypt}(pk, 0) = r \cdot pk + (e_{r_0}, e_{r_1}) \pmod Q$ represents the encryption of the plaintext zero, with e', e_{r_0}, e_{r_1} being small errors. The masking is performed using the random polynomial r, which is encrypted as auxiliary information $\Gamma \leftarrow \texttt{Encrypt}(pk, r) = w\cdot pk + (r + e_{x_0}, e_{x_1})$, where e_{x_0}, e_{x_1} are small errors. The evaluator then sets $cx = \Gamma(b' - b)$ to make $\langle sk, cx\rangle + \langle sk', cz\rangle \approx r(b' - b) + r(b - b') = 0$.

Challenge 3: In the decryption process, the evaluator is expected to compute a correction term $cx = \Gamma(b' - b)$ such that $\langle sk, cx\rangle + \langle sk', cz\rangle \approx 0$. However, since $b' - b \in R_Q$ is essentially a random polynomial, we have:

$$\begin{aligned}&\langle sk, cx\rangle = \langle sk, (b' - b)(w \cdot pk + (r + e_{x_0}, e_{x_1}))\rangle\\ &= (b' - b)\big((w \cdot b + r + e_{x_0}) + (w \cdot a + e_{x_1})s\big)\\ &= (b' - b)(r + we + e_{x_0} + e_{x_1}s) = r(b' - b) + E \pmod Q\end{aligned}$$

where $E = (b' - b)(we + e_{x_0} + e_{x_1}s) \in R_Q$ is a large and non-negligible noise term. As a result, the desired equation $\langle sk, cx\rangle + \langle sk', cz\rangle \approx 0$ fails to hold due to this accumulated noise, rendering the decryption equation invalid.

Solution: To control the noise growth, we apply the gadget decomposition technique to reduce the growth of E, ensuring that $\langle sk, cx\rangle + \langle sk', cz\rangle \approx 0$. For a detailed explanation of the gadget decomposition technique, the reader is referred to Section III-C, and for a comprehensive description of our masking scheme, Section V-A provides further details.

II. Related Work

MKHE enables computations on ciphertexts encrypted under distinct keys. López-Alt and Wichs [15] first introduced an MKHE scheme based on the NTRU cryptosystem, whose security relies on a relatively non-standard assumption concerning polynomial rings. This assumption differs from the more widely used Learning With Errors (LWE) assumption [25] or its ring-based variant [26], and it currently lacks a worst-case hardness theorem [18] to support its security. Subsequent work by Clear et al. [16] proposed an LWE-based MKHE scheme that employs a multikey variant of the GSW scheme and ciphertext extension techniques. Mukherjee et al. [17] later simplified this approach. Both of these schemes design masking scheme for the GSW cryptosystem. Specifically, the masking system enables evaluators to generate a specific mask by combining a universal mask and a target participant identity. This mask facilitates the "encoding" of ciphertexts from different identities into a larger matrix, which allows for joint homomorphic computations on these ciphertexts. Mukherjee et al. also proposed a 2-round (plain) MPC protocol in the common random string (CRS) model for secure distributed decryption. Peikert and Shiehian [18] further extended the multi-key GSW scheme to develop two multi-hop MKHEs. However, all these GSW-based MKHE variants face a major limitation: they can only encrypt a single bit in a large expanded GSW ciphertext, which results in substantial space and time complexities as the bit-length of the ciphertext grows quadratically with N^2, where N is the number of distinct participants. Brakerski et al. [23] proposed an MKHE scheme based on LWE that employs short ciphertexts and

TABLE I: The symbols and their corresponding interpretations.

Symbols	Interpretations
n / N	The number of the clients / The RLWE dimension
$\mathcal{H}$ / $\lfloor \cdot \rceil$	Gadget decomposition function / The rounding function
$\mathbf{g}$, τ	Gadget vector and its dimension
λ	Security parameter
$\mathbb{Z}_Q$	$\mathbb{Z} \cap (-Q/2, Q/2]$
$\langle u, v \rangle$	The inner product of two tuples/vectors u, v
$\|\|a\|\|_\infty$	The ℓ^∞-norm of the coefficient vector of a
χ / $U(\cdot)$	χ distributions over R / Uniform distribution
D_σ	Discrete Gaussian distribution with standard deviation σ
$a \leftarrow A$	a is sampled from a set or distribution A
ct / $\overline{ct}$ / $\widehat{ct}$	Fresh ciphertext/Expanded ciphertext/Masked ciphertext
c_j^i / $\overline{c}_j^i$	The jth component of the ciphertext ct_i / $\overline{ct}_i$

a quasi-linear expansion rate. However, both the asymptotic and concrete efficiency of this scheme have not been clearly understood, making its practical applicability uncertain.

A follow-up study by Chen et al. [19] introduced a multi-key TFHE and presented the implementation results. Meanwhile, another line of research [20], [21] has focused on designing multi-key variants of batch HEs, such as BFV [2], [3] and CKKS [7]. However, these constructions exhibit significant security vulnerabilities and thus applications such as privacy-preserving federated learning [27] and secure distributed sparse Gaussian processes [28] that utilize the multi-key CKKS scheme fail to meet their stated security objectives.

In this paper, we propose new multi-key variants of batch HE schemes, including CKKS and BFV, by introducing a novel masking scheme for the CKKS/BFV cryptosystem. Our scheme supports the encryption of both floating-point and integer values with multiple bits, without requiring the large ciphertext expansion as in [16]–[18]. Furthermore, our scheme addresses the security vulnerabilities present in existing multi-key CKKS/BFV constructions [20], [21].

III. Preliminaries

In this section, we introduce the background knowledge of ring learning with errors problem, multi-key homomorphic encryption, and gadget decomposition. Table I summarizes mathematical notations used in this paper.

A. Ring Learning with Errors

The Ring Learning With Errors (RLWE) assumption is based on polynomial arithmetic with coefficients in a finite field. Specifically, let N be a power of two. Define $R = \mathbb{Z}[x]/(x^N + 1)$ and $R_Q = \mathbb{Z}_Q[x]/(x^N + 1)$. Let χ be a probability distribution over R, and $\sigma > 0$ be a real number. The RLWE assumption, defined by the parameters (N, Q, χ, σ), asserts that it is computationally infeasible to distinguish between two scenarios: given polynomially many samples of either $(a, b) \in R_Q^2$ or $(a, a \cdot s + e) \in R_Q^2$, where s is drawn from χ and e is sampled from the discrete Gaussian distribution D_σ with mean 0 and standard deviation σ, the distribution of the two cases remains indistinguishable. The RLWE assumption serves as the foundation for homomorphic encryption schemes like BFV and CKKS.

B. Multi-Key Homomorphic Encryption

Multi-Key homomorphic encryption (MKHE) is an cryptographic scheme that extends the capabilities of traditional single-key HE to support computations on ciphertexts encrypted using distinct keys. It contains the following algorithms:

- $pp \leftarrow \texttt{MKHE.Setup}(1^\lambda)$. Generates a set of public parameters pp based on a given security parameter λ.
- $(sk, pk) \leftarrow \texttt{MKHE.KeyGen}(pp)$. Generates a secret-public key pair (sk, pk) using the public parameters pp.
- $ct \leftarrow \texttt{MKHE.Encrypt}(\mu, pk)$. Encrypts the plaintext μ with the public key pk and outputs the fresh ciphertext ct.
- $\overline{ct} \leftarrow \texttt{MKHE.Expand}\big(\{pk_1, \cdots, pk_n\}, i, ct\big)$. Expands the given ciphertext ct, encrypted under pk_i, into an expanded ciphertext $\overline{ct}$ associated with n public keys $\{pk_1, \cdots, pk_n\}$.
- $\overline{ct} \leftarrow \texttt{MKHE.Eval}(\mathcal{C}, \{\overline{ct}_1, \cdots, \overline{ct}_k\}, \{pk_{id}\}_{id \in T})$. Performs homomorphic evaluation on a circuit $\mathcal{C}$ using the ciphertexts $\{\overline{ct}_1, \cdots, \overline{ct}_k\}$, and outputs the ciphertext $\overline{ct}$ associated with the joint public key set $\{pk_{id}\}_{id \in T}$ that corresponds to all input ciphertexts involved.
- $\mu := \texttt{MKHE.Decrypt}(\overline{ct}; \{sk_{id}\}_{id \in T})$. Recovers the plaintext μ using the corresponding private keys $\{sk_{id}\}_{id \in T}$ of all public keys referenced in the given ciphertext $\overline{ct}$.

A unique feature of MKHE is its use of "reference sets" $\{pk_{id}\}_{id \in T}$. Each ciphertext maintains a reference to the public keys under which the data has been encrypted. Initially, a fresh ciphertext is tied to a single key. As homomorphic computations progress and involve ciphertexts encrypted under additional keys, the reference set expands. Decryption requires all secret keys corresponding to the keys in the reference set. Specifically, in collaborative scenarios, each participant partially decrypts the ciphertext with his secret key and broadcasts the partial decryption result. Then, the plaintext can be constructed by combining all participants' partial decryption results. This distributed decryption is as follows:

- $\nu_{id} := \texttt{MKHE.PartDec}(\overline{ct}, sk_{id})$. Partially decrypts the ciphertext $\overline{ct}$ with the secret key sk_{id} and returns the partial decryption result ν_{id}.
- $\mu := \texttt{MKHE.FullDec}(\overline{ct}, \{\nu_{id}\}_{id \in T})$. Fully decrypts to obtain the plaintext μ by combining the partial decryption results $\{\nu_{id}\}_{id \in T}$ corresponding to all ciphertexts associated with the public keys referenced in the given ciphertext $\overline{ct}$.

The correctness and security of MKHE is defined as follows.

Definition 1 (Correctness)**.** *Let $ct_1, \cdots, ct_n$ be MKHE ciphertexts encrypting messages $\mu_1, \cdots, \mu_n$, respectively. Denote by $\{pk_i\}_{i \in [1,n]}$ the combined public key set associated with these ciphertexts. Suppose $ct \leftarrow \texttt{MKHE.Eval}(\mathcal{C}, ct_1, \cdots, ct_n; \{pk_i\}_{i \in [1,n]})$ is the result of evaluating a circuit $\mathcal{C}$ over these ciphertexts. Then, decryption using the corresponding secret keys $\{sk_i\}_{i \in [1,n]}$ correctly recovers the result with overwhelming probability:*

$$\texttt{MKHE.Dec}(ct; \{sk_i\}_{i \in [1,n]}) = \mathcal{C}(\mu_1, \cdots, \mu_n).$$

For approximate encryption schemes such as CKKS, this correctness notion is relaxed to allow for small errors, resulting

in the approximate correctness condition:

$$\texttt{MKHE.Dec}(ct; \{sk_i\}_{i\in[1,n]}) \approx \mathcal{C}(\mu_1, \cdots, \mu_n).$$

Definition 2 (Simulation-Based Security)**.** *An MKHE scheme is secure if it is* simulation-based secure *that any real-world adversary interacting with the system cannot learn more than what is revealed by an ideal functionality.*

Formally, let λ be the security parameter. Consider a set of users $\mathcal{I}$ ($|\mathcal{I}| \geq 2$, i.e., multi-party setting), each holding a message μ_i and a public/secret key pair $(pk_i, sk_i) \leftarrow \texttt{MKHE.KeyGen}(pp)$, where $pp \leftarrow \texttt{MKHE.Setup}(1^\lambda)$. The parties perform the following:

- *Message encryption: $ct_i \leftarrow \texttt{MKHE.Encrypt}(pk_i, \mu_i)$;*
- *Ciphertext expansion: $\overline{ct} \leftarrow \texttt{MKHE.Expand}(\{pk_i\}_{i\in\mathcal{I}}, i, ct_i)$;*
- *Evaluation: $\overline{ct} \leftarrow \texttt{MKHE.Eval}(\mathcal{C}, \{ct_i\}_{i\in\mathcal{I}}; \{pk_i\}_{i\in\mathcal{I}})$;*
- *Decryption: $\nu_i := \texttt{MKHE.PartDec}(\overline{ct}, sk_i)$ and $\mu := \texttt{MKHE.FullDec}(\overline{ct}, \{\nu_i\}_{i\in\mathcal{I}})$.*

Let $\mathcal{A}$ be a real-world adversary who observes all ciphertexts, public keys, partial decryption, and the evaluation result. Then there exists a probabilistic polynomial-time simulator Sim *such that the following distributions are computationally indistinguishable:*

- ***Real-world view:** The adversary's view in the real protocol execution, including the public parameters pp, public keys $\{pk_i\}$, input ciphertexts $\{ct_i, \overline{ct}_i\}$, the evaluated ciphertext $\overline{ct}$, the partial decryption $\{\nu_i\}$, and optionally the final output μ.*
- ***Ideal-world simulation:** The simulated view produced by* Sim*, given only the public parameters pp and the final output $\mu = \mathcal{C}(\mu_1, \ldots, \mu_n)$.*

We say the scheme is secure if: $\text{View}^{real}_{\mathcal{A}}(\lambda) \approx_c \text{View}^{ideal}_{\mathsf{Sim}}(\lambda)$, i.e., no efficient adversary can distinguish between the real view and the ideal simulation with non-negligible probability.

C. Gadget Decomposition

In lattice-based HE schemes, the accumulation of noise during homomorphic operations poses a significant challenge to the efficiency and correctness of computations. One widely used technique for mitigating this noise growth is *Gadget Decomposition*. By leveraging structured representations, gadget decomposition not only reduces the complexity of operations but also provides a mechanism for controlling noise in ciphertexts, enabling efficient and accurate homomorphic computations. Informally, gadget decomposition is designed to represent elements in a ring as compact, structured linear combinations of predefined basis elements.

Definition 3 (Gadget Decomposition)**.** *Let Q and τ be a modulus and a positive integer, respectively. A gadget decomposition is defined as a mapping $\mathcal{H} : R_Q \to R^\tau$ that meets the conditions below for all $b \in R_Q$:*

- ***Reconstruction property:** A constant vector $\boldsymbol{g} = (g_0, g_1, \cdots, g_{\tau-1}) \in R_Q^\tau$ exists such that $\langle \mathcal{H}(b), \boldsymbol{g} \rangle \equiv b \pmod{Q}$.*
- ***Bounded coefficients:** The coefficients of $\mathcal{H}(b)$ are small, i.e., $\|\mathcal{H}(b)\|_\infty \leq B_\mathcal{H}$ for some constant $B_\mathcal{H} > 0$.*

The vector $\boldsymbol{g}$ is referred to as the gadget vector, while $\mathcal{H}(b)$ is a compact representation of b with bounded coefficients. The mapping $\mathcal{H}$ can be viewed as a right inverse of the inner product operation $\mathcal{G}(\boldsymbol{u}) = \langle \boldsymbol{g}, \boldsymbol{u} \rangle \pmod{Q}$, ensuring that $\mathcal{G}(\mathcal{H}(b)) = b$.

Definition 4 (Gadget Encryption)**.** *For a given message $\mu \in R$ and a secret key $sk = (1, s)$ with $s \in R$, a pair $\boldsymbol{\Gamma} = (\boldsymbol{\varsigma}_0, \boldsymbol{\varsigma}_1) \leftarrow \texttt{GgtEnc}(sk, \mu) \in R_Q^{\tau\times 2}$ is defined as a gadget encryption of the message if its decryption meets $\langle sk, \boldsymbol{\Gamma} \rangle \approx \mu \cdot \boldsymbol{g} \pmod{Q}$.*

Definition 5 (External Product)**.** *Define $b \boxdot \boldsymbol{\varsigma} = \langle \mathcal{H}(b), \boldsymbol{\varsigma} \rangle \pmod{Q}$ as the external product of b and $\boldsymbol{\varsigma}$, where $b \in R_Q$ and $\boldsymbol{\varsigma} \in R_Q^\tau$. Additionally, for $\boldsymbol{\Gamma} = (\boldsymbol{\varsigma}_0, \boldsymbol{\varsigma}_1) \in R_Q^{\tau\times 2}$, we define $b \boxdot \boldsymbol{\Gamma} = (b \boxdot \boldsymbol{\varsigma}_0, b \boxdot \boldsymbol{\varsigma}_1)$.*

By employing the gadget decomposition technique, it becomes feasible to homomorphically perform multiplication on arbitrary ring elements while avoiding the generation of excessive noise. Specifically, let $\boldsymbol{\Gamma} = (\boldsymbol{\varsigma}_0, \boldsymbol{\varsigma}_1) \in R_Q^{\tau\times 2}$ represent a gadget encryption of $\mu \in R$ using sk, satisfying $\langle sk, \boldsymbol{\Gamma} \rangle = \mu \cdot \boldsymbol{g} + \boldsymbol{e} \pmod{Q}$ for a small $\boldsymbol{e} \in R^\tau$. The external product $(c_0, c_1) \leftarrow b \boxdot \boldsymbol{\Gamma}$ then meets:

$$\begin{aligned} \langle sk, (c_0, c_1) \rangle &= \langle sk, (b \boxdot \boldsymbol{\varsigma}_0, b \boxdot \boldsymbol{\varsigma}_1) \rangle = \langle \mathcal{H}(b), \langle sk, \boldsymbol{\Gamma} \rangle \rangle \\ &= \langle \mathcal{H}(b), \mu \cdot \boldsymbol{g} + \boldsymbol{e} \rangle = b \cdot \mu + e \pmod{Q}, \end{aligned} \tag{3}$$

where the noise term $e = \langle \mathcal{H}(b), \boldsymbol{e} \rangle \in R$ remains small. Consequently, (c_0, c_1) can be viewed as a noisy encryption of $b \cdot \mu$, as intended.

IV. Overview of Prior Work

This section reviews the most relevant researches proposed by Chen and Dai et al. [20] and Kim and Song et al. [21]. The former designs multi-key BFV/CKKS variants, while the latter focuses on enhancing computational efficiency and mitigating noise growth in these schemes. Collectively, we refer to these schemes as CDKS. We begin by outlining the core construction of CDKS and then highlight a critical security vulnerability within the framework.

Algorithm 1 CDKS Relinearization

Input: $ct_{\text{mult}} = (c_{i,j})_{i,j\in[0,n]} \in R_Q^{(n+1)\times(n+1)}$, $\{evk_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)\}_{i\in[1,n]}$
Return: $\overline{ct}_{\text{mult}} = (\bar{c}_i)_{i\in[0,n]} \in R_Q^{n+1}$

$\bar{c}_0 \leftarrow c_0 c'_0$
for $i = 1$ to n **do**
  $\bar{c}_i \leftarrow c_0 c'_i + c_i c'_0 \pmod{Q}$
for $i = 1$ to n **do**
  for $j = 1$ to n **do**
    $\bar{c}_j \leftarrow \bar{c}_j + c_i c'_j \boxdot \mathbf{d}_i \pmod{Q}$
    $\bar{c}_i \leftarrow \bar{c}_i + c_i c'_j \boxdot \mathbf{b}_j \pmod{Q}$
    $(\bar{c}_0, \bar{c}_i) \leftarrow (\bar{c}_0, \bar{c}_i) + c_i c'_j \boxdot (\mathbf{v}_i, \mathbf{u}_i) \pmod{Q}$

A. Foundational Construction of CDKS

CDKS is built on the CRS model, where all key holders share access to identical public random polynomials. Specifically, tt contains the following algorithms. Components specific to the multi-key CKKS and multi-key BFV schemes are highlighted in blue and red, respectively.

- $pp \leftarrow \texttt{CDKS.Setup}(1^\lambda)$: Let N denote the RLWE dimension, $Q = \prod_{i=0}^{L} q_i$ represent the ciphertext modulus for some integers q_i, the plaintext modulus $t \in \mathbb{Z}$, the key distribution χ over R, and the error distribution D_σ with σ be positive value. $\mathbf{a} \leftarrow U(R_Q^k)$ is a CRS. and the public parameter is defined as $pp = (N, t, Q, \chi, \sigma, \mathbf{a})$.
- $(sk, pk, evk) \leftarrow \texttt{CDKS.KeyGen}(pp)$: Generates secret and public keys $(sk = (1, s), pk = (b, a))$ and an evaluation key $evk = (\mathbf{b}, \mathbf{d}, \mathbf{u}, \mathbf{v})$: Sample $s, \gamma \leftarrow \chi$, $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2 \leftarrow D_\sigma^\tau$, $\mathbf{u} \leftarrow U(R_Q^\tau)$ and compute $\mathbf{b} = -s \cdot \mathbf{a} + \mathbf{e}_0 \pmod Q$, $\mathbf{d} = -\gamma \cdot \mathbf{a} + s \cdot \mathbf{g} + \mathbf{e}_1 \pmod Q$, $\mathbf{v} = -s \cdot \mathbf{u} - \gamma \cdot \mathbf{g} + \mathbf{e}_2 \pmod Q$. Set $b = \mathbf{b}[0]$ and $a = \mathbf{a}[0]$ which are the first components of $\mathbf{b}$ and $\mathbf{a}$, respectively. The subscripts is used to identify keys associated with distinct key holders.
- $ct \leftarrow \texttt{CDKS.Encrypt}(\mu, pk)$: Samples $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Encrypts the given plaintext $\mu \in R$ and outputs the ciphertext $ct = w \cdot pk + (\mu + e_0, e_1) \pmod Q$. (Encrypts the given plaintext $\mu \in R_t$ and returns the ciphertext $ct = w \cdot pk + (\lfloor (Q/t) \cdot \mu \rceil + e_0, e_1) \pmod Q$).
- $\overline{ct} := \texttt{CDKS.Expand}(\{pk_1, \cdots, pk_n\}; i; ct)$: Expands the given ciphertext ct, encrypted under pk_i, into a expanded ciphertext $\overline{ct}$ associated with n public keys $\{pk_1, \cdots, pk_n\}$. Specifically, a ciphertext $ct = (c_0, c_1)$ is expanded into the ciphertext $\overline{ct} = (\bar{c}_0, \cdots, \bar{c}_n) \in R_Q^{n+1}$, where $\bar{c}_0 = c_0, \bar{c}_i = c_1$, and the remaining $\{\bar{c}_j\}_{j \in [1,n], j \neq \{0,i\}}$ are all set to 0.
- $\overline{ct}_{\text{add}} \leftarrow \texttt{CDKS.Add}(\overline{ct}, \overline{ct}')$: For the given ciphertexts $\overline{ct}, \overline{ct}' \in R_Q^{n+1}$, the addition is performed as $\overline{ct}_{\text{add}} = \overline{ct} + \overline{ct}' \pmod Q$.
- $\overline{ct}_{\text{mult}} \leftarrow \texttt{CDKS.Mult}(\overline{ct}, \overline{ct}'; \{evk_i\}_{i \in [1,n]})$: For the given ciphertexts $\overline{ct} = (c_i)_{i \in [0,n]}, \overline{ct}' = (c_i')_{i \in [0,n]} \in R_Q^{n+1}$ and their associated evaluation keys $\{evk_i\}_{i \in [1,n]}$, the multiplication is performed as $ct_{\text{mult}} = (c_i c_j')_{i,j \in [0,n]} \pmod Q$ ($ct_{\text{mult}} = (\lfloor (t/Q) c_i \cdot c_j' \rceil)_{i,j \in [0,n]} \pmod Q$). Run Algorithm 1 with $(ct_{\text{mul}}, \{evk_i\}_{i \in [1,n]})$ and output the result $\overline{ct}_{\text{mult}}$.
- $\mu \leftarrow \texttt{CDKS.Decrypt}(\overline{ct}, \overline{sk})$: Decrypts a given ciphertext $\overline{ct} = (\bar{c}_0, \bar{c}_1, \cdots, \bar{c}_n)$ using $\overline{sk} = (1, s_1, \cdots, s_n)$ and outputs $\mu = \langle \overline{sk}, \overline{ct} \rangle \pmod Q$. Return μ (Return $\mu = \lfloor (t/Q) \cdot \mu \rceil$).

In CDKS, the distributed decryption process is as follows:

- $\nu_i \leftarrow \texttt{CDKS.PartDec}(\bar{c}_i, s_i)$: Partially decrypts the given ciphertext component $\bar{c}_i$ using s_i and outputs $\nu_i = \bar{c}_i s_i + e_i \pmod Q$, where $e_i \leftarrow D_\sigma$.
- $\mu := \texttt{CDKS.Merge}(\bar{c}_0, \{\nu_i\}_{i \in [1,n]})$: Merges the results of partial decryption to compute $\mu = \bar{c}_0 + \sum_{i=1}^n \nu_i \pmod Q$. Return μ (Return $\mu = \lfloor (t/Q) \cdot \mu \rceil$).

The distributed decryption process is correct since

$$\mu = \bar{c}_0 + \sum_{i=1}^{n} \nu_i = \langle \overline{sk}, \overline{ct} \rangle + \sum_{i=1}^{n} e_i \approx \langle \overline{sk}, \overline{ct} \rangle \pmod Q.$$

B. Security Vulnerability of CDKS

The CDKS scheme fails to provide the level of security it claims. As a result, existing MPC applications built upon CDKS, such as secure federated learning (FL) [27] and secure distributed sparse gaussian process [28], inherit CDKS's inherent vulnerabilities and consequently fail to achieve their intended security guarantees. This limitation arises from a fundamental flaw in CDKS that allows either the server or the clients to recover plaintexts. To illustrate this issue, consider an FL system with n clients and an aggregation server (as depicted in Fig. 1) secured using CDKS. The core process unfolds as follows:

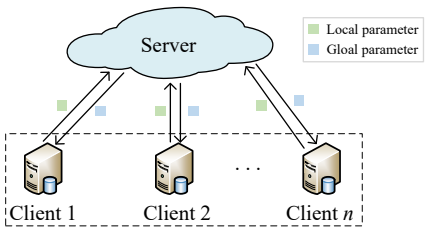


Fig. 1: FL system model.

- **Key generation:** Each client i generates its secret-public key pair (sk_i, pk_i).
- **Encryption:** Each client i trains its local model and encrypts its local paremeter μ_i into a fresh ciphertext $ct_i = (c_0^i, c_1^i)$. Then, it sends ct_i to the server.
- **Secure aggregation of local updates:** Upon receiving $ct_i = (c_0^i, c_1^i)$, the server applies $\texttt{CDKS.Expand}$ to expand the ciphertext into $\overline{ct}_i = (c_0^i, 0, \cdots, 0, c_1^i, 0, \cdots, 0)$, where c_0^i and c_1^i are replaced in the first and $(i+1)$th positions, respectively, while the remaining positions are padded with zeros. Then, by using $\texttt{CDKS.Add}$, the server aggregates all expanded ciphertexts $\{\overline{ct}_i\}_{i \in [1,n]}$ into the global ciphertext

$$\overline{ct} = \Big(\sum_{i=1}^{n} c_0^i, c_1^1, c_1^2, \cdots, c_1^n\Big) \triangleq (\bar{c}_0, \bar{c}_1, \bar{c}_2, \cdots, \bar{c}_n),$$

  which is then sent to all clients for decryption.
- **Decryption to obtain a global update**: Each client i first computes its partial decryption result $\nu_i \leftarrow \texttt{CDKS.PartDec}(\bar{c}_i, s_i) = c_1^i s_i + e_i \pmod Q (e_i \leftarrow D_\sigma)$, and broadcasts ν_i to the other clients or sends it to the aggregation server. Then, any client or the server computes the final decryption result as the global update:

$$\mu := \texttt{CDKS.Merge}(\bar{c}_0, \{\nu_i\}_{i \in [1,n]}) = \bar{c}_0 + \sum_{i=1}^{n} \nu_i \quad (4)$$
$$= \sum_{i=1}^{n} (c_0^i + c_1^i \cdot s_i) + \sum_{i=1}^{n} e_i \approx \sum_{i=1}^{n} \mu_i \pmod Q$$

Although the above process ensures correct parameter aggregation, it exposes local client parameters even under the honest-but-curious assumption, where both the server and the clients are assumed to faithfully execute the FL protocol while being curious to infer the privacy data of other clients based on the information it observes during the execution of the protocol. This security vulnerability stems from the ciphertext expansion and distributed decryption processes. Specifically, each component $\bar{c}_i$ ($i \in [1, n]$) of the expanded ciphertext $\overline{ct} = (\bar{c}_0, \bar{c}_1, \bar{c}_2, \cdots, \bar{c}_n)$ is associated with only the fresh ciphertext c_1^i of client i. This design facilitates decryption correctness: during partial decryption, each client i simply uses its secret key s_i to decrypts its corresponding component $\bar{c}_i$, obtaining

the partial decryption result $\nu_i = c_1^i \cdot s_i + e_i \pmod{Q}$. In the full decryption phase, the server (or all clients) merges the shared values as $\bar{c}_0 + \sum_{i=1}^n \nu_i \approx \sum_{i=1}^n \mu_i \pmod{Q}$, since each individual decryption step satisfies $c_0^i + \nu_i \approx \mu_i$, and the first component $\bar{c}_0 = \sum_{i=1}^n c_0^i$. Thus, the correctness of the full decryption is ensured. However, this simple mechanism, while guaranteeing correct decryption, introduces a critical security vulnerability. Since c_0^i, as part of the fresh ciphertext $ct_i = (c_0^i, c_1^i)$, is public, and the partial decryption result ν_i is shared to either the server or the clients for full decryption, any participant with access to ν_i can directly recover the client i's plaintext via $c_0^i + \nu_i \approx \mu_i \pmod{Q}$.

This compromise demonstrates that CDKS fails to provide adequate security guarantees in MPC scenarios. To address the security limitations of CDKS, in this work, we propose secure MKHE schemes, named SMHE, to achieve secure multi-key variants of CKKS and BFV.

V. New Multi-Key Variants of CKKS and BFV

To achieve our secure multi-key homomorphic encryption (SMHE), we retain the foundational structure of CDKS but introduce a masking scheme. This masking scheme ensures both the security and correctness of homomorphic addition, thereby enabling secure aggregation applications such as FL. Furthermore, ciphertexts processed through homomorphic addition with the masking scheme can still be directly evaluated by the homomorphic multiplication operations of CDKS without additional requirements or auxiliary computations.

A. A Masking Scheme for CKKS and BFV

We design a masking scheme tailored for CKKS and BFV, highlighting its role as a critical component in MKHE framework. Essentially, a masking scheme enables the use of a CKKS/BFV public key pk (with an associated secret key $sk = (1, s)$) and a plaintext μ to generate a triple $(ct, cz, \mathbf{\Gamma})$. Here, $ct = (c_0, c_1)$ represents CKKS/BFV encryption of μ under pk, while $cz = (z_0, z_1)$ and $\mathbf{\Gamma} = (\varsigma_0, \varsigma_1)$ act as auxiliary information with two key properties: (1) the triple $(ct, cz, \mathbf{\Gamma})$ provides computational hiding for μ, similar to ct alone, and (2) when provided with another CKKS/BFV public key pk' (corresponding to a secret key $sk' = (1, s')$), it becomes feasible to construct a pair of polynomials $cx = (x_0, x_1) \in R_Q^2$ according to $\mathbf{\Gamma}$, such that $\langle sk, cx\rangle + \langle sk', cz\rangle \approx 0$ and $\langle \overline{sk}, ct\rangle + \langle sk, cx\rangle + \langle sk', cz\rangle \approx \mu$. The latter also implies that $\langle (1, s, s'), (c_0 + x_0 + z_0, c_1 + x_1, z_1)\rangle = \langle \overline{sk}, \overline{ct}\rangle \approx \mu$. Here, $\overline{sk} = (1, s, s')$ is the decryption key and $\overline{ct} = (c_0 + x_0 + z_0, c_1 + x_1, z_1)$ is a masked ciphertext.

CKKS/BFV Masking Scheme. The masking scheme contains a triple of algorithms defined as follows:

- $\texttt{UniEnc}(\mu, pk)$: Given a message $\mu \in R_Q$ and a CKKS/BFV public key pk, returns a ciphertext ct.
- $\texttt{MaskEnc}(r, pk)$: Given a random masking r and a public pk, the masking encryption algorithm outputs a pair $(cz, \mathbf{\Gamma})$.
- $\texttt{Extend}(\mathbf{\Gamma}, pk, pk')$: Provided with $\mathbf{\Gamma}$ and two CKKS/BFV public keys pk, pk', outputs $cx \in R_Q^2$.

The masking scheme meets the following properties:

Semantic Security: For a given security parameter λ, the security of CKKS/BFV encryption ensures that:

$$(pp, pk, \texttt{UniEnc}(\mu, pk)) \overset{\text{comp}}{\approx} (pp, pk, \texttt{UniEnc}(\mu', pk)),$$
$$(pp, pk, \texttt{MaskEnc}(r, pk)) \overset{\text{comp}}{\approx} (pp, pk, \texttt{MaskEnc}(r', pk)),$$

where $pp \leftarrow \texttt{FHE.Setup}(1^\lambda)$, $(sk, pk) \leftarrow \texttt{FHE.Keygen}(pp)$, $r' \leftarrow \chi$, and $\mu' \leftarrow R_Q$. FHE represents the traditional single-key CKKS/BFV scheme.

Correctness: Let $pp \leftarrow \texttt{FHE.Setup}(1^\lambda)$, and consider two independently generated key pairs (sk, pk) and (sk', pk'), obtained from $\texttt{FHE.Keygen}(pp)$. For any $\mu \in R_Q$, let $ct \leftarrow \texttt{UniEnc}(\mu, pk)$, $(cz, \mathbf{\Gamma}) \leftarrow \texttt{MaskEnc}(r, pk)$, and $cx \leftarrow \texttt{Extend}(\mathbf{\Gamma}, pk, pk')$. Then $\mu := \texttt{FHE.Decrypt}(sk, ct)$ and

$$\langle sk, ct\rangle + \langle sk, cx\rangle + \langle sk', cz\rangle = \mu + e,$$

where $\|e\|_\infty \le (2N^2 + 4N)B_\chi + \tau N \cdot B_\chi B_{\mathcal{H}}$.

Instantiation. We now instantiate our masking scheme.

- $\texttt{UniEnc}(\mu, pk)$: Given a plaintext $\mu \in R_Q$ and a public key $pk = (b, a) \in R_Q^2$, sample $w \leftarrow \chi$ and $e_{w_0}, e_{w_1} \leftarrow D_\sigma$. Encrypts μ with pk using CKKS/BFV encryption algorithm to output a fresh ciphertext ct:

$$\begin{aligned} ct &\leftarrow \texttt{FHE.Enc}(pk, \mu) \\ &=_{\text{CKKS}} w \cdot pk + (\mu + e_{w_0}, e_{w_1}) \pmod{Q}. \\ &=_{\text{BFV}} w \cdot pk + (\lfloor (Q/t) \cdot \mu \rceil + e_{w_0}, e_{w_1}) \pmod{Q}. \end{aligned} \tag{5}$$

- $\texttt{MaskEnc}(r, pk)$: Given a random masking $r \in R_Q$, this algorithm outputs the masking ciphertexts $(cz, \mathbf{\Gamma})$:
  1) Sample $e_{r_0}, e_{r_1} \leftarrow D_\sigma$.
  2) Encrypt the value 0 with r and pk using CKKS/BFV encryption algorithm and output

$$\begin{aligned} cz &\leftarrow \texttt{FHE.Encrypt}(pk, 0; r) \\ &= r \cdot pk + (e_{r_0}, e_{r_1}) \pmod{Q}. \end{aligned} \tag{6}$$

  3) Perform gadget encryption on r and output $\mathbf{\Gamma} \leftarrow \texttt{GgtEnc}(sk, r)$, which meets $\langle sk, \mathbf{\Gamma}\rangle \approx r \cdot \mathbf{g} \pmod{Q}$.

- $\texttt{Extend}(\mathbf{\Gamma}, pk, pk')$: On input $\mathbf{\Gamma} \in R_Q^{\tau \times 2}$ and public keys $pk = (b, a) \in R_Q^2, pk' = (b', a) \in R_Q^2$, the algorithm outputs $cx = (b' - b) \boxdot \mathbf{\Gamma}$, such that $\langle sk, cx\rangle \approx r(b' - b)$.

- $\texttt{Extend}^*(\mathbf{\Gamma}, pk, \{pk_i\}_{i\in[1,n]})$: This algorithm takes $\mathbf{\Gamma} \in R_Q^{\tau \times 2}$ and the public keys pk and $\{pk_i\}_{i\in[1,n]}$ as input, computes $\sum_{i\in[1,n]}(b_i - b) = \sum_{i\in[1,n]} b_i - nb$, and outputs $cx = \left(\sum_{i\in[1,n]}(b_i - b)\right) \boxdot \mathbf{\Gamma}$, such that $\langle sk, cx\rangle \approx r \cdot \sum_{i\in[1,n]}(b_i - b)$.

The $\texttt{Extend}^*$ algorithm simplifies the process of running the $\texttt{Extend}$ algorithm for all n public key $\{pk_i\}_{i\in[1,n]}$. Rather than executing $\texttt{Extend}(\mathbf{\Gamma}, pk, pk_1) + \cdots + \texttt{Extend}(\mathbf{\Gamma}, pk, pk_n) = \sum_{i\in[1,n]}(b_i - b) \boxdot \mathbf{\Gamma}$, it consolidates these computations into a single execution of $\texttt{Extend}^*(\mathbf{\Gamma}, pk, \{pk_i\}_{i\in[1,n]})$.

Semantic Security. The attacker's view consists of the distribution $(pp, pk, ct, cz, \mathbf{\Gamma})$, where $pp \leftarrow \texttt{SMHE.Setup}(1^\lambda)$, $(sk, pk) \leftarrow \texttt{SMHE.Keygen}(pp)$, $ct \leftarrow \texttt{UniEnc}(\mu, pk)$, and $(cz, \mathbf{\Gamma}) \leftarrow \texttt{MaskEnc}(r, pk)$, where $\mu \in R_Q$.

The semantic security proof leverages the security of

CKKS/BFV encryption and follows these steps: (1) Modify $\mathbf{\Gamma}$ to the gadget encryption of 0 instead of the gadget encryption of r. This is justified by the semantic security of CKKS/BFV encryption. (2) Replace ct and cz with encryptions of random messages. Similarly, this step relies on the security of CKKS/BFV encryption, as the random sample w is unknown and the random sample r is no longer accessible after the first modification. As a result, the distribution becomes independent of μ, establishing semantic security.

Correctness. Let $\{(sk, pk), (sk', pk')\}$ represent two valid key pairs produced by $\texttt{SMHE.Keygen}(pp)$. Recall that $sk = (1, s), sk' = (1, s'); s, s' \leftarrow \chi; , pk = (b, a) \in R_q^2, pk' = (b', a) \in R_q^2$ with $b = -a \cdot s + e \pmod{Q}, b' = -a \cdot s' + e' \pmod{Q}$, and $\|e\|_\infty, \|e'\|_\infty \leq \beta_\chi$.

The masking ciphertext with public key pk is $(cz, \mathbf{\Gamma}) \leftarrow \texttt{MaskEnc}(r, pk)$. For a message μ, let $ct \leftarrow \texttt{UniEnc}(\mu, pk)$ and $cx \leftarrow \texttt{Extend}(\mathbf{\Gamma}, pk, pk')$. We have

$$\begin{aligned}\langle sk, ct\rangle &= \langle (1, s), (wb + \lfloor (Q/t)\cdot\mu \rceil + e_{w_0}, wa + e_{w_1})\rangle \qquad (7)\\ &= \lfloor (Q/t)\cdot\mu \rceil + we + e_{w_0} + se_{w_1} = \lfloor (Q/t)\cdot\mu \rceil + e_c,\\ \langle sk', cz\rangle &= \langle (1, s'), (rb + e_{r_0}, ra + e_{r_1})\rangle \qquad (8)\\ &= rb + e_{r_0} + ras' + s'e_{r_1} = r(b - b') + e'_c,\end{aligned}$$

where $e_c = we + e_{w_0} + se_{w_1}$ and $e'_c = re' + e_{r_0} + s'e_{r_1}$ with $\|e_c\|_\infty, \|e'_c\|_\infty \leq (N^2 + 2N)B_\chi$. Due to the correctness of linear combinations, we can also deduce that

$$\begin{aligned}\langle sk, cx\rangle &= \langle sk, (b' - b) \boxdot (\varsigma_0, \varsigma_1)\rangle\\ &= \langle \mathcal{H}(b' - b), \langle sk, (\varsigma_0, \varsigma_1)\rangle\rangle = r(b' - b) + e_r,\end{aligned} \qquad (9)$$

where $e_r = \langle \mathcal{H}(b' - b), \mathbf{e}\rangle$, $\|e_r\|_\infty \leq \tau N \cdot B_\chi B_\mathcal{H}$. Combining these results, we can obtain that $\langle sk, cx\rangle + \langle sk', cz\rangle \approx 0$ and $\langle sk, ct\rangle + \langle sk, cx\rangle + \langle sk', cz\rangle = \lfloor (Q/t)\cdot\mu \rceil + e^* \approx \lfloor (Q/t)\cdot\mu \rceil$, where $\|e^*\|_\infty \leq (2N^2 + 4N)B_\chi + \tau N \cdot B_\chi B_\mathcal{H}$, as required.

B. SMHE Construction

We construct our SMHE based on the proposed masking scheme, which contains the following algorithms:

- $pp \leftarrow \texttt{SMHE.Setup}(1^\lambda) \rightarrow pp$: Takes as input a security parameter λ and outputs the system parameters $pp = \{N, t, Q, \chi\, \sigma, \mathbf{a}, \mathcal{H}, \mathbf{g}\}$, where $N = N(\lambda)$ is the RLWE dimension; $t \in \mathbb{Z}$ is the plaintext modulus; $Q = \sum_{i=1}^{L} q_i$ is the ciphertext modulus; χ and D_σ are the key distribution and error distribution, respectively; $\mathbf{a} \leftarrow U(R_q^d)$ is a CRS; $\mathcal{H} : R_Q \rightarrow R^\tau$ and $\mathbf{g} \in R_Q^\tau$ are a gadget decomposition and its gadget vector, respctively.
- $(sk, pk, evk) \leftarrow \texttt{SMHE.KeyGen}(pp)$: Generates a secret-public key pair $(sk = (1, s), pk = (b, a))$ and an evaluation key $evk = (\mathbf{b}, \mathbf{d}, \mathbf{u}, \mathbf{v})$ following the CDKS construction.
- $C \leftarrow \texttt{SMHE.Encrypt}(\mu, pk)$: This algorithm encrypts the given plaintext $\mu \in R$ ($\mu \in R_t$) into the ciphertext ct and generates masking ciphertexts $\{cx, \mathbf{\Gamma}\}$ under the encryption key pk, producing a tuple $C = \{ct, cx, \mathbf{\Gamma}\}$. The encryption and masking components are produced using the functions $\texttt{UniEnc}(\mu, pk)$ and $\texttt{MaskEnc}(r, pk)$, respectively. Specifically, the algorithm samples $w, r \leftarrow \chi$ and $e_{v0}, e_{v1}, e_{r0}, e_{r1} \leftarrow D_\sigma$ and computes $ct = w \cdot pk + (\mu + e_{v0}, e_{v1}) \pmod{Q}$, $cz = r \cdot pk + (e_{r0}, e_{r1}) \pmod{Q}$ ($ct = w \cdot pk + (\lfloor (Q/t) \cdot \mu \rceil + e_{v0}, e_{v1}) \pmod{Q}$, $cz = r \cdot pk + (\lfloor (Q/t) \cdot \mu \rceil + e_{r_0}, e_{r_1}) \pmod{Q}$), and $\mathbf{\Gamma} \leftarrow \texttt{GgtEnc}(sk, r)$.
- $\overline{ct} := \texttt{SMHE.Expand}(\{pk_j\}_{j\in[1,n]}, i, ct)$: The algorithm expands the given ciphertext $ct = (c_0, c_1)$, encrypted under the public key pk_i, into an expanded ciphertext $\overline{ct} = (\bar{c}_j)_{j\in[0,n]}$, where:

$$\bar{c}_0 = c_0 \quad \text{and} \quad \bar{c}_j = \begin{cases} c_1 & \text{if } j = i,\\ 0 & \text{otherwise,}\end{cases} \quad \text{for } j \in [1, n].$$

  We denote the associated key and the reference set of the expanded ciphertext $\overline{ct}$ as pk_i and $\{pk_j\}_{j\in[1,n]}$, respectively.
- $\overline{ct}_{\text{add}} \leftarrow \texttt{SMHE.Add}_2\big(\overline{ct}_1, \overline{ct}_2; pk_1, pk_2; \{cz_i,\ \mathbf{\Gamma}_i\}_{i\in[1,2]}\big)$: This algorithm takes as inputs two expanded ciphertexts $\overline{ct}_1 = (c_0^1, c_1^1, 0), \overline{ct}_2 = (c_0^2, 0, c_1^2) \in R_q^3$, which correspond to the fresh ciphertexts $ct_1 = (c_0^1, c_1^1), ct_2 = (c_0^2, c_1^2)$, encrypted under keys pk_1 and pk_2, respectively. The associated masking parameters are $\{cz_i, \mathbf{\Gamma}_i\}_{i\in[1,2]}$. The algorithm first performs $cx_1 = (x_0^1, x_1^1) \leftarrow \texttt{Extend}(\mathbf{\Gamma}_1, pk_1, pk_2), cx_2 = (x_0^2, x_1^2) \leftarrow \texttt{Extend}(\mathbf{\Gamma}_2, pk_2, pk_1)$. Then, it masks the two ciphertexts as $\widehat{ct}_1 = (c_0^1 + x_0^1 + z_0^2, c_1^1, z_0^1), \widehat{ct}_2 = (c_0^2 + x_0^2 + z_0^1, z_0^2, c_1^2)$[1]. Finally, the algorithm outputs $\overline{ct}_{\text{add}} = \widehat{ct}_1 + \widehat{ct}_2 \pmod{Q}$.
- $\overline{ct}_{\text{add}} \leftarrow \texttt{SMHE.Add}\big(\overline{ct}, \overline{ct}'; \{pk_j\}_{j\in T}, \{pk'_j\}_{j\in T'}; \{cz_k, \mathbf{\Gamma}_k\}_{k\in T\cup T'}\big)$: This algorithm is a general case of homomorphic addition. It takes as inputs two ciphertexts $\overline{ct} = (c_i)_{i\in[0,n]}, \overline{ct}' = (c'_i)_{i\in[0,n]} \in R_q^{n+1}$, the corresponding reference sets $\{pk_j\}_{j\in T}$ and $\{pk'_j\}_{j\in T'}$, and the involved masking parameters $\{cz_k, \mathbf{\Gamma}_k\}_{k\in T\cup T'}$. It invokes Algorithm 2 on the inputs and outputs the homomorphic addition result $\overline{ct}_{\text{add}} = (\bar{c}_i)_{i\in[0,n]}$.
- $\overline{ct}_{\text{mult}} \leftarrow \texttt{SMHE.Mult}(\overline{ct}, \overline{ct}'; \{evk_i\}_{i\in[1,n]})$: For two given ciphertexts $\overline{ct} = (c_i)_{i\in[0,n]}, \overline{ct}' = (c'_i)_{i\in[0,n]} \in R_Q^{n+1}$ and their associated public keys $\{pk_i\}_{i\in[1,n]}$, the multiplication is performed as $\overline{ct}_{\text{mult}} = (c_i c'_j)_{i,j\in[0,n]} \pmod{Q}$ ($\overline{ct}_{\text{mult}} = (\lfloor (t/Q) c_i \cdot c'_j \rceil)_{i,j\in[0,n]} \pmod{Q}$). Run Algorithm 1 with $(ct_{\text{mul}}, \{evk_i\}_{i\in[1,n]})$ and output the result $\overline{ct}_{\text{mult}}$.
- $\nu_i \leftarrow \texttt{SMHE.PartDec}(\bar{c}_i, s_i)$: Partially decrypts a given ciphertext component $\bar{c}_i$ using s_i and outputs $\nu_i = \bar{c}_i s_i + e_i \pmod{Q}$, where $e_i \leftarrow \chi$.
- $\mu := \texttt{SMHE.Merge}(\bar{c}_0, \{\nu_i\}_{i\in[1,n]})$: Compute $\mu = \bar{c}_0 + \sum_{i=1}^{n} \nu_i \pmod{Q}$. Return μ (Return $\mu = \lfloor (t/Q) \cdot \mu \rceil$).

1) Security of SMHE Construction: We now present the security analysis of the proposed SMHE scheme under the semi-honest adversarial model using a simulation-based proof.

Security Setting. We consider a semi-honest adversary $\mathcal{A}$ who follows the protocol honestly but may try to infer additional information from observed messages. To prove security, we follow the real/ideal world simulation paradigm

[1]The masked ciphertexts $\widehat{ct}_1, \widehat{ct}_2$ can be equivalently interpreted as being obtained via an expansion of the masked fresh ciphertexts. That is, although the actual implementation performs masking after ciphertext expansion, the resulting ciphertexts $\widehat{ct}_1, \widehat{ct}_2$ can be conceptually understood as the expansion of masked ciphertexts: $\widehat{ct}_1 = \texttt{SMHE.Expand}(\{pk_i\}_{i\in[1,2]}, 1, ct_1 + cx_1 + cz_2)$, $\widehat{ct}_2 = \texttt{SMHE.Expand}(\{pk_i\}_{i\in[1,2]}, 2, ct_2 + cx_2 + cz_1)$. This conceptual view clarifies the design intuition behind our masking strategy: masking is applied before expansion to hide sensitive plaintexts while enabling multi-key compatibility.

Algorithm 2 SMHE addition algorithm

Input: $\overline{ct} = (c_i)_{i\in[0,n]}$ and $\overline{ct}' = (c'_i)_{i\in[0,n]}$, associated with public keys $\{pk_j\}_{j\in T}$ and $\{pk_j\}_{j\in T'}$, respectively; The involved masking parameters $\{cz_k, \mathbf{\Gamma}_k\}_{k\in T\cup T'}$.
Return: $\overline{ct}_{add} = (\bar{c}_i)_{i\in[0,n]} \in R_Q^{n+1}$

$\bar{c}_0 = c_0 + c'_0 \pmod Q$
$\{cx_i = (0,0)\}_{i\in[1,n]}$, $\{cz'_i = (0,0)\}_{i\in[1,n]}$
for $i \in [1,n]$ **do**
  if $i \in T$ **then**
    $cx_i \leftarrow \texttt{Extend}^*(\mathbf{\Gamma}_i, pk_i, \{pk_j\}_{j\in T'\setminus i}) \pmod Q$
    $cz'_i \leftarrow \sum_{j\in T'\setminus i} cz_j \pmod Q$
  if $i \in T'$ **then**
    $cx_i \leftarrow cx_i + \texttt{Extend}^*(\mathbf{\Gamma}_i, pk_i, \{pk_j\}_{j\in T\setminus i}) \pmod Q$
    $cz'_i \leftarrow cz'_i + \sum_{j\in T\setminus i} cz_j \pmod Q$
  $(\bar{c}_0, \bar{c}_i) \leftarrow (\bar{c}_0, c_i + c'_i) + cx_i + cz'_i \pmod Q$

as defined in Definition 2, requiring that any view generated in the real-world execution is computationally indistinguishable from a simulated view generated in the ideal world, where only the final output is known.

Simulator Construction. Let $\mathcal{I}$ be the set of n honest parties, each holding a message μ_i and a key pair (pk_i, sk_i) generated via $\texttt{MKHE.KeyGen}$. Let $\mathcal{C}$ be the function evaluated over the encrypted messages, and let $\mu = \mathcal{C}(\mu_1, \ldots, \mu_n)$ be the final output.

We construct a PPT simulator Sim that, given only the public parameters pp and the final output μ, generates a view that is computationally indistinguishable from the adversary's real-world view:

- *Encryption Simulation:* SMHE employs RLWE-based encryption (e.g., CKKS or BFV). Under the RLWE assumption, the ciphertexts are semantically secure. Hence, Sim can simulate each ciphertext ct_i as a uniformly random element from the ciphertext space, without knowing the corresponding plaintext μ_i.
- *Evaluation Simulation:* The evaluation phase includes ciphertext expansion, masking, and arithmetic operations. Due to the introduction of masking terms during expansion, the evaluated ciphertext $\overline{ct}$ is statistically independent of the inputs. Thus, Sim can simulate $\overline{ct}$ and intermediate ciphertexts using fresh samples of the appropriate algebraic structure.
- *Partial Decryption Simulation:* Each party generates a partial decryption share ν_i, which includes fresh RLWE-style noise. The simulator Sim can generate these as random RLWE-like elements that are indistinguishable from actual decryptions. The final decryption result μ is known, so Sim can simulate the full decryption outcome consistently.

Security Guarantee. Let $\text{View}_{\mathcal{A}}^{\text{real}}$ denote the adversary's view in the real execution, and let $\text{View}_{\mathsf{Sim}}^{\text{ideal}}$ be the simulated view. Under the RLWE assumption, we have:

$$\text{View}_{\mathcal{A}}^{\text{real}}(\lambda) \approx_c \text{View}_{\mathsf{Sim}}^{\text{ideal}}(\lambda).$$

That is, no efficient adversary can distinguish between the real and ideal views with non-negligible probability. This proves that SMHE achieves simulation-based security against semi-honest adversaries.

Remark. Although our analysis is in the semi-honest setting, the SMHE scheme can be extended to handle malicious adversaries by incorporating verifiable computation techniques, such as zero-knowledge proofs [29] or interactive oracle proofs [30], which allow each party to prove correctness of operations without revealing private data. We leave the integration of such mechanisms to future work.

2) Correctness of Homomorphic Evaluation: We now present the correctness of the proposed homomorphic evaluation algorithm. We begin by analyzing the correctness in the two-party secure computation setting and subsequently extend the discussion to the general case.

Correctness of Homomorphic Addition for Two-Party Computation. The correctness of SMHE homomorphic addition is ensured by the correctness of the underlying masking scheme. Assume the two parties are P_1 and P_2, who independently generate their key materials as

$$(sk_i, pk_i, mp_i = (cz_i, \mathbf{\Gamma}_i), evk_i) \leftarrow \texttt{SMHE.KeyGen}(pp)$$

for $i = 1, 2$. Each party P_i encrypts its plaintext μ_i into a fresh ciphertext $ct_i = (c_0^i, c_1^i) \leftarrow \texttt{SMHE.Encrypt}(\mu_i, pk_i)$. The fresh ciphertexts are then expanded (associated with only the two-party encryption keys pk_1 and pk_2) as

$$\overline{ct}_1 := \texttt{SMHE.Expand}(\{pk_1, pk_2\}, 1, ct_1) = (c_0^1, c_1^1, 0),$$
$$\overline{ct}_2 := \texttt{SMHE.Expand}(\{pk_1, pk_2\}, 2, ct_2) = (c_0^2, 0, c_1^2),$$

which are further masked and added to obtain

$$\begin{aligned}
&\overline{ct}_{add} = (\bar{c}_0, \bar{c}_1, \bar{c}_2) \\
&\leftarrow \texttt{SMHE.Add}_2\left(\overline{ct}_1, \overline{ct}_2; pk_1, pk_2; \{cz_i, \mathbf{\Gamma}_i\}_{i\in[1,2]}\right) \\
&= (c_0^1 + c_0^2 + x_0^1 + x_0^2 + z_0^1 + z_0^2, c_1^1 + x_1^1 + z_1^2, c_1^2 + x_1^2 + z_1^1)
\end{aligned}$$

where the masking components are generated as

$$cz_1 = (z_0^1, z_1^1), cx_1 = (x_0^1, x_1^1) \leftarrow \texttt{Extend}(\mathbf{\Gamma}_1, pk_1, pk_2),$$
$$cz_2 = (z_0^2, z_1^2), cx_2 = (x_0^2, x_1^2) \leftarrow \texttt{Extend}(\mathbf{\Gamma}_2, pk_2, pk_1).$$

Decryption of $\overline{ct}_{add}$ is performed as follows: Each party P_i computes a partial decryption

$$\nu_i \leftarrow \texttt{SMHE.PartDec}(\bar{c}_i, s_i) = \bar{c}_i s_i + e_i \pmod Q, i \in [1,n]$$

where $e_i \leftarrow \chi$. The full decryption result is

$$\begin{aligned}
\mu &:= \texttt{SMHE.Merge}(\bar{c}_0, \{\nu_i\}_{i\in[1,n]}) \\
&= \bar{c}_0 + \bar{c}_1 s_1 + \bar{c}_2 s_2 + e_1 + e_2 \\
&\approx \langle (1, s_1, s_2), (c_0^1 + c_0^2 + x_0^1 + z_0^2 + x_0^2 + z_0^1, \\
&\quad c_1^1 + x_1^1 + z_1^2, c_1^2 + x_1^2 + z_1^1) \rangle \\
&= \langle sk_1, (ct_1 + cx_1 + cz_2) \rangle + \langle sk_2, (ct_2 + cx_2 + cz_1) \rangle \\
&\approx \langle sk_1, ct_1 \rangle + \langle sk_2, ct_2 \rangle \approx \mu_1 + \mu_2 \pmod Q.
\end{aligned} \tag{10}$$

Eq. (10) holds due to the correctness of the masking scheme, which ensures that $\langle sk_1, (cx_1 + cz_2) \rangle + \langle sk_2, (cx_2 + cz_1) \rangle \pmod Q \approx 0$. Therefore, the correctness of the homomorphic addition for two parties in SMHE is achieved.

Correctness of Homomorphic Addition in a General Case. Let $\overline{ct} = (c_i)_{i\in[0,n]}$ and $\overline{ct}' = (c'_i)_{i\in[0,n]}$ represent two expanded ciphertexts associated with encryption

keys $\{pk_j\}_{j\in T}$ and $\{pk_j\}_{j\in T'}$, respectively. Denote the plaintexts corresponding to these ciphertexts are μ and μ', respectively. That is, $\langle \overline{sk}, \overline{ct}\rangle \approx \mu \pmod Q$ and $\langle \overline{sk}, \overline{ct}'\rangle \approx \mu' \pmod Q$ hold, where $\overline{sk} = (1, s_1, \cdots, s_n)$. The masking parameters associated with the two expanded ciphertexts are given as $\{(cz_k, \mathbf{\Gamma}_k)\}_{k\in T\cup T'}$. The result of the homomorphic addition between the two expanded ciphertexts is $\overline{ct}_{add} = (\bar{c}_k)_{k\in[0,n]} \leftarrow \texttt{SMHE.Add}(\overline{ct}, \overline{ct}'; \{pk_j\}_{j\in T}, \{pk_j\}_{j\in T'}; \{cz_k, \mathbf{\Gamma}_k\}_{k\in T\cup T'})$ and its decryption proceeds as follows:

$$\begin{aligned}\langle \overline{sk}, \overline{ct}_{add}\rangle &= \bar{c}_0 + \sum_{i\in[1,n]} s_i \cdot \bar{c}_i \qquad (11)\\ &= \big((c_0 + c_0') + \sum_{i\in[1,n]} s_i \cdot (c_i + c_i')\big) + \sum_{i\in T\cup T'} \langle sk_i, (cx_i + cz_i')\rangle \\ &\approx \langle \overline{sk}, \overline{ct}\rangle + \langle \overline{sk}, \overline{ct}'\rangle \approx \mu + \mu' \pmod Q\end{aligned}$$

Eq. (11) holds due to the correctness of the masking scheme, which ensures that $\sum_{i\in T\cup T'} \langle sk_i, (cx_i + cz_i')\rangle \pmod Q \approx 0$. Therefore, the correctness of the homomorphic addition in SMHE is achieved.

Correctness of Homomorphic Multiplication. The correctness of homomorphic multiplication in SMHE is ensured by the correctness of ciphertext multiplications proposed in [20] and [21]. While this applies to the multiplication of two expanded ciphertexts derived from fresh ciphertexts, we now demonstrate that the correctness also holds for multiplications between an expanded ciphertext and a masked ciphertext, as well as for multiplications between two masked ciphertexts.

- Correctness of multiplications between an expansion ciphertext and a masked ciphertext: Let $\widehat{ct} = (\hat{c}_i)_{i\in[0,n]}$ denote a masked ciphertext, for example, $\widehat{ct} = \texttt{SMHE.Add}(\overline{ct}, \overline{ct}'; \{pk_j\}_{j\in T}, \{pk_j\}_{j\in T'}; \{cz_k, \mathbf{\Gamma}_k\}_{k\in T\cup T'})$, with a plaintext value of $\mu + \mu' \pmod Q$. Here, $\overline{ct} = (c_i)_{i\in[0,n]}$ and $\overline{ct}' = (c_i')_{i\in[0,n]}$ are two expanded ciphertexts defined as earlier. The result of the homomorphic multiplication between the masked ciphertext $\widehat{ct}$ and the expanded ciphertext $\overline{ct}'$ is $\overline{ct}_{\text{mult}} = (\bar{c}_i)_{i\in[0,n]} \leftarrow \texttt{SMHE.Mult}(\widehat{ct}, \overline{ct}'; \{evk_i\}_{i\in[1,n]})$, where

$$\begin{aligned}\bar{c}_0 &= \hat{c}_0 \cdot c_0' + \sum_{i\in[1,n]} \Big(\sum_{j\in[1,n]} \hat{c}_i \cdot c_j' \boxdot \mathbf{b}_j \Big) \boxdot \mathbf{v}_i \pmod Q,\\ \bar{c}_i &= \hat{c}_i \cdot c_0' + \hat{c}_0 \cdot c_i' + \sum_{j\in[1,n]} \hat{c}_j \cdot c_i' \boxdot \mathbf{d}_j \\ &\quad + \Big(\sum_{j\in[1,n]} \hat{c}_i \cdot c_j' \boxdot \mathbf{b}_j \Big) \boxdot \mathbf{u}_i \pmod Q\end{aligned}$$

for $i \in [1, n]$. The decryption of $\overline{ct}_{\text{mult}}$ proceeds as follows:

$$\begin{aligned}\langle \overline{sk}, \overline{ct}_{\text{mult}}\rangle &= \bar{c}_0 + \sum_{i\in[1,n]} s_i \cdot \bar{c}_i \\ &= \hat{c}_0 c_0' + \sum_{i\in[1,n]} s_i(\hat{c}_i \cdot c_0' + \hat{c}_0 \cdot c_i') + \sum_{i,j=1}^{n} \big((\hat{c}_i c_j' \boxdot \mathbf{d}_i) \cdot s_j \\ &\quad + \hat{c}_i c_j' \boxdot \mathbf{b}_j(\mathbf{v}_i + s_i \cdot \mathbf{u}_i)\big) \pmod Q.\end{aligned}$$

According to the relinearization algorithm in [20], [21], the following approximation holds:

$$(\hat{c}_i c_j' \boxdot \mathbf{d}_i) \cdot s_j + \hat{c}_i c_j' \boxdot \mathbf{b}_j(\mathbf{v}_i + s_i \cdot \mathbf{u}_i) \approx \hat{c}_i c_j' \cdot s_i s_j \pmod Q.$$

Thus, the decryption simplifies to (we define $s_0 = 1$):

$$\begin{aligned}\langle \overline{sk}, \overline{ct}_{\text{mult}}\rangle &\approx \sum_{i\in[0,n]} \sum_{j\in[0,n]} \hat{c}_i c_j' \cdot s_i s_j \\ &= \langle \overline{sk}, \widehat{ct}\rangle \times \langle \overline{sk}, \overline{ct}'\rangle = (\mu + \mu') \times \mu' \pmod Q.\end{aligned}$$

Therefore, the correctness of the homomorphic multiplication between a masked ciphertext and an expanded ciphertext is established.

- Correctness of multiplications between two masked ciphertexts: Similarly, the correctness of the homomorphic multiplication between two masked ciphertexts can be demonstrated by applying the same reasoning and leveraging the properties of the relinearization algorithm.

VI. Application to Privacy-Preserving FL

A. Specific Phase of PPFL Model

We utilize the proposed SMHE to construct PPFL model. As presented in Fig. 2, PPFL contains the following steps:

Initialization: Public parameters and keys generation.

- *Public parameter generation.* The server and the clients negotiate the security parameter λ and generates the public parameters $pp = \{N, t, Q, \chi\, \sigma, \mathbf{a}, \mathcal{H}, g\} \leftarrow \texttt{SMHE.Setup}(1^\lambda)$ an initial global model parameter $\hat{w}^0$.
- *Key generation.* Each client i generates its secret-public key pair $(sk_i = (1, s_i), pk_i = \{b_i, a\})$ by executing $\texttt{SMHE.KeyGen}(pp)$.

Model training and encryption (Client side): Each client i locally trains its model and encrypts its local update.

- *Model initialization.* Initializes the local model as $w_i^t \leftarrow \hat{w}^{(t-1)}$, where $t \geq 1$ represents the t-th round of training.
- *Training.* Trains its local model w_i^t with a mini-batch dataset to generate a local gradient g_i^t (a.k.a local model update).
- *Encryption.* The client encodes g_i^t as $\mu_i \in R$, encrypts μ_i as $\{ct_i, cz_i, \mathbf{\Gamma}_i\} \leftarrow \texttt{SMHE.Encrypt}(\mu_i, ek_i)$, and sends $\{ct_i, cz_i, \mathbf{\Gamma}_i\}$ to the server.

Secure aggregation (Server side): After receiving all the n clients' submissions, the server performs secure aggregation.

- *Ciphertext expandsion.* The server expands the ciphertexts $ct_i = (c_{i,0}, c_{i,1})$ $(i \in [1, n])$ into $\overline{ct}_i = (\bar{c}_{i,j})_{j\in[1,n]}$ by invoking $\texttt{SMHE.Expand}(\{pk_j\}_{j\in[1,n]}; i; ct_i)$. Here,

$$\bar{c}_{i,0} = c_{i,0} \quad \text{and} \quad \bar{c}_{i,j} = \begin{cases} c_{i,1} & \text{if } j = i, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } j \in [1, n]$$

- *Weighted aggregation.* The server selects some clients and aggregates their local gradients as a global gradient. Denote the selected client set as S_{benign} with the size $m = |S_{benign}|$. The server aggregates all the m ciphertexts $\{\overline{ct}_{id_i}\}_{id_i\in S_{benign}}$ as an aggregation ciphertext $\overline{ct}_{add}^m$. Specifically, the server first computes

$$\overline{ct}_{add}^1 \leftarrow \texttt{SMHE.Add}(\overline{ct}_{id_1}, (0)_{m+1}; id_1, id_m; \{cz_{id_1}, \mathbf{\Gamma}_{id_1}\}, \{cz_{id_m}, \mathbf{\Gamma}_{id_m}\})$$

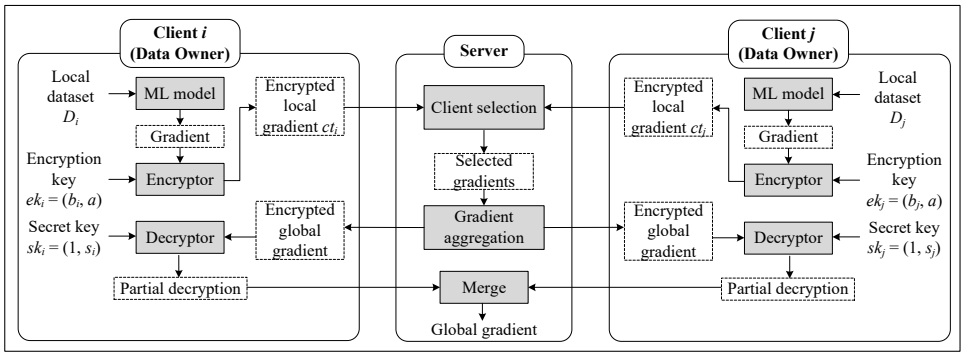


Fig. 2: High-level overview of robust and privacy-preserving federated learning.

For $i \in [2, m]$, the server computes

$$\overline{ct}^{i}_{add} \leftarrow \texttt{SMHE.Add}(\overline{ct}^{i-1}_{add}, \overline{ct}_{id_i}; id_{i-1}, id_i; \{cz_{id_{i-1}}, \mathbf{\Gamma}_{id_{i-1}}\}, \{cz_{id_i}, \mathbf{\Gamma}_{id_i}\}).$$

Here, we have $id_{i-1} = id_m$ when $i = 1$, and $id_{i+1} = id_1$ when $i = m$. The aggregation ciphertext is $\overline{ct}^{m}_{add} \triangleq (\bar{c}_0, \bar{c}_{id_1}, \cdots, \bar{c}_{id_m})$ and $\bar{c}_{id_i}$ is then sent to the client id_i.

Partial decryption (Client side): Each client id_i ($id_i \in S_{benign}$) computes the partial decryption result $\nu_{id_i} = \texttt{SMHE.PartDec}(\bar{c}_{id_i}, s_{id_i}) = \bar{c}_{id_i} \cdot s_{id_i} + e_{id_i} \pmod{Q}$ with $e_{id_i} \leftarrow \chi$. Then, it sends ν_{id_i} to the server.

Full decryption and model update (Server side):

- *Full decryption.* After receiving $\{\nu_{id_i}\}_{id_i \in S_{benign}}$ from all the m clients in S_{benign}, the server obtains the final decryption result $\mu = \bar{c}_0 + \sum_{id_i \in S_{benign}} \nu_{id_i} \pmod{Q}$. Then the server decodes μ as the weighted aggregation gradient $\hat{g}_t$.
- *Parameter update.* The server updates the global model as $\hat{w}_t := \hat{w}_{t-1} - \eta \frac{\hat{g}_t}{\sum_{i=1}^{m} \alpha_i}$ and sends $\hat{w}_t$ to the clients for the next iteration until the final training model is obtained.

B. Security and Correctness

The security of the PPFL model is guaranteed by the underlying security of the SMHE scheme, as analyzed in Section V-B1. We now analyze the correctness of the ciphertext aggregation and decryption processes within PPFL.

During the aggregation process, the aggregated ciphertext is computed as $\overline{ct}^{1}_{\text{add}} \leftarrow \texttt{SMHE.Add}(\overline{ct}_{id_1}, (0)_{m+1}; id_1, id_m; \{cz_{id_1}, \mathbf{\Gamma}_{id_1}\}, \{cz_{id_m}, \mathbf{\Gamma}_{id_m}\})$, yielding:

$$\overline{ct}^{1}_{\text{add}} = (c_0^{id_1} + x_0^{id_1,id_m} + z_0^{id_m} + x_0^{id_m,id_1} + z_0^{id_1}, \; c_1^{id_1} + x_1^{id_1,id_m} + z_1^{m}, 0, \ldots, 0, x_1^{id_m,id_1} + z_1^{id_1}),$$

and the final aggregated ciphertext is denoted as $\overline{ct}^{m}_{\text{add}} \triangleq (\bar{c}_0, \bar{c}_{id_1}, \ldots, \bar{c}_{id_m})$, where

$$\bar{c}_0 = \sum_{id_i \in S_{benign}} \left(c_0^{id_i} + x_0^{id_i,id_{i-1}} + x_0^{id_i,id_{i+1}} + 2z_0^{id_i} \right),$$

$$\bar{c}_{id_i} = c_1^{id_i} + x_1^{id_i,id_{i-1}} + x_1^{id_i,id_{i+1}} + z_1^{id_{i-1}} + z_1^{id_{i+1}}, i \in [1, m].$$

Here, $m = |S_{benign}|$ and $cx_{id_i,id_j} = (x_0^{id_i,id_j}, x_1^{id_i,id_j}) \leftarrow \texttt{Extend}(\mathbf{\Gamma}_{id_i}, pk_{id_i}, pk_{id_j})$ for $id_i, id_j \in S_{benign}$, with $id_{i-1} = id_m$ when $i = 1$, and $id_{i+1} = id_1$ when $i = m$.

The decryption of the aggregated ciphertext $\overline{ct}^{n}_{\text{add}}$ is

$$\begin{aligned}
\mu &= \bar{c}_0 + \sum_{id_i \in S_{benign}} (\bar{c}_{id_i} \cdot s_{id_i} + e_{id_i}) \qquad (12)\\
&= \sum_{id_i \in S_{benign}} \Big((c_0^{id_i} + c_1^{id_i} s_{id_i} + e_{id_i}) + \langle sk_{id_i}, cx_{id_i,id_{i-1}} \rangle \\
&\quad + \langle sk_{id_i}, cz_{id_{i-1}} \rangle + \langle sk_{id_i}, cx_{id_i,id_{i+1}} \rangle + \langle sk_{id_i}, cz_{id_{i+1}} \rangle \Big) \\
&= \sum_{id_i \in S_{benign}} (c_0^{id_i} + c_1^{id_i} s_{id_i}) + e^* \approx \sum_{id_i \in S_{benign}} \mu_{id_i} \pmod{Q}.
\end{aligned}$$

Eq. (12) holds due to the correctness of the masking scheme, which ensures that

$$\begin{aligned}
&\langle sk_{id_i}, cx_{id_i,id_{i-1}} \rangle + \langle sk_{id_i}, cz_{id_{i-1}} \rangle \\
&= \langle sk_{id_i}, (b_{id_{i-1}} - b_{id_i}) \boxdot (\boldsymbol{\varsigma}_0^{id_{i-1}}, \boldsymbol{\varsigma}_1^{id_{i-1}}) \rangle \\
&\quad + \langle sk_{id_i}, (r_{id_{i-1}} ek_{id_i} + (e_0, e_1)) \rangle \\
&\approx r_{id_{i-1}}(b_{id_{i-1}} - b_{id_i}) + r_{id_{i-1}}(b_{id_i} - b_{id_{i-1}}) = 0 \pmod{Q},
\end{aligned}$$

and similarly, $\langle sk_{id_i}, cx_{id_i,id_{i+1}} \rangle + \langle sk_{id_i}, cz_{id_{i+1}} \rangle \approx 0 \pmod{Q}$. The error bound of the decryption is $\|e^*\|_\infty \leq (3m^3 + 6m^2 + m)B_\chi + 2dm^2 \cdot B_\chi B_{\mathcal{H}}$.

C. Implementation

We conduct PPFL experiments using SMHE, where the multi-key CKKS scheme is applied to encrypt the gradients.

1) Experimental Setup: The experiments were conducted on a computer equipped with an Nvidia GeForce GTX 1080 Ti GPU and an Intel Core i7-6700 CPU. The SMHE algorithms were implemented in C++ using the NTL 10.4.0 [31] and GMP 6.2.1 [32] libraries to handle arbitrary-length integers and high-precision arithmetic. The FL models were executed using the PyTorch 1.11.0 framework in Python 3.8. To build our PPFL framework, a dynamic library containing all SMHE-related code was generated and invoked within a Python script. The code is available at https://github.com/JiahuiWu2022/SMHE.git.

Benchmark Models, Dateset, and Parameter Setting. To evaluate the effectiveness of our SMHE-based PPFL model, we conduct comparative experiments against the existing multiparty HE schemes: CDKS [21], THE [14], and MKGSW [17]. All models are assessed under two distinct learning scenarios to ensure comprehensive and fair comparison. In the first scenario, we train a fully connected neural network (FCN)

on the MNIST dataset. The network includes an input layer with 784 neurons, two hidden layers with 128 and 64 neurons respectively, and an output layer with 10 neurons representing digit classes from 0 to 9 using one-hot encoding. The model is optimized using the Adam algorithm with a mini-batch size of 64. In the second scenario, we adopt the CIFAR-10 dataset to train a lightweight version of AlexNet with approximately 1.25 million parameters. Training is conducted using a batch size of 128 and the RMSprop optimizer with a decay factor of 10^{-6}. For both FL tasks, the datasets are randomly partitioned among multiple clients for simulating collaborative training.

For the SMHE parameters, we largely follow the settings used in CDKS and THE [14], [20], [21] to ensure a fair comparison. Specifically, the distribution $\chi \leftarrow R$ is instantiated with coefficients drawn from $\{0, 1\}$, and the error term is sampled from a discrete Gaussian distribution $e \leftarrow D_\sigma$ with standard deviation $\sigma = 3.2$. For gadget decomposition, we adopt the RNS-friendly method proposed by Bajard et al. [33]. We set the gadget vector dimension to $\tau = 8$, the RLWE polynomial dimension to $N = 2^{14}$, the security parameter to $\lambda = 256$, and the number of slots to $n_s = 8192$. The actual security level under standard RLWE assumptions corresponds to 128 bits, as estimated by the LWE estimator. To accelerate the masking encryption and masking extension procedures, we incorporate the SIMD strategy proposed in [34]. For the MKGSW parameters, we follow the TFHE setting [35] to achieve a 128-bit security level by setting the LWE dimension to 512, the modulus to 2^{32}, and the gadget dimension to 10. The computation and communication complexity of SMHE scales linearly with the number of clients n, which is consistent with both CDKS and THE, whereas MKGSW exhibits quadratic growth with respect to n. Since SMHE inherits the homomorphic multiplication procedure from CDKS without modification, its multiplication complexity remains identical to that of CDKS. Therefore, in our experiments, we focus on comparing the runtime and communication overhead of the homomorphic addition algorithm among SMHE, CDKS, THE, and MKGSW in the context of PPFL applications.

2) Experimental Results: We report the performance of PPFL models using SMHE, CDKS, THE, and MKGSW, focusing on runtime, communication traffic, and model accuracy.

Runtime. Fig. 3 presents the runtime breakdown of PPFL models across different computation phases on both the client and server sides for a single training iteration. The results are averaged over 100 training iterations for all models except the MKGSW-based model, whose runtime is approximated using the MKGSW evaluation time reported in TFHE [35]. As a bit-level HE scheme, MKGSW incurs an impractically high runtime due to its inherent computational complexity. For example, under the FCN setting, the total runtime reaches 3570.65s on the client side and 2702.17s on the server side. This excessive overhead renders MKGSW unsuitable for real-world PPFL applications. In contrast, the other three models, THE-based, CDKS-based, and our SMHE-based models, exhibit significantly lower runtimes, making them more feasible for practical deployment. Among these, THE-based and CDKS-based models demonstrate comparable runtime performance. Our SMHE-based scheme introduces a moderate increase in runtime relative to THE and CDKS, primarily due to the incorporation of the masking extension mechanism. For instance, under the AlexNet setting, the total client-side runtime of SMHE increases by less than $2\times$ compared to the CDKS- and THE-based models (4.64s vs. 2.72s and 7.77s, respectively). Similarly, on the server side, SMHE incurs a higher overall runtime – 2.17s for FCN and 38.22s for AlexNet – compared to CDKS (0.97s and 31.23s) and THE (0.96s and 32.58s). In all cases, the runtime increase remains within a $2\times$ range, which we consider a reasonable trade-off given the enhanced security guarantees provided by SMHE.

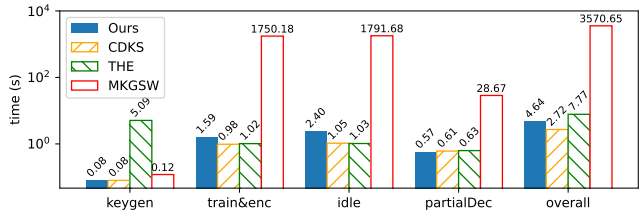
Besides, compared to THE, our approach significantly reduces the client-side key generation time (e.g., 0.08s vs. 5.09s for FCN) by allowing each participant to generate their key locally, without the need for interactive threshold key generation protocols. This eliminates the communication overhead inherent in THE's distributed key setup, leading to a substantial reduction in key generation and distribution latency.

Overall, SMHE strikes a practical balance between improved security, flexibility, and runtime efficiency. Compared to MKGSW, it achieves vastly lower computational overhead, while maintaining acceptable runtime increases relative to THE and CDKS. These features make SMHE a scalable and effective solution for PPFL applications.

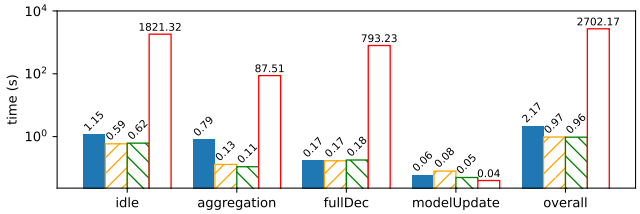
TABLE II: Comparison of communication traffic for one iteration of different PPFL models.

Model\Network	FCN	AlexNet
Plain (#gradient)	109.386K	1.25M
THE-based (cipher traffic)	7.00MB	76.50MB
CDKS-based (cipher traffic)	7.00MB	76.50MB
SMHE-based (cipher traffic)	16.00MB	139.50MB
MKGSW-based (cipher traffic)	33.39GB	381.47GB

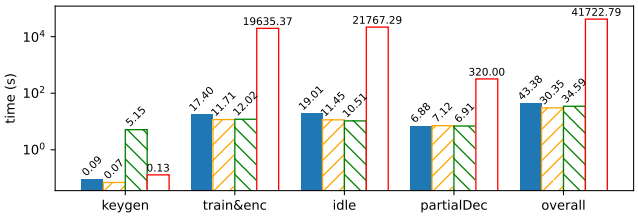
Communication Traffic. Table II presents the communication traffic per iteration for different PPFL models under FCN and AlexNet networks. The row labeled "Plain (#gradient)" reflects the number of model gradients in the plain FL, $109{,}386$K for FCN and 1.25M for AlexNet. For encrypted models, the MKGSW-based model introduces an extremely high communication burden, 33.39GB per iteration for FCN and 381.47GB for AlexNet, mainly due to its bit-level encryption. This renders MKGSW impractical for real-world PPFL applications, where communication efficiency is a critical concern. In contrast, the other three models: THE-based, CDKS-based, and our SMHE-based scheme offer significantly lower communication costs and are more suitable for practical deployment. Specifically, THE-based and CDKS-based models share similar ciphertext formats and therefore exhibit identical communication traffic, requiring only 7MB and 76.5MB per iteration for FCN and AlexNet, respectively. Our SMHE-based model incurs a moderate increase in communication traffic, reaching 16MB for FCN and 139.5MB for AlexNet. This overhead stems from the ciphertext extension mechanism introduced by our masking design to support secure multi-key computation. However, despite the increase, the total traffic remains within approximately $2\times$ that of CDKS and THE, which is a reasonable cost given the added benefits. Overall, SMHE offers a practical and scalable solution that



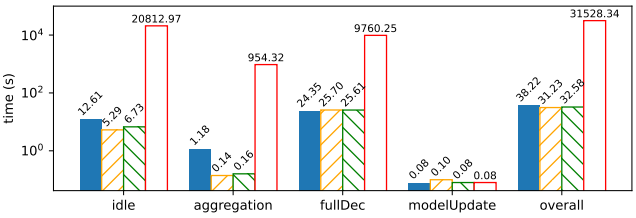
(a) FCN client-side time distribution



(b) FCN server-side time distribution



(c) AlexNet client-side time distribution



(d) AlexNet server-side time distribution

Fig. 3: Runtime breakdown comparison of PPFL models using different HEs , where "idle" denotes the duration during which the clients/server remain inactive while waiting for other parties to complete their operations. For CDKS and THE, "train&enc" represents the time spent on local model training and encryption. For MKGSW and SMHE, this stage additionally includes the time for masking generation.

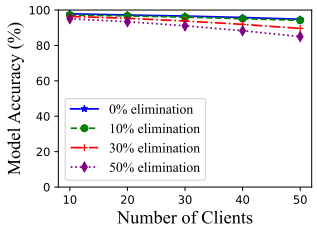
balances enhanced security and key flexibility with acceptable communication overhead, unlike MKGSW, whose bit-level structure incurs prohibitive costs in PPFL settings.

Model Accuracy. Fig. 4 displays the final model accuracy of the PPFL models under different client elimination rates. As observed from the results, the SMHE (CDKS)-based model exhibits strong robustness against client elimination. Even with 50 distributed clients and a high elimination rate of 50%, the final model accuracy remains above 85% for FCN and 66% for AlexNet, indicating that the model tolerates a substantial fraction of client drop without significant performance degradation. In contrast, the THE-based model (where the threshold is set to 0.8 in the experiment) shows a drastic decline in accuracy once the proportion of eliminated clients exceeds the threshold. The final model accuracy drops sharply, eventually reaching a level close to that of random guessing, indicating the failure of effective model convergence. These results demonstrate that SMHE-based model offers greater fault tolerance and flexibility in the presence of unreliable or excluded clients, whereas the THE-based system is sensitive to threshold violations due to their dependency on sufficient participant contributions.

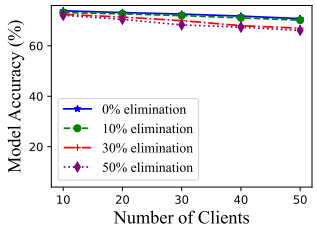
TABLE III: Comparisons of total training time and network traffic for converged FCN and AlexNet models under different PPFL modes.

Model	Mode	Epochs	Accuracy	Time(h)	Traffic(GB)
FCN	Plain	34	97.94%	0.13	0.65
	THE	34	97.64%	7.75	21.85
	CDKS	37	97.62%	3.57	23.77
	SMHE	35	97.90%	6.22	41.40
AlexNet	Plain	268	74.01%	4.21	48.67
	THE	267	73.79%	194.29	777.92
	CDKS	270	73.97%	180.12	786.67
	SMHE	263	73.84%	232.49	1397.31

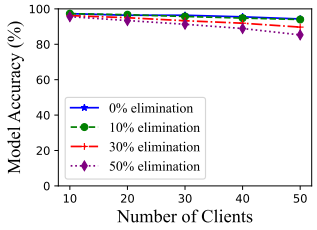
Training to Convergence. We next compare the total



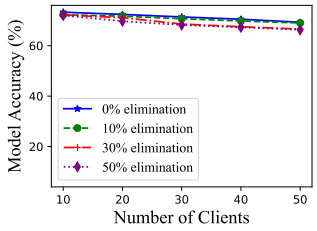
(a) FCN model accuracy for SMHE-based PPFL model



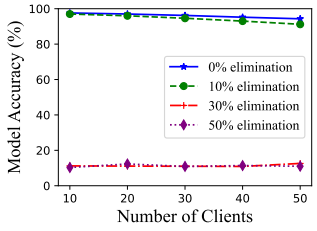
(b) AlexNet model accuracy for SMHE-based PPFL model



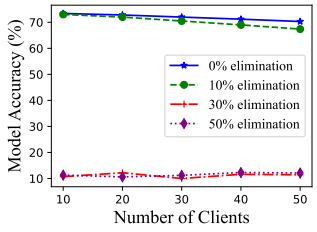
(c) FCN model accuracy for CDKS-based PPFL model



(d) AlexNet model accuracy for CDKS-based PPFL model



(e) FCN model accuracy for THE-based PPFL model



(f) AlexNet model accuracy for THE-based PPFL model

Fig. 4: Comparison of final model accuracy under different client elimination rates using SMHE/CDKS/THE scheme, where the models are all trained for $10,000$ iterations.

training time and network traffic of THE-, CDKS-, and SMHE-based PPFL models until convergence is reached. The number of clients is set to 10. Due to the exceedingly high time and communication costs involved in the full training process, we simulate federated learning locally until convergence and estimate the total training time and communication cost based on the bandwidth and iteration counts. Table III presents the estimated results for both FCN and AlexNet models, from which we can observe that (1) All PPFL models using SMHE, CDKS, and THE achieve high prediction accuracy, exceeding 97% on FCN and around 74% on AlexNet. (2) For the FCN model, SMHE achieves comparable accuracy (97.90%) to the plaintext baseline (97.94%), with only slightly more training epochs (35 vs. 34). It incurs 6.22 hours of training time and 41.40 GB network traffic, which are less than $2\times$ that of THE and CDKS. (3) For the more complex AlexNet model, SMHE also reaches similar accuracy to the plaintext case (73.84% vs. 74.01%) with slightly fewer epochs. However, it requires 232.49 hours and 1397.31 GB of traffic, which are less than $1.8\times$ of THE and CDKS. Overall, compared with THE and CDKS, the SMHE-based PPFL models introduce only a modest increase in training time and communication cost (less than $2\times$) while ensuring stronger security guarantees and offering better dynamism, making it a practical and secure choice for PPFL applications.

VII. Conclusion

In this paper, we propose a novel secure multi-key homomorphic encryption (SMHE) scheme tailored for privacy-preserving FL. Our scheme addresses critical security vulnerabilities identified in previous MKHE schemes, specifically those proposed by Chen et al. and Kim et al., which inadvertently reveal plaintext information during multiparty secure computation tasks such as federated learning. By introducing a masking scheme into the CKKS and BFV frameworks, our enhanced scheme ensures the confidentiality of data while supporting efficient and secure homomorphic operations. Future work includes further optimizing the efficiency of ciphertext expansion in SMHE and exploring its integration with advanced applications such as secure model inference and encrypted machine learning.

References

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. ACM Symp. Theory Comput. (STOC)*, 2009, pp. 169–178.

[2] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Proc. Int. Cryptol. Conf. (CRYPTO)*, 2012, pp. 868–886.

[3] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[4] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Int. Cryptol. Conf. (CRYPTO)*, 2013, pp. 75–92.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, 2014.

[6] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, 2016, pp. 3–33.

[7] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, 2017, pp. 409–437.

[8] H. B. McMahan *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1, 2021. [Online]. Available: https://arxiv.org/abs/1912.04977

[9] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold fhe," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Tech. (EUROCRYPT)*, 2012, pp. 483–501.

[10] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Proc. Int. Cryptol. Conf. (CRYPTO)*, 2018, pp. 565–596.

[11] C. Mouchet, E. Bertrand, and J.-P. Hubaux, "An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption," *Journal of Cryptology*, vol. 36, no. 2, p. 10, 2023.

[12] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proc. Priv. Enhancing Technol. Symp. (PETS)*, vol. 2021, no. 4, pp. 291–311, 2021.

[13] J. Park, "Homomorphic encryption for multiple users with less communications," *IEEE Access*, vol. 9, pp. 135 915–135 926, 2021.

[14] J. Ma, S.-A. Naas, S. Sigg, and X. Lyu, "Privacy-preserving federated learning based on multi-key homomorphic encryption," *Int. J. Intell. Syst.*, vol. 37, no. 9, pp. 5880–5901, 2022.

[15] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proc. ACM Symp. Theory Comput. (STOC)*, 2012, pp. 1219–1234.

[16] M. Clear and C. McGoldrick, "Multi-identity and multi-key leveled fhe from learning with errors," in *Proc. Int. Cryptol. Conf. (CRYPTO)*, 2015, pp. 630–656.

[17] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key fhe," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Tech. (EUROCRYPT)*, 2016, pp. 735–763.

[18] C. Peikert and S. Shiehian, "Multi-key fhe from lwe, revisited," in *Proc. Theory Cryptogr. Conf. (TCC)*, 2016, pp. 217–238.

[19] H. Chen, I. Chillotti, and Y. Song, "Multi-key homomorphic encryption from tfhe," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, 2019, pp. 446–472.

[20] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2019, pp. 395–412.

[21] T. Kim, H. Kwak, D. Lee, J. Seo, and Y. Song, "Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2023, pp. 726–740.

[22] L. Brandão and R. Peralta, "Nist first call for multi-party threshold schemes," *Nat. Inst. Standards Technol., Gaithersburg, MD, USA*, 2023.

[23] Z. Brakerski and R. Perlman, "Lattice-based fully dynamic multi-key fhe with short ciphertexts," in *Proc. Int. Cryptol. Conf. (CRYPTO)*, 2016, pp. 190–213.

[24] L. Chen, Z. Zhang, and X. Wang, "Batched multi-hop multi-key fhe from ring-lwe with compact ciphertext extension," in *Proc. Theory Cryptogr. Conf. (TCC)*, 2017, pp. 597–627.

[25] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, no. 6, pp. 1–40, 2009.

[26] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of the ACM*, vol. 60, no. 6, pp. 1–35, 2013.

[27] X. Hao, C. Lin, W. Dong, X. Huang, and H. Xiong, "Robust and secure federated learning against hybrid attacks: A generic architecture," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 1576–1588, 2024.

[28] A. Nawaz, G. Chen, M. U. Raza, Z. Iqbal, J. Li, V. C. Leung, and J. Chen, "Secure distributed sparse gaussian process models using multi-key homomorphic encryption," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, vol. 38, no. 13, 2024, pp. 14 431–14 439.

[29] H. Sun, J. Li, and H. Zhang, "zkllm: Zero knowledge proofs for large language models," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2024, pp. 4405–4419.

[30] D. F. Aranha, A. Costache, A. Guimarães, and E. Soria-Vazquez, "Heliopolis: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, 2025, pp. 302–334.

[31] "Ntl libarary," *https://libntl.org/*, 2017.

[32] "Bmp libarary," *https://gmplib.org/*, 2020.

[33] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A full rns variant of fv like somewhat homomorphic encryption schemes," in *Proc. Sel. Areas Cryptogr. (SAC)*, 2016, pp. 423–442.
[34] J. Wu, W. Zhang, and F. Luo, "Esafl: Efficient secure additively homomorphic encryption for cross-silo federated learning," *IEEE Trans. Dependable Secure Comput.*, 2025.
[35] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.