A New Approach Towards Encrypted Data Sharing and Computation: Enhancing Efficiency Beyond MPC and Multi-Key FHE

Anil Kumar Pradhan

Vaultree Ltd.

Abstract. In this paper, we introduce a novel approach to Multi-Key Fully Homomorphic Encryption (MK-FHE) that enhances both efficiency and security beyond the capabilities of traditional MK-FHE and Multi-Party Computation (MPC) systems. Our method generates a unified key structure, enabling constant ciphertext size and constant execution time for encrypted computations, regardless of the number of participants involved. This approach addresses critical limitations such as ciphertext size expansion, noise accumulation, and the complexity of relinearization, which typically hinder scalability in multi-user environments. We also propose a new decryption method that simplifies decryption to a single information exchange, in contrast to traditional multi-key FHE systems that require information to be passed between all parties sequentially.
Additionally, it significantly enhances the scalability of MK-FHE systems, allowing seamless integration of additional participants without introducing performance overhead. Through theoretical analysis and practical implementation, we demonstrate the superiority of our approach in large-scale, collaborative encrypted computation scenarios, paving the way for more robust and efficient secure data processing frameworks. Further more, unlike the threshold based FHE schemes, the proposed system doesn't require a centralised trusted third party to split and distribute the individual secret keys, instead each participant independently generates their own secret key, ensuring both security and decentralization.

Keywords: Multi-Party Computation, Multi-Key FHE, Fully Homomorphic Encryption (FHE)

1 Introduction

The rapid advancement of cloud computing and distributed systems has led to a growing demand for secure and efficient methods of encrypted data sharing and computation across multiple parties. As organizations increasingly rely on cloud infrastructure for collaboration and data processing, the need to perform computations on sensitive data without compromising privacy has become critical. Traditional encryption techniques are insufficient for this purpose, as they require data to be decrypted for processing, exposing it to potential threats. Hence, more advanced cryptographic techniques, such as Multi-Party Computation (MPC) and Multi-Key Fully Homomorphic Encryption (MK-FHE), have gained significant attention for their ability to perform secure computations on encrypted data [2,13,24]. However, both MPC and MK-FHE come with notable limitations that hinder their practicality in large-scale applications.

Multi-Party Computation (MPC) provides a powerful framework for secure computation by enabling multiple parties to collaboratively compute a function over their inputs without revealing them to one another [15,19]. This is achieved through complex interactive protocols that allow each participant to contribute without exposing their data [3]. MPC's flexibility allows it to handle a wide range of computations, making it highly adaptable [5]. However, the interactive nature of MPC presents scalability challenges, as the number of communication rounds and the complexity of protocols grow rapidly with the number of participants. This makes MPC less suitable for scenarios where large-scale, non-interactive computation is required [10].

Fully Homomorphic Encryption (FHE) represents a breakthrough in cryptography by enabling arbitrary computations on encrypted data without the need for decryption [13, 14, 22]. FHE allows operations like addition and multiplication to be performed directly on ciphertexts, which opens up vast possibilities for secure data processing in sensitive environments such as medical research, financial analysis, and cloud computing [1, 12]. However, standard FHE is typically limited to single-user applications, making it challenging to extend its benefits to multi-party systems where multiple users want to perform computations on shared encrypted data [8, 18].

Multi-Key Fully Homomorphic Encryption (MK-FHE) was introduced as an advanced version of FHE to support multi-user scenarios, allowing computations on encrypted data from multiple participants without requiring them to decrypt their data [7, 9]. In MK-FHE, each user encrypts their data with their own key, and a computation can be performed on the combined ciphertexts without ever exposing the underlying data [17]. This non-interactive approach overcomes some of the limitations of MPC by removing the need for communication between parties during the computation phase [23]. However, MK-FHE introduces new challenges, such as exponential growth in ciphertext size, increased noise accumulation, and more complex relinearization processes as the number of participants grows [21]. These issues become more pronounced as additional users are added, resulting in exponential growth in computational overhead. This significantly degrades performance and scalability, making MK-FHE impractical for large-scale applications [6].

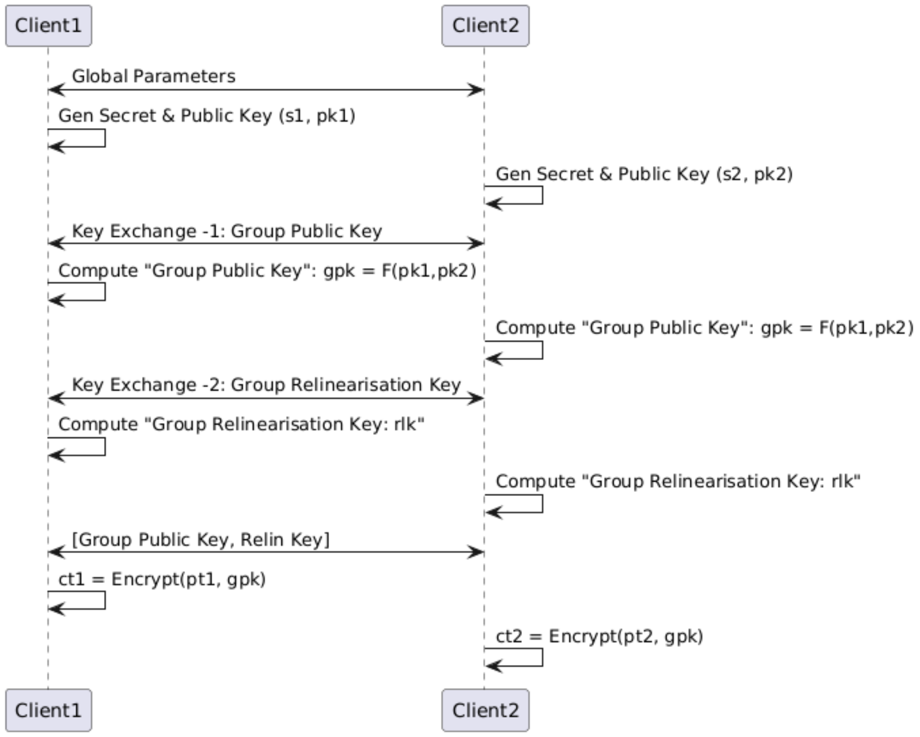
Despite these limitations, MK-FHE remains a promising solution for secure multi-party computation. To address these challenges, this paper presents a novel approach that addresses the inherent inefficiencies in existing systems. By introducing a unified key generation and management process, our method reduces noise accumulation and maintains constant ciphertext size and execution time, regardless of the number of participants [16]. This innovation enhances both efficiency and scalability, making secure multi-party computation more practical and accessible for a broader range of applications [11]. This approach is particularly well-suited for environments where the number of participants may increase over time, such as collaborative cloud computing platforms, without compromising performance or security [4, 20].

Higher-Level Design:

The core technical innovation in our approach lies in the novel key generation process, and the simplified decryption mechanism, which together ensure both efficiency and scalability. These innovations allow for secure and practical multi-user encrypted computation without the typical performance overhead seen in traditional Multi-Key Fully Homomorphic Encryption (MK-FHE) systems.

- Key Generation
  - Each participant generates their own unique secret key and public key independently, without relying on any centralized trusted entity.
  - Key Exchange -1: Through a collaborative key exchange protocol, participants combine their public keys to compute a unified *group public key* and corresponding to a *group secret key.* This process effectively amalgamates the secret keys of all participants without revealing them.
  - Key Exchange -2: Participants execute another key exchange protocol to compute a *group relinearization key* corresponding to a *group secret key*, which is essential for efficient homomorphic multiplication.
- Encryption: All users encrypt their data using the *group public key.*
- Homomorphic Operations: Once the key generation process is complete, the encryption, homomorphic addition, and multiplication operations follow the standard procedures of single-key Fully Homomorphic Encryption (FHE) systems.
- Decryption: Each participant generates a *partial decryption key* of the ciphertext using their secret key, and these *partial decryption keys* are combined to perform the final decryption.

Unlike traditional MK-FHE systems that require multiple sequential information exchanges between participants, our approach enables all users to participate in parallel that minimizes communication overhead and simplifies the decryption process, further enhancing efficiency. Additionally, the partial decryption keys are ciphertext-specific, ensuring that they cannot be reused across different ciphertexts, maintaining strong security.


Client1
Client2
Global Parameters
Gen Secret & Public Key (s1, pk1)
Gen Secret & Public Key (s2, pk2)
Key Exchange -1: Group Public Key
Compute "Group Public Key": gpk = F(pk1,pk2)
Compute "Group Public Key": gpk = F(pk1,pk2)
Key Exchange -2: Group Relinearisation Key
Compute "Group Relinearisation Key: rlk"
Compute "Group Relinearisation Key: rlk"
[Group Public Key, Relin Key]
ct1 = Encrypt(pt1, gpk)
ct2 = Encrypt(pt2, gpk)
Client1
Client2

New Multi-Key FHE Flow Diagram

This method of key generation not only maintains the constant ciphertext size and execution time but also allows for seamless integration of additional participants without impacting system performance. By reducing the computational burden associated with multi-user setups and eliminating the complexities of traditional MK-FHE, our approach represents a significant advancement in the field of homomorphic encryption, enabling more robust and efficient secure computation frameworks.

This paper presents a significant advancement in the field of encrypted data sharing and computation by overcoming the critical limitations of existing Multi-Key FHE systems. The proposed approach not only enhances the practicality and efficiency of secure data sharing and computation but also opens up new possibilities for scalable, collaborative encrypted data processing. By reducing the computational burden associated with multi-user systems, this approach represents a leap forward in homomorphic encryption, enabling more robust and efficient secure computation frameworks. Our contribution represents a significant step forward in the development of homomorphic encryption technologies, enabling more robust and efficient systems for secure data sharing and computation in distributed environments.

The rest of the paper is organized as follows: *Preliminaries* introduces the foundational cryptographic concepts, including Fully Homomorphic Encryption (FHE), Multi-Key FHE, and the Ring Learning with Errors (RLWE) problem, that underpin the proposed scheme. *Core Ideas and Analysis* dives into the Multi-Key Fully Homomorphic Encryption Protocol, explaining the key principles and innovative methods behind the system. In the *Construction* section, the detailed steps of the cryptographic protocol are outlined, including key generation, encryption, and decryption processes. *Security Analysis* provides a comprehensive examination of the scheme's resilience against cryptographic attacks, focusing on confidentiality, integrity, and key security. The *Performance Analysis and Key Improvements* section evaluates the system's computational efficiency and scalability, with a discussion on potential optimizations. Finally, *Applications and Use-Cases* explores practical scenarios in which the proposed MK-FHE protocol can be applied, including privacy-preserving data analysis and collaborative cloud computing environments.

2 Preliminaries

2.1 Overview of Encrypted Data Sharing and Collaboration

Multi-Party Computation (MPC): Multi-Party Computation (MPC) is a cryptographic protocol that enables multiple parties to jointly compute a function over their private inputs while ensuring that no party learns anything about the other parties' inputs except for what can be inferred from the output. This is achieved through techniques such as secret sharing and secure function evaluation, where inputs are split into shares distributed among the parties or encrypted, and the computation is carried out collaboratively.

Fully Homomorphic Encryption (FHE): A cryptographic method that allows computation on ciphertexts, producing an encrypted result that, when decrypted, matches the result of operations performed on the plaintexts. FHE supports arbitrary computations on encrypted data.

Multi-Key Fully Homomorphic Encryption (Multi-Key FHE): Multi-Key Fully Homomorphic Encryption (Multi-Key FHE) is an advanced cryptographic method that extends the capabilities of Fully Homomorphic Encryption (FHE) to support computations on encrypted data from multiple parties, each using their own encryption key. This powerful technique allows data encrypted under different keys to be combined and processed together, enabling joint computation on sensitive data without requiring any party to reveal their private inputs.

2.2 Advantages and Benefits of Multi-Key FHE

Multi-Key Fully Homomorphic Encryption offers significant advantages over traditional Multi-Party Computation, particularly in scenarios where non-interactive, secure computation is required. Its ability to perform computations on encrypted data from multiple parties, combined with strong privacy guarantees, scalability, and efficiency, makes it a powerful tool for privacy-preserving applications in diverse fields. While MPC provides flexibility and adaptability, Multi-Key FHE's non-interactive nature and robust security make it an increasingly attractive option for secure, collaborative computation. Conclusion

Both MPC and Multi-Key FHE address the need for secure multi-party computation, but they approach the problem differently. MPC focuses on interactive protocols where privacy is maintained through secret sharing and controlled communication, making it flexible but sometimes complex. In contrast, Multi-Key FHE allows non-interactive computation on encrypted data, offering strong security but often at a higher computational cost and with less flexibility in terms of the types of computations that can be performed. Each has its advantages, with MPC being more general and adaptable, while Multi-Key FHE is powerful for specific use cases involving encrypted data. Comparison Between MPC and Multi-Key FHE

Aspect	**Multi-Party Computation (MPC)**	**Multi-Key Fully Homomorphic Encryption**
Computation Method	Interactive: Requires multiple rounds of communication between parties.	Non-interactive: Computation is performed on encrypted data without interaction.
Privacy Model	Privacy through secret sharing and controlled communication.	Strong privacy through encryption; no plaintext exposure.
Flexibility	Highly flexible, suitable for a wide range of functions and scenarios.	Supports arbitrary computations on encrypted data, ideal for specific applications like encrypted machine learning.
Complexity	May require complex communication protocols, especially in the presence of malicious adversaries.	Reduced complexity in multi-party settings; streamlined non-interactive processes.
Scalability	May face challenges with scalability due to communication overhead.	Scales efficiently with the number of parties, making it suitable for large-scale applications.
Security	Secure against semi-honest and malicious adversaries with proper protocols.	Strong security guarantees, including resistance to quantum attacks and minimal trust assumptions.

Table 1. Comparison Between MPC and Multi-Key FHE

2.3 FHE Scheme

The Fully Homomorphic Encryption (FHE) scheme allows for arbitrary computations on encrypted data without needing to decrypt it. The BFV and BGV scheme are based on the Learning With Errors (LWE) problem and its ring variant, the Ring Learning With Errors (RLWE) problem.

Notation:

– $[x]_q = x \mod q$

Below is an explanation of the FHE schemes, including its key components, encryption and decryption processes, and homomorphic operations.

Key Components

– Plaintext Space: Typically a polynomial ring $\mathbb{Z}_t[X]/(X^n+1)$, where t is the plaintext modulus.
– Ciphertext Space: Polynomials in $\mathbb{Z}_q[X]/(X^n+1)$, where q is the ciphertext modulus.
– Key Generation:
  - Secret Key (sk): A polynomial $\mathbf{s}$ with small coefficients.
  - Public Key (pk): Consists of two polynomials $([\mathbf{a}.\mathbf{s}+\text{Encode}(0,e)]_q, [-\mathbf{a}]_q)$, where $\mathbf{a}$ is random, and e being a small error polynomial. It represents the encryption of zero.
  - Relinearisation Key (rlk): $rlk = (k_1, k_2)$ is a series of ciphertexts that encrypt a function of the secret key. This is used to enable efficient ciphertext multiplication.

Most FHE have the same structure but just different encoding/decoding methods, here are some examples

– BFV: $\mathit{Encode}(m,e) = \varDelta m + e, \varDelta = [q/t], \text{Decode}(x) = [x/\varDelta]$
– BGV: $\mathit{Encode}(m,e) = m + te, \text{Decode}(x) = x \mod t$
– CRT-FHE [22]: $\mathit{Encode}(m,e) = CRT_{p_1,p_2}(m,e), \text{Decode}(x) = x \mod p_1$

Encryption :
To encrypt a plaintext $m \in \mathbb{Z}_t[X]/(X^n+1)$:

– Choose a random small polynomial $\mathbf{u}$ and two small error polynomials e_1 and e_2.
– Compute the ciphertext $c = (c_0, c_1)$ as:

$$c_0 = \mathbf{pk}[0] \cdot \mathbf{u} + \text{Encode}(m, e_1) \mod q$$
$$c_1 = \mathbf{pk}[1] \cdot \mathbf{u} + \text{Encode}(0, e_2) \mod q$$

Decryption To decrypt a ciphertext $c = (c_0, c_1)$:

– Compute:
$$m' = c_0 + c_1 \cdot \mathbf{s} \mod q$$
– Decode
$$m = \text{Decode}(m')$$

Homomorphic Operations

Addition Given two ciphertexts $c_1 = (c_{1,0}, c_{1,1})$ and $c_2 = (c_{2,0}, c_{2,1})$:

– Compute the sum:

$$c_{\text{sum}} = (c_{1,0} + c_{2,0}, c_{1,1} + c_{2,1})$$

– The resulting ciphertext c_{sum} encrypts the plaintext $m_{\text{sum}} = m_1 + m_2$.

Multiplication Given two ciphertexts $c_1 = (c_{1,0}, c_{1,1})$ and $c_2 = (c_{2,0}, c_{2,1})$:

– Compute the product terms:

$$d_0 = c_{1,0} \cdot c_{2,0}, d_1 = c_{1,0} \cdot c_{2,1} + c_{1,1} \cdot c_{2,0}, d_2 = c_{1,1} \cdot c_{2,1}$$

– The resulting ciphertext is a three-tuple (d_0, d_1, d_2), which needs to be relinearized to a two-tuple to maintain the structure of the ciphertext.

Relinearization Relinearization is used after multiplication to convert a three-tuple ciphertext back into a two-tuple form. This involves using additional keys, known as relinearization keys (k_1, k_2), to reduce the degree of the ciphertext.

– Use relinearization keys to compute:

$$c_{\text{rel}} = (d_0 + k_1 \cdot d_2, d_1 + k_2 \cdot d_2)$$

– The resulting ciphertext c_{rel} is a valid ciphertext that encrypts the product of the original plaintexts.

Security

The security of the FHE schemes relies on the hardness of the RLWE problem. The assumption is that it is computationally infeasible to distinguish between samples from the RLWE distribution and uniformly random samples.

The ability to perform addition, subtraction, and multiplication on encrypted data makes the FHE schemes suitable for a wide range of applications requiring data privacy and security.

2.4 Multi-Key FHE

In the context of Ring Learning with Errors (RLWE), Multi-Key Fully Homomorphic Encryption (MK-FHE) can be described as follows:

Multi-Key Operation: Operations on ciphertexts encrypted under multiple keys $s_1, s_2, \ldots, s_k$ produce a ciphertext under the combined key $s_{\text{agg}} = (s_1, s_2, \ldots, s_k)$.

$$(c_1 s_1 + \cdots + c_k s_k + \text{Encode}(m, e), -c_1, \ldots, -c_k)$$

$$\left(\sum_{i=1}^{k} c_i \cdot s_i + \text{Encode}(m, e), -c_1, \ldots, -c_k \right)$$

Decrypting this ciphertext requires the combined secret keys or collaborative decryption process to recover the plaintext:

$$m = \text{Decode}\left(c_0 + \sum_{i=1}^{k} c_i \cdot s_i\right)$$

This structure maintains the privacy of individual keys while enabling joint computations. This process ensures that the individual secrets s_i are never revealed during decryption, maintaining privacy while allowing joint computation on encrypted data.

Multi-Key FHE in the RLWE framework is powerful because it enables collaborative computations while preserving the privacy of each participant's data. The ring structure and RLWE problem underpin the security and efficiency of this scheme, making it suitable for various secure multi-party applications.

2.5 Problems with Traditional Multi-Key FHE

– *Ciphertext Size Explosion:*

Tensor Product of Keys: In traditional Multi-Key FHE (MK-FHE), when ciphertexts encrypted under different keys $s_1, s_2, \ldots, s_k$ are multiplied, the resulting ciphertext is encrypted under the tensor product of these keys, leading to an increase in size. Single Key Ciphertext

$$ct = (c_0, c_1)$$

Multi Key Ciphertext

$$ct = (c_0, c_1, ..., c_k)$$

Ciphertext size grows linearly with increasing number of users. This growth slows down homomorphic operations and increases storage requirements.

– *Storage and Communication Overhead*:

The larger ciphertext size Size(CT) not only demands more storage but also increases the communication overhead, particularly in applications where ciphertexts need to be transmitted across networks. This overhead is proportional to the number of keys k involved, given by:

$$\text{Overhead} \propto k \times \text{Size}(CT)$$

– *Re-linearization Complexity:*

Re-linearization Process: After ciphertext multiplication in MK-FHE, the resulting ciphertext must be reduced to a standard form, a process known as re-linearization. This involves reducing the expanded key space back to a manageable size, typically requiring additional keys (re-linearization keys rk). If the original keys are s_1, s_2, the re-linearization process reduces the tensor product:

$$s_1 \otimes s_2 \rightarrow \text{Relin}(s_1 \otimes s_2, rk)$$

This operation incurs significant computational overhead, often quantified as:

$$\text{Complexity}_{\text{Relin}} \approx O(N \cdot k^2)$$

where N is the polynomial degree in the ring $\mathbb{R}$, and k is the number of keys.

– *Increased Computational Complexity:*

Key Aggregation: In MK-FHE, when performing operations on ciphertexts encrypted under multiple keys, the keys are aggregated into a combined key $s_{\text{agg}} = (s_1, s_2, \ldots, s_k)$. This aggregation increases the complexity of encryption and decryption processes, which can be expressed as:

$$\text{Complexity}_{\text{Enc/Dec/Addition}} \propto O(k)$$

Ciphertext Growth: The ciphertext growth resulting from multi-key operations can be modeled by:

$$\text{Size}(CT_{\text{sum}}) = \text{Size}(CT_1) + \text{Size}(CT_2) + \cdots + \text{Size}(CT_k)$$

This growth slows down homomorphic operations and increases storage requirements.

– *Decryption Complexity:*

Collaborative Decryption: Decrypting a ciphertext in MK-FHE requires the collaboration of all participating users, where each user contributes their secret key share. The decryption of a ciphertext CT under aggregated key $s_{\text{agg}} = (s_1, s_2, \ldots, s_k)$ requires:

$$m = \left(c_0 + \sum_{i=1}^{k} c_i \cdot s_i \right) \mod t$$

This process adds complexity, particularly in distributed systems, and increases the risk of inefficiency if any party is unavailable.

– *Performance Overhead:*
  - Noise Growth: As operations are performed on ciphertexts in MK-FHE, the noise level within the ciphertexts tends to grow more rapidly than in single-key FHE. The noise growth can be modeled as:

    $$\text{Noise}_{\text{new}} = \text{Noise}_{\text{old}} + \text{Noise}_{\text{add/mul}}$$

    where the additional noise $\text{Noise}_{\text{add/mul}}$ increases with the number of keys k involved. This necessitates more frequent bootstrapping or noise management operations, impacting performance.

- Execution Time: The execution time for homomorphic operations in MK-FHE is also affected by the number of keys, as the complexity of managing multiple keys and larger ciphertexts increases. The execution time T_{exec} can be expressed as:

$$T_{\text{exec}} \propto O(k \cdot \log(N))$$

where N is the polynomial degree, and k is the number of keys.

– *Implementation Complexity:*
- System Complexity: Implementing MK-FHE requires careful management of key distribution, noise growth, and secure decryption protocols. The system complexity can be quantified as a function of the number of participants and the operations they perform:

$$\text{Complexity}_{\text{System}} \propto O(k \cdot N \cdot \log(N))$$

where k is the number of keys and N is the polynomial degree.
- Error Propagation: Errors from any party's contributions can propagate throughout the system, potentially leading to incorrect results or necessitating complex error correction mechanisms. This can be expressed as an increase in the probability of error $P(\text{Error})$ with the number of keys:

$$P(\text{Error}) \propto O(k \cdot \text{Error}_{\text{single}})$$

These challenges demonstrate the inherent trade-offs between the flexibility of MK-FHE and the increased computational and implementation complexity.

In the next sections we described the a novel approach for encrypted data sharing and computation that overcomes these challenges.

3 Core Ideas and Analysis: Multi-Key Fully Homomorphic Encryption Protocol

In multi-party computing scenarios, Fully Homomorphic Encryption (FHE) enables secure computation on encrypted data from multiple users without revealing their individual inputs. Traditional multi-key FHE schemes allow each user to independently generate and encrypt data using their own secret and public keys, but this introduces significant overhead during homomorphic operations. The proposed novel approach for multi-key FHE protocol addresses these inefficiencies by introducing a collaborative key exchange mechanism to compute a common group public key and relinearization key, enabling more efficient homomorphic operations on data encrypted by multiple parties. This approach not only preserves the privacy of individual participants but also simplifies the process of computation and decryption, ensuring that no single party has access to the aggregated group secret key while allowing secure decryption through a collaborative process.

Phase 1: Key Generation In the key generation phase, each user independently generates a private secret key s_j, which remains confidential. The corresponding public key pk_j is derived from the secret key and made publicly available. A "Group Public Key" (gpk) is then generated from the individual public keys of all users. This gpk corresponds to a hypothetical "Group Secret Key" (gsk), which is an aggregation of the individual secret keys, but crucially, the group secret key is never explicitly computed or accessible to any user. This ensures that while the group public key can be used for encryption, the underlying group secret key remains unknown.

Each user's public key is derived from their secret key as follows:

$$pk_j = ([a_j(L_j s_j) + \text{Encode}(0, e_j)]_q, -a_j) \quad \text{for } j = 1, 2, \ldots, k$$

where $\text{Encode}(m, e)$ is an encoding function. In the BFV scheme, for instance, $\text{Encode}(m, e) = \Delta m + e$ where $\Delta = \lfloor q/t \rfloor$, and in the BGV scheme, $\text{Encode}(m, e) = te + m$, with t being the plaintext modulus and q the ciphertext modulus.

The aggregated group secret key can be expressed as follows

$$s = \sum_{j=1}^{k} L_j s_j$$

This group secret key corresponds to a group public key gpk, which can be written as an encryption of zero under the group secret key:

$$gpk = ([as + \text{Encode}(0, e)]_q, -a)$$

Substituting s, the group public key becomes:

$$gpk = \left(\left[a \sum_{j=1}^{k} s_j L_j + \text{Encode}(0, e) \right]_q, -a \right) = \sum_{j=1}^{k} ([a s_j L_j + e_j]_q, -a)$$

Thus, the group public key can be computed as the sum of individual public keys. The key exchange process ensures that none of the participants need to share their secret keys or rely on a trusted third party.

Phase 2: Generation of Relinearization Key Relinearization is a key operation in FHE to reduce the size of ciphertexts after multiplicative operations. In this context, the relinearization key is an encryption of the square of the group secret key, s^2, further optimized as the encryption of $w^l s^2$ for different powers l.

The relinearization key is defined as:

$$rlk = \left\{ E(w^l s^2) \right\}_{l=0}^{\lfloor \log_w(q) \rfloor}$$

where the square of the secret key is given by:

$$s^2 = \left(\sum_{j=1}^{k} L_j s_j\right)^2 = \sum_{j=1}^{k} (L_j s_j)^2 + 2 \sum_{1 \le i < j \le k} L_i L_j s_i s_j$$

Since no single user possesses the group secret key, another key exchange process is required to compute the relinearization key. This is achieved in two steps:

1. Each user j encrypts and publishes $(L_j s_j)^2$ and $2L_i L_j s_j$ for $i \neq j$.
2. Each user i uses the published values to compute:

$$s_i * E(2L_i L_j s_j) + \text{Encode}(0, e_j) = E(2L_i L_j s_i s_j)$$

Once all this information is published, the relinearization key can be computed by each user as:

$$\sum_{j=1}^{k} E[w^l (L_j s_j)^2] + \sum_{1 \le i < j \le k} E(w^l 2L_i L_j s_i s_j)$$

$$= E\left(w^l \left(\sum_{j=1}^{k} L_j s_j\right)^2\right) = E(w^l s^2) \quad \text{for } l = 1, 2, \ldots, \lfloor \log_w(q) \rfloor$$

Thus, each user has access to both the group public key and the group relinearization key.

Phase 3: Encryption and Homomorphic Computation Once the group public key is established, each participant encrypts their data using *gpk*. Since all data is encrypted under a common key, homomorphic operations such as addition and multiplication can be performed on the encrypted data seamlessly, as in single-key FHE schemes.

The uniform encryption scheme allows for efficient computation across data from multiple participants without the additional complexity typically associated with multi-key FHE systems.

Phase 4: Decryption All encrypted data and computed results are encrypted with respect to the group secret key *gsk*, which no user possesses individually. This ensures that no user can unilaterally decrypt the data. To decrypt, a requesting user must gain approval from all other participants.

Each participant generates a Partial Decryption Key (PDK) for the specific ciphertext, ensuring that the PDK is ciphertext-specific and cannot be reused for other decryption requests.

For a ciphertext $ct = (c_0, c_1)$ encrypting a message m:

$$ct = ([as + \text{Encode}(m, e)]_q, -a)$$

Each user's partial decryption key for this ciphertext is:

$$pdk_j = [c_1 L_j s_j + \text{Encode}(0, e_j)]_q \quad \text{for } j = 1, 2, \ldots, k$$

After collecting all the PDKs, the requesting user can compute:

$$\left[c_0 + \sum_{j=1}^{k} pdk_j\right]_q = \left[c_0 + \sum_{j=1}^{k}(c_1 L_j s_j + \text{Encode}(0, e_j))\right]_q$$

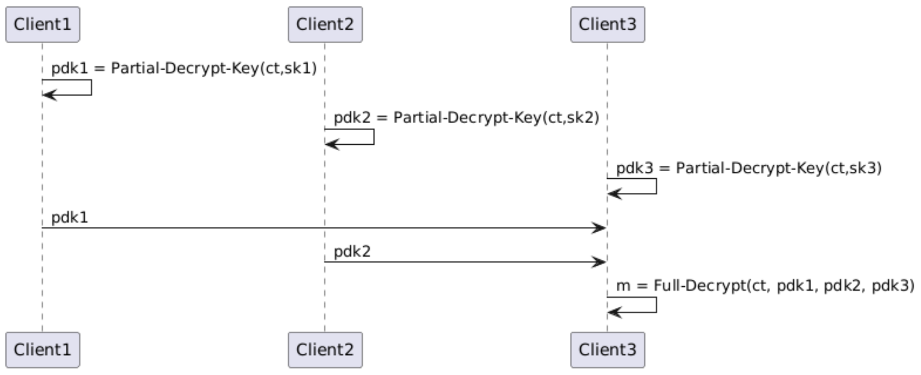
$$= \left[c_0 + c_1 \sum_{j=1}^{k} L_j s_j + \sum_{j=1}^{k} \text{Encode}(0, e_j)\right]_q = \left[c_0 + c_1 s + \sum_{j=1}^{k} \text{Encode}(0, e_j)\right]_q$$

$$= \text{Encode}(m, e)$$

The message m can then be recovered by decoding Encode(m, e).

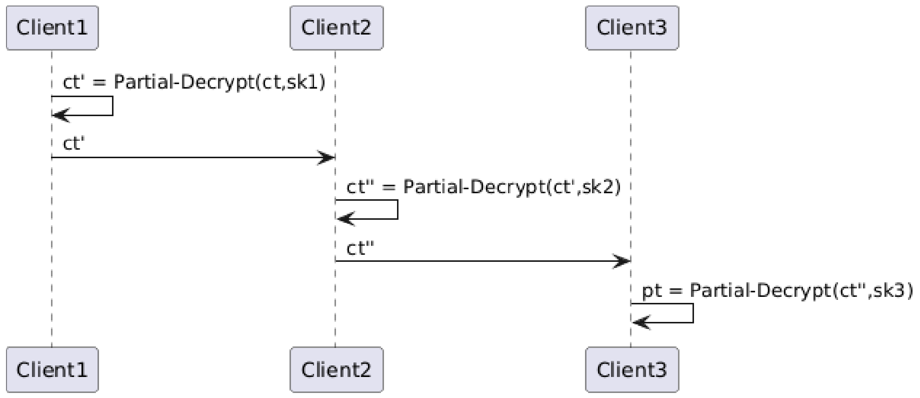
3.1 Difference between Old and New Decryption Method

New Decryption Diagram The new decryption method is described in the 3.1


Client1
Client2
Client3
pdk1 = Partial-Decrypt-Key(ct,sk1)
pdk2 = Partial-Decrypt-Key(ct,sk2)
pdk3 = Partial-Decrypt-Key(ct,sk3)
pdk1
pdk2
m = Full-Decrypt(ct, pdk1, pdk2, pdk3)
Client1
Client2
Client3

New Decryption Diagram

Old Decryption Diagram The old decryption method is described in the 3.1


Client1
Client2
Client3
ct' = Partial-Decrypt(ct,sk1)
ct'
ct'' = Partial-Decrypt(ct',sk2)
ct''
pt = Partial-Decrypt(ct'',sk3)
Client1
Client2
Client3

Old Decryption Diagram

In the traditional method, users must perform the "Partial Decryption" process sequentially, with each user waiting for the previous one to complete before beginning their own decryption. This limitation prevents parallelization and can slow down the decryption process. In contrast, the new method allows all users to generate their partial decryption keys simultaneously, enabling parallel processing. This significantly improves efficiency and accelerates the overall decryption process.

This multi-key FHE scheme allows each participant to independently generate secret and public keys while using a group key exchange protocol to derive a common group public key and relinearization key. Data from different users, encrypted under the same group public key, can be homomorphically computed on efficiently. Decryption requires collaboration, ensuring privacy and security, as no individual can decrypt the data alone.

4 Construction

This section provides a detailed framework of the proposed Multi-Key Fully Homomorphic Encryption (MK-FHE) scheme, which enables secure multi-party computations without compromising individual user privacy. It describes the essential building blocks of the system, beginning with independent secret key generation and the collaborative formation of a group public key, and group relinearisation key. The section also covers the encryption mechanism, relinearization for efficient homomorphic multiplication, and the decryption process. By combining these components, the proposed construction allows for efficient, non-interactive computations on encrypted data while maintaining strong security guarantees.

4.1 Terminologies

Consider a scenario where k users $U_1, U_2, ..., U_k$ wish to perform computation on their combined data, but they do not want to expose their data to each other.

Public Parameters:

– Ciphertext Modulus: q
– Plaintext Modulus: t
– RLWE Dimention: n
– Base and Level: $(w, L = \lfloor \log_w(q) \rfloor)$
– Plaintext Space: A polynomial ring $\mathbb{Z}_t[X]/(X^n + 1)$, where t is the plaintext modulus.
– Ciphertext Space: Polynomials in $\mathbb{Z}_q[X]/(X^n + 1)$, where q is the ciphertext modulus.
– Participating Users: $U_1, U_2, ..., U_k$

All users must agree to use the same set of parameters including the *Plaintext Space* and *Ciphertext Space.*

4.2 User Key Generation

In the key generation process, all participating users must agree on a common mask $a^* \in \mathbb{Z}_q[X]/(X^n + 1)$ and a shared set of values $(L_1, ..., L_k)$. These parameters will be used in the generation of public keys by each participant.

– Common Parameters for Key Generation:

$$[a^*, (L_1, ..., L_k)]$$

Generating Secret Key Each participating user independently generates a unique secret key that remains private. This ensures that no user shares their secret key with others.

– For $i = 1, 2, \ldots, k$
  - User U_i generates a secret key $s_i \in \mathbb{Z}_q[X]/(X^n + 1)$

Generating User Public Key Each user independently generates their own public key using their respective secret key and the agreed-upon common parameters.

Each user U_i performs the following steps to compute the public key

– Input: $s_i, [a^*, (L_1, ..., L_k)]$
– Output: pk_i

Steps:

– Compute:

$$pk_i = ([a^*(L_i s_i) + \text{Encode}(0, e_i)]_q, [-a^*]_q)$$

where e_i is a small error polynomial.

– Return pk_i

The public key pk_i represents an encryption of zero. All users then publish their public keys to the group.

$$(pk_i = [pk_i[0],\ pk_i[1]])_{i=1}^k$$

Users→	U_1	U_2	...	U_k
Secret Key →	s_1	s_2	...	s_k
Public Key →	pk_1	pk_2	...	pk_k

Table 2. User Keys

4.3 Generating Group Public Key

The *Group Public Key gpk* represents an encryption of zero with respect to a *Group Secret Key s* , which is effectively a combination of the individual secret keys from all participants. However, at no point during the process is the group secret key explicitly computed or revealed, as the users do not share their secret keys with each other. Although the group secret key remains hidden, the *Group Public Key gpk* is derived using the individual public keys of the users, without requiring access to their secret keys.

Algorithm: Generate Group Public Key

- Input: Public Keys of all user: $(pk_1, pk_2, ..., pk_k)$
- Output: Group Public Key gpk
- Steps:
  - Compute the sum of the first component of each user's public key modulo q:
$$gpk = (\sum_{i=1}^{k} pk_i[0] \mod q, pk_1[1])$$
  - Return gpk

The first component of gpk can be expanded as:

$$\sum_{i=1}^{k} pk_i[0] \mod q = \sum_{i=1}^{k} a^*(L_i s_i) + \text{Encode}(0, e_i) \mod q$$

$$= a^*(\sum_{i=1}^{k} L_i s_i) + \text{Encode}(0, \sum_{i=1}^{k} e_i) \mod q = [a^* s + \text{Encode}(0, e^*)]_q$$

where $e^* = \sum_{i=1}^{k} e_i$. Thus, the group public key is:

$$gpk = ([a^* s + \text{Encode}(0, e^*)]_q, [-a^*]_q)$$

Hence, the computed *Group Public Key* is indeed an encryption of zero with respect to the *Group Secret Key s*, ensuring both correctness and security in the multi-user homomorphic encryption setting.

4.4 Generating Re-linearisation Key

The relinearization key facilitates efficient homomorphic multiplication by allowing ciphertexts to be relinearized after multiplication operations. Specifically, the relinearization key is an encryption of the square of the *Group Secret Key* s^2 under the same *Group Secret Key* s.

Let the encryption of a plaintext m under the secret key s be denoted as:

$$E(m, s) = ([a^* s + m + \text{Encode}(0, e)]_q, [-a^*]_q)$$

For simplicity, we will use the notation $E(m)$ for $E(m, s)$. Consequently, the relinearization key rlk can be expressed as:

$$rlk = \left\{E(w^l \cdot s^2)\right\}_{l=0}^{\lfloor \log_w(q) \rfloor}$$

where w is a parameter used for encoding powers of the group secret key s^2.

Since no single user holds the complete group secret key s, the generation of the relinearization key requires a collaborative key exchange process. This process is divided into two phases, each generating auxiliary keys that are later combined to produce the final relinearization key.

Steps

- Key Exchange -1:
  - Each participant generates the first set of auxiliary keys, termed "Auxiliary Keys - 1" and share them with all users. This can include encryption of $(L_j s_j)^2$ and $2L_i L_j s_j$.
- Key Exchange -2:
  - Following the first exchange, participants generate the second set of auxiliary keys, termed "Auxiliary Keys - 2". The shared "Auxiliary Keys - 1" are used in the process. This can include computing encryption of $2L_i s_i . L_j s_j$ by multiplying s_i as plaintext to encryption of $2L_i L_j s_j$. These keys are also published.
- Compute the Relinearisation Key
  - The auxiliary keys from both exchanges are combined to compute the relinearization key rlk.

This process ensures that the relinearization key is computed securely without any party needing access to the full group secret key, maintaining the security of the system while enabling efficient homomorphic operations.

4.5 Key Exchange -1

For each user U_j, where $j = 1, 2, \ldots, k$, the following steps are performed to generate the first set of auxiliary keys, denoted as Auxiliary Key - 1 R_j.

Auxiliary Key -1 [For each User $U_j, j = 1, 2, ..., k$,]

- Input:
  - Secret Key: s_j
  - Group Pubic Key: *gpk*
- Output:
  - Auxiliary Key -1 R_j
- **Steps**:
  - Compute the Encryption of $(L_j s_j)^2$ using the group public key

$$R_j[0] = (E(w^l (s_j L_j)^2))_{l=0}^{\lfloor \log_w(q) \rfloor}$$

  This represents the encrypted version of $(L_j s_j)^2$, where w^l is used for encoding powers of $(L_j s_j)^2$.

  - Compute the Encryption of $2 s_j L_i L_j$ for all other users $i \neq j$ using the Group Public Key:

$$R_j[1] = \{(E(w^l 2 s_j L_i L_j))_{l=0}^{\lfloor \log_w(q) \rfloor} : i = 1, ..., k, i \neq j\}$$

  - Return and Publish R_j

$$R_j = (R_j[0], R_j[1])$$

So

$$R_j = \Big((E(w^l (s_j L_j)^2))_{l=0}^{\lfloor \log_w(q) \rfloor},\ \{\{E(w^l 2 s_j L_i L_j)\}_{l=0}^{\lfloor \log_w(q) \rfloor} : i = 1, ..., k, i \neq j\} \Big)$$

This completes the generation of Auxiliary Key - 1 for user U_j, ensuring secure and encrypted contributions towards the relinearization process.

Users→	U_1	U_2	...	U_k
Aux Keys →	$E(w^l (s_1 L_1)^2)$	$E(w^l (s_2 L_2)^2)$	...	$E(w^l (s_k L_k)^2)$

Table 3. Square Terms

	U_1	U_2	...	U_k
U_1	x	$E[w^l 2 L_1 L_2 s_1]$	...	$E[w^l 2 L_1 L_k s_1]$
U_2	$E[w^l 2 L_2 L_1 s_2]$	x	...	$E[w^l 2 L_2 L_k s_2]$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
U_k	$E[w^l 2 L_k L_1 s_k]$	$E[w^l 2 L_k L_2 s_k]$	...	x

Table 4. Cross Multiplied Terms

4.6 Key Exchange -2

For each user U_j, where $j = 1, 2, \ldots, k$, the following steps are performed to generate the second set of auxiliary keys, denoted as Auxiliary Key - 2 R_j.

Auxiliary Key -2 [For User j]

- Input:
  - Auxiliary Key -1: $R[1] = \{\{E(w^l * 2s_iL_iL_j)\}_{l=0}^{\lfloor \log_w(q) \rfloor} : i = 1, 2, ..., k\}$
  - Secret Key: s_j
- Steps:
  - Compute the following for each user $i = 1, 2, \ldots, k$:

$$R[2] = \left\{ \left(s_j * E(w^l * 2s_iL_iL_j) + \text{Encode}(0, e_j)\right)_{l=0}^{\lfloor \log_w(q) \rfloor} : i = 1, 2, ..., k \right\}$$

By multiplying the encrypted term $E(w^l \cdot 2s_iL_iL_j)$ by s_j, user j computes the auxiliary contribution for the second key exchange phase. Further optimization techniques like gadget decomposition and powers of 2 can be utilised to optimise the multiplication. Simplify the expression:

$$R[2] = \left\{ \left(E(w^l * 2(L_is_i)(L_js_j))\right)_{l=0}^{\lfloor \log_w(q) \rfloor} : i = 1, 2, ..., k \right\}$$

This represents the encrypted interaction term $2(L_is_i)(L_js_j)$, reflecting the combined contribution of both users' secret keys.

The resulting auxiliary key $R[2]$ will be used in the final computation of the relinearization key, ensuring that the process is securely performed without exposing individual secret keys.

	U_1	U_2	...	U_k
U_1	x	$E(w^l 2L_1s_1L_2s_2)$	...	$E(w^l 2L_1s_1L_ks_k)$
U_2	$E(w^l 2L_2s_2L_1s_1)$	x	...	$E(w^l 2L_2s_2L_ks_k)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
U_k	$E(w^l 2L_ks_kL_1s_1)$	$E(w^l 2L_ks_kL_2s_2)$	...	x

Table 5. Cross Multiplied Terms

4.7 Computing Relinearization Key from Auxiliary Keys

- Input:
  - Auxiliary Keys
    * $R[0] = \{\{E(w^l(L_is_i)^2)\}_{l=0}^{\lfloor \log_w(q) \rfloor} : i = 1, 2, ..., k\}$
    * $R[2] = \{\{E(w^l * L_is_iL_js_j)\}_{l=0}^{\lfloor \log_w(q) \rfloor} : i = 1, 2, ..., k\}$
- Output:

- Relinearization Key: rlk

Steps:

- Compute Homomorphic Addition of Squared terms

$$X = \left\{ \sum_{i=1}^{k} E(w^l (L_i s_i)^2) \right\}_{l=0}^{\lfloor \log_w(q) \rfloor}$$

- Compute Homomorphic Addition of Cross Multiplied terms

$$Y = \left\{ \sum_{1 \le i < j \le n} E(w^l \cdot L_i s_i \cdot L_j s_j) \right\}_{l=0}^{\lfloor \log_w(q) \rfloor}$$

- Combine the Results to Generate the Relinearization Key

$$rlk = X + Y$$

- Return rlk

The combined result can be expressed as:

$$X + Y = \left\{ E(w^l (L_i s_i)^2) + \sum_{1 \le i < j \le n} E(w^l * L_i s_i L_j s_j) \right\}_{l=0}^{\lfloor \log_w(q) \rfloor}$$

$$= \left\{ E(w^l s^2) \right\}_{l=0}^{\lfloor \log_w(q) \rfloor}$$

Thus, the relinearization key rlk is an encryption of s^2, allowing for secure homomorphic operations without revealing individual users' secret keys.

Set of all keys:

Users→	U_1	U_2	...	U_k
Secret Key →	s_1	s_2	...	s_k
Public Key →	pk_1	pk_2	...	pk_k
Group Public Key	$gpk = ([a^* s + \text{Encode}(0, e^*)]_q, [-a^*]_q)$			
Group Relin Key	$rlk = \left\{ E(w^l s^2) \right\}_{l=0}^{\lfloor \log_w(q) \rfloor}$			

Table 6. User Keys

4.8 Encryption

In this step, all users encrypt their data using the *Group Public Key (GPK)*, following the standard public key encryption method as outlined in single-key Fully Homomorphic Encryption (FHE) schemes.

Encrypt (m, gpk)

- Input:
  - Plain text: $m \in \mathbb{Z}_t[X]/(X^n + 1)$
  - Group Public Key: $gpk = (gpk[0], gpk[1])$
- Output:
  - Ciphertext: $c = (c_0, c_1)$

To encrypt a plaintext $m \in \mathbb{Z}_t[X]/(X^n + 1)$, the following steps are performed:

- Steps:
  - Choose a random small polynomial $\mathbf{u}$ and two small error polynomials e_1 and e_2.
  - Compute the ciphertext $c = (c_0, c_1)$:
    * The ciphertext $c = (c_0, c_1)$ is computed using the group public key $gpk = (gpk[0], gpk[1])$ as follows:

$$c_0 = gpk[0] \cdot \mathbf{u} + \text{Encode}(m, e_1) \mod q$$

$$c_1 = gpk[1] \cdot \mathbf{u} + \text{Encode}(0, e_2) \mod q$$

The Encode() function is defined according to the specific FHE scheme being used, ensuring the correct transformation of plaintext and error terms into ciphertext form while maintaining homomorphic properties.

4.9 Decryption

Decryption in this system requires collaboration among participants, ensuring that no single user can independently decrypt the data, thus maintaining privacy and security. Each participant generates a *Partial Decryption Key (PDK)* specific to the given ciphertext, ensuring that the key is unique and cannot be reused for decryption of other ciphertext. Once all the partial decryption keys are collected from the participants, the requesting user can proceed with the full decryption.

Each user U_j generates the partial decryption key as follows:

Generation of Partial Decryption Key (ct, s_j)

- Input:
  - Ciphertext: $ct = (c_0, c_1)$
  - Secret Key of User U_j: s_j
- Output:
  - Partial Decryption Key pdk_j

Steps:

– Compute

$$pdk_j = [c_1 L_j s_j + \text{Encode}(0, e_j)]_q \quad \text{for}$$

where e_j is a small error term, and L_j is the parameter associated with user U_j.

Once all the partial decryption keys are collected from the participants, the requesting user can proceed with the full decryption.

Full Decryption $(ct, \{pdk_j : j = 1, 2, ..., k\})$

– Input:
  - Ciphertext: $ct = (c_0, c_1)$
  - Set of Partial Decryption Keys: $\{pdk_1, pdk_2, ..., pdk_k\}$

– Output:
  - Plain text: m

Procedure:

– Compute the intermediate result

$$m^* = \left[c_0 + \sum_{j=1}^{k} pdk_j \right]_q$$

– Decode the message

$$m = \text{Decode}(m^*)$$

The intermediate value m^* simplifies as follows:

$$m^* = \left[c_0 + \sum_{j=1}^{k} pdk_j \right]_q = \left[c_0 + \sum_{j=1}^{k} (c_1 L_j s_j + \text{Encode}(0, e_j)) \right]_q$$
$$= \left[c_0 + c_1 s + \sum_{j=1}^{k} \text{Encode}(0, e_j) \right]_q = \text{Encode}(m, e)$$

where s is the *Group Secret Key.*

Finally, the plaintext m can be recovered by decoding $m^* = \text{Encode}(m, e)$, yielding the original message.

In the traditional method, users must perform the "Partial Decryption" process sequentially, with each user waiting for the previous one to complete before beginning their own decryption. This limitation prevents parallelization and can slow down the decryption process. In contrast, the new method allows all users to generate their partial decryption keys simultaneously, enabling parallel processing. This significantly improves efficiency and accelerates the overall decryption process.

4.10 Homomorphic Operations

Since all data is encrypted under the group public key, homomorphic operations such as addition and multiplication can be performed on the encrypted data seamlessly, as in single-key FHE schemes.

The uniform encryption scheme allows for efficient computation across data from multiple participants without the additional complexity typically associated with multi-key FHE systems.

Addition Given two ciphertexts $c_1 = (c_{1,0}, c_{1,1})$ and $c_2 = (c_{2,0}, c_{2,1})$:

1. Compute the sum:

$$c_{\text{sum}} = (c_{1,0} + c_{2,0}, c_{1,1} + c_{2,1})$$

2. The resulting ciphertext c_{sum} encrypts the plaintext $m_{\text{sum}} = m_1 + m_2$.

Multiplication Given two ciphertexts $c_1 = (c_{1,0}, c_{1,1})$ and $c_2 = (c_{2,0}, c_{2,1})$:

1. Compute the product terms:

$$d_0 = c_{1,0} \cdot c_{2,0},\ d_1 = c_{1,0} \cdot c_{2,1} + c_{1,1} \cdot c_{2,0},\ d_2 = c_{1,1} \cdot c_{2,1}$$

2. The resulting ciphertext is a three-tuple (d_0, d_1, d_2), which needs to be relinearized to a two-tuple to maintain the structure of the ciphertext.

Relinearization Relinearization is used after multiplication to convert a three-tuple ciphertext back into a two-tuple form. This involves using the group relinearization keys generated in the key generation process, to reduce the degree of the ciphertext.

5 Security Analysis

The proposed cryptographic scheme introduces a novel approach to secure multiparty computations through the Multi-Key Fully Homomorphic Encryption (MK-FHE). In this section, we provide a detailed security analysis of the system, considering key security properties such as confidentiality, integrity, and resilience against various cryptographic attacks. The security of the scheme is primarily based on the hardness of the Ring Learning with Errors (RLWE) problem, a well-established foundation in lattice-based cryptography.

5.1 Basic Analysis: User Secret Key Confidentiality

In this scheme, each user generates their own secret key independently, without any need to share their key with others at any point during the process. This isolation of secret keys ensures that no party can infer another user's key, enhancing confidentiality and minimizing the attack surface. The decentralized nature of secret key generation ensures that even if communication channels between users are compromised, the secret keys remain protected.

The process of secret key generation and usage follows strong principles of cryptographic security, as users encrypt their data under their own secret key. Since no secret keys are shared, the system inherently protects each user's key from exposure to adversaries, even if some parties in the system are compromised.

5.2 Semantic Security: RLWE-based Cryptosystem

The security of the entire system relies on the RLWE (Ring Learning with Errors) cryptosystem, a lattice-based encryption technique known for its resistance to both classical and quantum attacks. Since all protocols, including encryption, key exchange, and homomorphic operations, are built on RLWE, the semantic security of the proposed scheme is directly tied to the hardness of the RLWE problem. As RLWE is a hard problem, breaking it is computationally infeasible under standard cryptographic assumptions, ensuring the system is secure against known attacks.

The RLWE problem can be formulated as follows: given a tuple $(b = a \cdot s + e, -a)$, where a is uniformly random from a ring, s is the secret, and e is a small error, it is computationally infeasible to distinguish $(b, -a)$ from a random pair. This underpins the semantic security of the system, which ensures that ciphertexts are indistinguishable from random noise, barring knowledge of the secret key s.

Given that the encryption is based on the hardness of RLWE, an adversary's advantage in breaking the encryption scheme is negligible, provided the underlying parameters are chosen correctly. The RLWE assumption guarantees that recovering the plaintext from the ciphertext is as hard as solving the RLWE problem, which is considered intractable under both classical and quantum computational models:

$$c_0 = as + \text{Encode}(m, e_1) \mod q$$

This guarantees that an adversary who intercepts ciphertexts will not be able to learn any meaningful information about the plaintexts, as the ciphertexts are indistinguishable from random noise without knowledge of the decryption key. Hence, the semantic security of the system is as strong as that of the underlying RLWE crypto system.

5.3 Key Exchange/Key Generation Security

Public Key Generation The key exchange protocol ensures that each user's contribution to the group public key gpk remains secure. Users do not share their secret keys s_j, and instead, contribute masked values based on the common mask a^*, ensuring security throughout the process.

$$pk_j = (a^* \cdot s_j + \text{Encode}(0, e_j) \mod q, -a^*)$$

where a^* is a common random element, ensuring that no individual user's secret key is exposed during the process. The public key is secure because each contribution is masked by a^*, and the randomness of e_j and the RLWE assumption ensures that no adversary can infer any participant's secret key from the public key or group public key gpk.

The group public key $gpk = (gpk[0], gpk[1])$ through a combination of independent public keys, with no direct exposure of any individual secret key during the process. The group public key acts as a composite of user contributions without revealing individual secrets.

Thus, the group secret key or any user secret key cannot be extracted from the key generation process, even in the presence of an adversary capable of observing the entire public key generation exchange.

Relinearization Key Generation The relinearization key rlk is generated to enable efficient homomorphic multiplication by transforming a product of ciphertexts into a form compatible with the scheme's encryption process. In the proposed scheme, the relinearization key is the encryption of a function of group secret key under the group secret key, ensuring that it does not leak any sensitive information during the computation.:

$$rlk = E(s^2)$$

where s is the group secret key. The encryption of the relinearization key ensures that no sensitive information about s is revealed.

– **Security of Auxiliary Keys:** The auxiliary keys used for relinearization are encrypted under the group secret key making them inaccessible to any party that does not possess the group secret key. If the group secret key s remains confidential, the auxiliary keys also remain secure. Thus, any operation using these keys does not leak information about individual secret keys.

This ensures that relinearization does not introduce vulnerabilities and that the security of homomorphic operations is maintained throughout the computation process.

5.4 Decryption Security

Partial Decryption Key Confidentiality: During the decryption process, each participant generates a partial decryption key specific to their secret key and the ciphertext.

In the decryption phase, each user U_j generates a partial decryption key (PDK) pdk_j specific to the ciphertext $ct = (c_0, c_1)$. The PDK is computed as:

$$pdk_j = [c_1 L_j s_j + \text{Encode}(0, e_j)]_q$$

Again the RLWE hardness assumption ensures that partial decryption keys do not reveal the underlying user secret key. This property is critical for multi-party decryption, as it prevents any user or external adversary from learning another participant's secret key through decryption operations.

Ciphertext-specific Decryption Keys: Partial decryption keys are tied to specific ciphertexts, ensuring that even if a partial decryption key is intercepted, it cannot be reused for other ciphertexts. This ciphertext-specific binding ensures that decryption is secure and that leakage from one decryption operation does not compromise other ciphertexts or decryption keys.

Mandatory Collaboration: Once all PDKs are collected, the final decryption is performed by summing them with c_0:

$$m^* = \left[c_0 + \sum_{j=1}^{k} pdk_j \right]_q$$

The plaintext m is then recovered through the decryption process:

$$m = \text{Decode}(m^*)$$

This structure guarantees that no single participant can decrypt the ciphertext alone, and each PDK does not reveal the underlying secret keys, preserving the confidentiality of individual users.

5.5 Other Security Practices

Secure Channel Usage To further enhance security, the system employs secure communication channels for all key exchanges, including the generation of the group public key, relinearization key, and partial decryption keys. By using secure channels, the scheme ensures that any information exchanged during these critical phases is protected against eavesdropping, man-in-the-middle attacks, and other network-based threats.

Using secure channels prevents adversaries from tampering with or intercepting the messages exchanged during the key exchange and decryption processes, thereby maintaining the integrity and confidentiality of the key generation and usage protocols.

5.6 Summary

The security of the proposed MK-FHE scheme is rooted in several key principles:

- *User Secret Key Confidentiality*: Users generate and maintain independent secret keys.

– *RLWE-based Security*: The encryption scheme is built on the hardness of the RLWE problem.
– *Key Generation and Decryption*: The use of auxiliary and partial decryption keys ensures security during operations, while ciphertext-specific keys protect against reuse.
– *Secure Channels*: The use of encrypted communication ensures that no sensitive information is leaked during exchanges.

Together, these principles ensure that the system remains resilient against cryptographic attacks while providing efficient and scalable secure multi-party computation. Overall, the system achieves a strong balance between security and efficiency, making it suitable for large-scale, collaborative encrypted data processing.

6 Performance Analysis and Key Improvements

The comparison table 7 highlights the key differences between traditional Multi-Key Fully Homomorphic Encryption (MK-FHE) and the proposed Multi-Key FHE scheme.

7 Applications and Use-cases

The proposed Multi-Key Fully Homomorphic Encryption (MK-FHE) scheme has the potential to revolutionize numerous industries by enabling secure, collaborative, and scalable data processing across multiple parties. The following applications and use cases highlight the practical impact of our approach:

1. Collaborative Cloud Computing As businesses and organizations increasingly rely on cloud-based platforms for data storage and computation, the need for secure processing of sensitive data becomes paramount. Our MK-FHE scheme allows multiple users to perform computations on encrypted data stored in the cloud without revealing their private inputs. This enables secure, joint data analysis and processing, with applications in:
   – *Secure Data Sharing*: Multiple organizations can collaborate on analyzing shared datasets (e.g., medical research, financial audits) while keeping sensitive data encrypted and private.
   – *Cross-Organizational AI/ML Models*: Organizations can collaboratively train machine learning models on combined datasets without revealing proprietary data, improving performance and insights in fields like predictive healthcare and fraud detection.
2. Privacy-Preserving Machine Learning The rise of artificial intelligence (AI) and machine learning (ML) has created a demand for privacy-preserving models, especially when using sensitive data such as healthcare records, financial transactions, or user behavior patterns. Our MK-FHE scheme enables secure, multi-party computation of encrypted data, allowing for:

Feature	Traditional Multi-Key FHE	Proposed Multi-Key FHE
Key Generation	Each participant generates a secret key and corresponding public key independently. Keys are combined through key aggregation.	Each participant generates a secret key and computes an individual public key. Group public key and relinearization keys are computed using a customized key exchange protocol.
Ciphertext Size	Ciphertext size increases linerly with the number of users, causing significant overhead.	Constant and manageable: Maintains a fixed ciphertext size, independent of the number of participants, enhancing efficiency.
Re-linearization	Requires complex and resource-intensive re-linearization due to multiple key interactions.	Simplified process: Utilizes a group relinerisation key, drastically reducing the complexity and computational demands of re-linearization.
Noise Growth	Rapid noise growth as a result of complex relinearization requires frequent bootstrapping, degrading performance.	Controlled noise accumulation due to simple relinearisation, reducing the need for frequent bootstrapping.
Decryption	Collaborative decryption all parties to work sequentially one after another, which adds complexity and potential inefficiencies.	Decryption remains collaborative but is optimized for efficiency, uses parallel techniques.
Computational Complexity	Increased complexity due to key aggregation and large ciphertext sizes.	Lower computational complexity, with operations similar to single-key FHE due to customized key exchange protocol.
Storage and Communication Overhead	Substantial overhead from large ciphertexts, especially problematic in multi-party scenarios.	Minimal overhead: Optimized ciphertext sizes and efficient communication protocols significantly reduce storage and transmission costs.
Performance Overhead	Significant performance overhead due to complex operations, large ciphertexts, and frequent noise management.	Optimized performance: Maintains consistent efficiency even as participant numbers grow, avoiding the typical exponential performance degradation.
Scalability	Limited scalability due to exponential growth in ciphertext size and noise, hampering large-scale adoption.	Highly scalable: Easily supports an increasing number of participants without performance degradation, making it ideal for large-scale applications.

Table 7. Key differences between traditional Multi-Key Fully Homomorphic Encryption (MK-FHE) and the proposed Multi-Key FHE scheme

  - *Private AI Inference*: In scenarios where data owners need to perform inference using an external model (e.g., facial recognition or sentiment analysis), our scheme ensures that both the data and the model remain encrypted, preserving privacy for all parties.

   Federated Learning with Privacy: Multiple institutions can securely collaborate to build and train shared ML models without exposing individual datasets, crucial in sectors like healthcare (e.g., predicting disease outbreaks) and finance (e.g., anti-money laundering).
3. Financial Services The financial industry handles highly sensitive data that requires stringent privacy and security measures. Our MK-FHE scheme enhances privacy-preserving computations for various financial services, including:
   - *Anti-Money Laundering (AML)*: Banks and financial institutions can collaborate on detecting suspicious transactions across different data sources while keeping customer information private.
   - *Secure Risk Analysis*: Financial institutions can perform joint risk assessments on pooled, encrypted data to identify potential risks and trends while ensuring that private data is never exposed.
4. Healthcare and Genomic Research The healthcare sector increasingly relies on secure data sharing to improve diagnosis, treatment, and research. Our MK-FHE scheme enables privacy-preserving collaboration between hospitals, research institutions, and pharmaceutical companies, with applications in:
   - *Medical Research*: Multiple hospitals and research centers can jointly analyze patient data, such as genome sequences or clinical trial results, without compromising patient privacy. This could lead to breakthroughs in personalized medicine and drug development.
   - *Collaborative Diagnosis and Treatment*: Healthcare providers can collaboratively evaluate patient data while maintaining patient confidentiality, enabling better diagnostic models and treatment plans in privacy-sensitive scenarios.
5. Digital Marketing and Privacy-Preserving Advertising Our MK-FHE scheme allows for secure, privacy-preserving advertising strategies in digital marketing. Advertisers can target users based on encrypted data while ensuring that sensitive user information remains confidential. Potential use cases include:
   - *Privacy-Preserving Behavioral Targeting*: Advertisers can securely target users based on their behavior patterns without compromising their privacy, ensuring compliance with data privacy regulations like GDPR.
   - *Encrypted Analytics*: Businesses can run analytics on encrypted customer data to gain insights into consumer preferences and behavior while protecting personal information.
6. Privacy-Preserving Record Linkage In sectors like government, insurance, and social services, combining datasets from different organizations for analysis often requires linking records that share common attributes. Using our MK-FHE scheme, record linkage can be performed securely across encrypted datasets, ensuring:

– *Secure Data Integration*: Organizations can combine and analyze data without revealing sensitive details, useful in areas like social service assessments, fraud detection, and public health initiatives.
– *Anonymized Cross-Organizational Insights*: Government agencies or NGOs can collaborate on large-scale data analysis for policy-making, while ensuring citizen privacy and security.

The wide-ranging applications of our Multi-Key FHE scheme demonstrate its potential to reshape secure multi-party computation across industries. By enabling efficient and scalable encrypted data processing, our approach provides a foundation for privacy-preserving technologies that meet the growing demand for data security and collaboration. From healthcare to finance, AI to digital marketing, the proposed MK-FHE scheme enables new possibilities for privacy-conscious, secure data-sharing and computation at scale.

8 Conclusion

In this paper, we have introduced a novel approach to Multi-Key Fully Homomorphic Encryption (MK-FHE) that addresses the key challenges of scalability, efficiency, and security in multi-party encrypted computations. Our approach enables secure computation across multiple participants while maintaining constant ciphertext size and execution time, regardless of the number of users involved.

We demonstrated how our method overcomes the traditional limitations of MK-FHE, such as noise growth, ciphertext expansion, and complex relinearization, by employing a unified group key structure. This advancement enhances the scalability and practicality of homomorphic encryption systems, particularly in environments where the number of participants may increase over time, such as cloud computing, collaborative data analysis, and privacy-preserving machine learning.

The security of our scheme is grounded in the hardness of the Ring Learning with Errors (RLWE) problem, ensuring that the system is resistant to both classical and quantum attacks. Moreover, our security analysis confirmed that key generation, encryption, and decryption processes are designed to prevent leakage of sensitive information, even in the presence of adversaries.

Overall, this work represents a significant step forward in the field of homomorphic encryption, enabling more efficient, scalable, and secure frameworks for multi-party encrypted computations. By reducing computational complexity and improving performance, our approach opens new possibilities for practical applications of homomorphic encryption in various industries, including healthcare, finance, and digital marketing, where privacy and security are paramount.

References

1. Shaikha Al-Riyami et al. Privacy-preserving medical data sharing based on fully homomorphic encryption. *Journal of Biomedical Informatics*, 2018.

2. Michael Armbrust et al. A view of cloud computing. *Communications of the ACM*, 53:50–58, 2010.
3. Michael Ben-Or et al. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium*, 1988.
4. Ranjan Bhaskar and Edan Beig. *Cloud computing in Industry 4.0: Innovations and challenges*. Springer, 2022.
5. Peter Bogetoft et al. Secure multiparty computation goes live. *Financial Cryptography*, 2009.
6. Zvika Brakerski et al. Leveled fully homomorphic encryption without bootstrapping. In *ACM CCS*, 2014.
7. Hao Chen et al. Practical multi-key homomorphic encryption. In *ACM CCS*, 2017.
8. Jung Hee Cheon et al. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.
9. Ilaria Chillotti et al. Faster fully homomorphic encryption: Bootstrapping in less than a second. In *ASIACRYPT*, 2016.
10. Ivan Damgård et al. Multiparty computation. Technical report, University of Aarhus, 2012.
11. Léo Ducas et al. A blueprint for practical fully homomorphic encryption. In *IACR Cryptology*, 2015.
12. Zekeriya Erkin et al. Privacy-preserving data aggregation in financial networks. In *Financial Cryptography*, 2009.
13. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
14. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *EUROCRYPT*, 2013.
15. Oded Goldreich. *Foundations of cryptography: Volume 2*. Cambridge University Press, 2004.
16. Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. In *CRYPTO*, 2019.
17. Shai Halevi and Victor Shoup. Algorithms in fully homomorphic encryption: A survey. In *Proceedings of the Third Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, 2020.
18. Tancrède Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes fv and yashe. In *IACR Cryptology*, 2014.
19. Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1, 2009.
20. Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical report, National Institute of Standards and Technology (NIST), 2011.
21. Mathieu Mouchet et al. Multi-key fully homomorphic encryption: Challenges and progress. *Public Key Cryptography*, 2020.
22. Anil Kumar Pradhan. A New CRT-based Fully Homomorphic Encryption. *Cryptology ePrint Archive, Paper 2024/1105*, 2024.
23. Dan Rotaru et al. Practical multi-key fhe. In *ACM CCS*, 2021.
24. Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation Computer Systems*, 28:583–592, 2012.