Decentralized Multi-Client Functional Encryption for Set Intersection with Improved Efficiency

Kwangsu Lee*

Abstract

Functional encryption (FE) is a new paradigm of public key encryption that can control the exposed information of plaintexts by supporting computation on encrypted data. In this paper, we propose efficient multi-client FE (MCFE) schemes that compute the set intersection of ciphertexts generated by two clients. First, we propose an MCFE scheme that calculates the set intersection cardinality (MCFE-SIC) and prove its static security under dynamic assumptions. Next, we extend our MCFE-SIC scheme to an MCFE scheme for set intersection (MCFE-SI) and prove its static security under dynamic assumptions. The decryption algorithm of our MCFE-SI scheme is more efficient than the existing MCFE-SI scheme because it requires fewer pairing operations to calculate the intersection of two clients. Finally, we propose a decentralized MCFE scheme for set intersection (DMCFE-SI) that decentralizes the generation of function keys. Our MCFE schemes can be effectively applied to a privacy-preserving contact tracing system to prevent the spread of recent infectious diseases.

Keywords: Functional encryption, Multi-client setting, Private set intersection, Contact tracing, Bilinear maps.

*Sejong University, Seoul, Korea. Email: kwangsu@sejong.ac.kr.

Contents

1 Introduction

Functional encryption (FE) is a cryptographic technique that supports an controlled functional evaluation on encrypted data, and has an interesting feature that the result of the function evaluation is directly revealed in the decryption [8]. That is, in FE, a user creates a ciphertext for a plaintext x using a public key, and an entity who possesses the secret key for a function f issued by a trusted center can obtain $f(x)$ by decrypting the ciphertext. As an interesting extension of FE, multiple-input FE (MIFE) that handles multiple ciphertexts during decryption and multi-client FE (MCFE) that provides independent encryption keys for each clients were proposed [14]. FE schemes that support arbitrary functions can be constructed by using indistinguishable obfuscation, but indistinguishable obfuscation is still inefficient to implement. In order to construct an efficient FE scheme, research on FE that supports only special functions instead of general functions has been actively conducted recently [2–4].

Recently, FE schemes that supports the set intersection operation was proposed [26, 30]. An interesting application of the FE schemes for set intersection is privacy-preserving contact tracing [1], which allows a user to check the possibility of contact with a confirmed patient while preserving the location privacy of the user. The first FE scheme for set intersection was proposed by Kamp et al. [30], but their FE scheme has the disadvantage of requiring to independently perform the setup algorithm for all pairs of clients. Later, Lee and Seo proposed a multi-client FE for set intersection (MCFE-SI) scheme that supports the generation of function keys between multiple clients after running a single setup algorithm [26]. They designed their MCFE-SI scheme in bilinear groups by inventing the equal-then-derive technique. That is, a client of an index i who has a set $X = \{x_k\}$ of items creates a ciphertext element $H(x_k)^{\alpha_i}$ for each item by using algebraic PRF, and it additionally sets a temporal key $K = e(H(x_k), \hat{g})^{\beta_i}$ as a symmetric key to encrypt an item x_k. If both i and j clients encrypt the same item x, then the temporal key $K = e(H(x)^{\alpha_i} H(x)^{\alpha_j}, \hat{g}^{\beta_i/(\alpha_i+\alpha_j)})$ can be derived if a function key $\hat{g}^{\beta_i/(\alpha_i+\alpha_j)}$ is provided.

In this paper, we intend to improve the performance and functionality of the MCFE-SI scheme of Lee and Seo [26]. The first problem with the MCFE-SI scheme of Lee and Seo is that their decryption algorithm is relatively slow. In other words, the decryption algorithm of their MCFE-SI scheme requires a process of decrypting all combinations of ciphertext elements of the two clients i and j and checking whether the correct value is derived. Thus, this decryption algorithm requires approximately $O(\ell^2)$ pairing operations where ℓ is the number of items and it cause a serious problem in performance when the size of a set increases. The second problem is that their MCFE-SI scheme requires a trusted center to generate function keys. The existence of a trusted center can hinder the deployment of this system to the real environment since there are issues such that the activities of users can be monitored by a central authority. Therefore, in this paper, we ask whether it is possible to design an MCFE-SI scheme that supports efficient decryption and decentralized function key generation.

1.1 Our Contributions

In this paper, we devise efficient MCFE-SI schemes and give positive answers to the preceding questions. The detailed results of our contributions are summarized as follows.

MCFE for Set Intersection Cardinality. We first propose an MCFE for set intersection cardinality scheme (MCFE-SIC) that calculates the cardinality of the intersection of two client's sets. To support the set intersection cardinality, we use the ciphertext structure of the MCFE-SI scheme proposed by Lee and Seo [26], and modify their scheme to provide a new function key to check whether the ciphertext elements generated by different clients contain equal items. At this time, in order to test the equality of the ciphertext ele-

ments generated by different clients, we notice that the ciphertext structure of Lee and Seo uses an algebraic pseudo-random function (PRF) which is defined as $H(x)^{\alpha_i}$ where x is an item and α_i is the secret key of an i-index client and H is a hash function. If a function key is provided as $(\hat{g}^{\alpha_i r}, \hat{g}^{\alpha_j r})$ where r is a random exponent, it is possible to check whether the ciphertext elements of two clients i and j are encryption of the same item through the equation $e(H(x)^{\alpha_i}, \hat{g}^{\alpha_j r}) = e(H(x)^{\alpha_j}, \hat{g}^{\alpha_j r})$ by using a pairing operation. The decryption algorithm of this scheme additionally exposes equality pattern between ciphertext elements in addition to the set intersection cardinality. The ciphertext of our MCFE-SIC scheme consists of ℓ ciphertext elements, the function key consists of two group elements, and the decryption algorithm requires 2ℓ pairing operations and $O(\ell \log \ell)$ comparison operations for sorting where ℓ is the number of items of a set.

MCFE for Set Intersection. Next, we propose an MCFE for Set Intersection (MCFE-SI) scheme with improved decryption performance compared to the previous MCFE-SI scheme. The idea of improving the decryption performance is to efficiently find a matching pair of ciphertext elements that contain the same item from two client ciphertexts by using the function key of our MCFE-SIC scheme. In order to decrypt the ciphertext elements of the actual set item in the ciphertext, we use the same equal-then-derive method proposed by Lee and Seo [26]. That is, when two matching ciphertext elements of two clients are $H(x)^{\alpha_i}$ and $H(x)^{\alpha_j}$, we can derive a temporal key $K = e(H(x)^{\alpha_i} H^{\alpha_j}, \hat{g}^{\beta_i/(\alpha_i+\alpha_j)}) = e(H(x), \hat{g})^{\beta_i}$ for symmetric-key decryption if a function key $\hat{g}^{\beta_i/(\alpha_i+\alpha_j)}$ is provided. To analyze the security of our MCFE-SI scheme, we prove the security of our scheme by using newly introduced complexity assumptions in the static-IND security model in which function key queries, corrupted clients, and challenge messages as initially submitted. Compared to the MCFE-SI scheme of Lee and Seo that requires approximately $O(\ell^2)$ pairing operations in decryption, Our MCFE-SI scheme is more efficient since the decryption algorithm requires only 2ℓ pairing operations and $O(\ell \log \ell)$ comparison operations where ℓ is the number of items of a set.

Decentralized MCFE for Set Intersection. Finally, we propose a decentralized MCFE scheme for set intersection (DMCFE-SI) that removes the trusted center that generates function keys in our MCFE-SI scheme. The function key of our MCFE-SI scheme is composed of two key elements $\hat{g}^{\alpha_i r}$ and $\hat{g}^{\alpha_j r}$ for calculating the set intersection cardinality and one key element $\hat{g}^{\beta_i/(\alpha_i+\alpha_j)}$ for deriving a temporal key. The difficulty of decentralizing the generation of function keys is that two clients i and j should select the same random exponent r and the exponent inverse operation $(\alpha_i + \alpha_j)^{-1}$ which includes client secret keys should be decentralized. In order to select the same random exponent, each client exposes a public key and runs the Diffie-Hellman non-interactive key exchange scheme between two clients. Decentralizing the exponent inverse operation cannot be solved in a simple way. In order to solve this problem, each client creates an encoded secret key by encrypting a secret key with one-time pad, and an entity that combines the partial function keys to performs the exponent inversion operation by itself after combining the encoded secret keys of two clients. We can prove the security of our DMCFE-SI scheme since the additionally exposed encoded secret keys are information theoretically secure.

1.2 Related Work

Functional Encryption. Boneh, Sahai, and Waters introduced the concept of functional encryption (FE) as a new paradigm for public key encryption [8]. They showed that identity-based encryption [7], attribute-based encryption [17], and predicate encryption [9] are all special forms of FE. For the first time, Garg et al. designed the first FE scheme that supports arbitrary functions by using indistinguishable obfuscation [13]. After that, Goldwasser et al. extended FE that process only one ciphertext in decryption to introduce the concept of multiple-input functional encryption (MIFE) and multi-client functional encryption (MCFE) that support function operations on multiple ciphertexts, and constructed these schemes by using indistinguish-

able obfuscation [14]. In addition, there have been various attempts to design FE schemes that support arbitrary functions by using other cryptographic primitives instead of using inefficient indistinguishable obfuscation [15, 16]. In order to design an efficient FE scheme, an FE scheme for inner-products (FE-IP) was proposed [2]. Since then, the FE-IP has been expanded to support function hiding, full security, multiple inputs, multiple clients, and decentralized key generation [3–5, 10, 25]. As another efficient FE schemes, an FE scheme that supports the set intersection operation and an FE schemes that supports conjunctive equality and range query operations between multiple clients have been proposed [24, 26, 30].

Private Set Intersection. Private set intersection (PSI) is a cryptographic technique that allows two parties computes the intersection of their private sets without revealing any other information of their sets. Compared to functional encryption that supports the set intersection operation, a PSI protocol requires additional interactions between two parties when calculating the set intersection. One simple way to implement a PSI protocol is to use the Diffie-Hellman key exchange protocol, which is efficient in the terms of communication but it requires public key operations [20]. Freedman et al. proposed a PSI protocol by using oblivious polynomial evaluation that expresses sets as polynomials [12]. After that, oblivious PRF based PSI protocols, garbled circuit based PSI protocols, and oblivious transfer based PSI protocols have been proposed [18, 19, 23, 27]. In order to reduce the communication overhead of PSI protocols, delegated PSI protocols in which a cloud server performs most of the computation of clients were also proposed [21]. Recently, private set intersection cardinality (PSI-CA) protocols for contact tracing has been proposed [11, 29].

2 Preliminaries

In this section, we define functional encryption, symmetric-key encryption, and pseudo-random function. We also introduced complexity assumptions to prove the security of our functional encryption schemes.

2.1 Multi-Client Functional Encryption

Multi-client functional encryption (MCFE) is an extension of functional encryption (FE) that supports computation on encrypted data, and it requires a client secret key for encryption and handles multiple ciphertexts during decryption [14]. In MCFE, the client of an index i encrypts a plaintext x_i with a time label T using the client secret key SK_i to generate a ciphertext $CT_{i,T}$. Subsequently, an entity who has a function key DK_f for a function f decrypts ciphertexts $CT_{1,T}, \ldots, CT_{n,T}$ with the same time label T and obtains a decrypted result $f(x_1, \ldots, x_n)$. The IND security model of MCFE is defined by Goldwasser et al. [14]. A more detailed syntax of MCFE is given as follows.

Definition 2.1 (Multi-Client Functional Encryption)**.** A multi-client functional encryption (MCFE) scheme consists of four algorithms **Setup**, **GenKey**, **Encrypt**, and **Decrypt**, which are defined as follows:

Setup$(1^\lambda, n)$. The setup algorithm takes as input a security parameter λ. It outputs a master key MK, client secret keys $(SK_i)_{i=1}^n$, and public parameters PP.

GenKey(f, MK, PP). The key generation algorithm takes as input a function f, the master key MK, and the public parameters PP. It outputs a function key DK_f.

Encrypt(x, T, SK_i, PP). The encryption algorithm takes as input a message x and the public parameters PP. It outputs a ciphertext CT.

Decrypt$(CT = (CT_i), DK_f, PP)$. The decryption algorithm takes as input a ciphertext CT encrypting a message x, a secret key SK_f corresponding to a function f, and the public parameters PP. It outputs a value $f(x)$.

The correctness property of FE is defined as follows: For all $(MK, PP) \leftarrow$ **Setup**(1^λ), $SK_f \leftarrow$ **GenKey**(f, MK, PP) for any function $f \in \mathcal{F}$, and $CT \leftarrow$ **Encrypt**(x, PP) for any $x \in \mathcal{X}$, it is required that **Decrypt**$(CT, SK_f, PP) = f(x)$.

2.2 Symmetric Key Encryption

Symmetric key encryption (SKE) is an encryption method that uses the same key for encryption and decryption. The general security model of SKE is the IND security model that allows multiple challenge ciphertext queries. For this paper, we use a one-message IND security model that only allows only one challenge ciphertext query. The detailed syntax of SKE is given as follows.

Definition 2.2 (Symmetric Key Encryption)**.** A symmetric key encryption (SKE) scheme consists of three algorithms **GenKey**, **Encrypt**, and **Decrypt**, which are defined as follows:

GenKey(1^λ). The key generation algorithm takes as input a security parameter λ. It outputs a symmetric key K.

Encrypt(M, K). The encryption algorithm takes as input a message $M \in \mathcal{M}$ and the symmetric key K. It outputs a ciphertext C.

Decrypt(C, K). The decryption algorithm takes as input a ciphertext CT and the symmetric key K. It outputs a message M or a symbol $\perp$.

The correctness property of SKE is defined as follows: For all K generated by **GenKey** and any message $M \in \mathcal{M}$, it is required that **Decrypt**$($**Encrypt**$(M, K), K) = M$.

2.3 Pseudo-Random Function

A pseudo-random function (PRF) is a function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ where $\mathcal{K}$ is a key space, $\mathcal{X}$ is a domain, and $\mathcal{Y}$ is a range. Let $F(k, \cdot)$ be an oracle for a uniformly chosen $k \in \mathcal{K}$ and $f(\cdot)$ be an oracle for a uniformly chosen function $f : \mathcal{X} \to \mathcal{Y}$. We say that a PRF F is secure if for all efficient adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ defined as $\mathbf{Adv}^{PRF}_{\mathcal{A}}(\lambda) = \left| \Pr[\mathcal{A}^{F(k,\cdot)} = 1] - \Pr[\mathcal{A}^{f(\cdot)} = 1] \right|$ is negligible in the security parameter λ.

2.4 Bilinear Groups

A bilinear group generator $\mathcal{G}$ takes as input a security parameter λ and outputs a tuple $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ where p is a random prime and $\mathbb{G}, \hat{\mathbb{G}}$, and $\mathbb{G}_T$ be three cyclic groups of prime order p. Let g and $\hat{g}$ be generators of $\mathbb{G}$ and $\hat{\mathbb{G}}$, respectively. The bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ has the following properties:

1. Bilinearity: $\forall u \in \mathbb{G}, \forall \hat{v} \in \hat{\mathbb{G}}$ and $\forall a, b \in \mathbb{Z}_p$, $e(u^a, \hat{v}^b) = e(u, \hat{v})^{ab}$.
2. Non-degeneracy: $\exists g \in \mathbb{G}, \hat{g} \in \hat{\mathbb{G}}$ such that $e(g, \hat{g})$ has order p in $\mathbb{G}_T$.

We say that $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ are asymmetric bilinear groups with no efficiently computable isomorphisms if the group operations in $\mathbb{G}, \hat{\mathbb{G}}$, and $\mathbb{G}_T$ as well as the bilinear map e are all efficiently computable, but there are no efficiently computable isomorphisms between $\mathbb{G}$ and $\hat{\mathbb{G}}$.

2.5 Complexity Assumptions

We introduce complexity assumptions necessary to prove the security of our MCFE schemes. These complexity assumptions we introduce are dynamic assumptions that are defined depending on the key queries of an attacker. Note that these assumptions are slight modifications of the assumptions introduced by Lee and Seo [26]. We analyze that these complexity assumptions hold in the general group model in Section 6.

Let n be a positive integer, ρ be a target index such that $\rho \in [n]$, and $Q = \{(i,j)\}$ be a set of index tuple that $i, j \in [n]$ and $i < j$. From n, ρ, and Q, we define an index set $J = \{k : 1 \leq k \neq \rho \leq n$ such that $(k,\rho) \notin Q$ if $k < \rho$ and $(\rho,k) \notin Q$ if $k > \rho\}$. This set can be computed by using the function *ComputeJ* which is described as follows:

$\underline{ComputeJ(n,\rho,Q)}$ where $Q = \{(i,j)\}$
1. Initialize a set $J = \emptyset$.
2. For each $k \in \{1,\ldots,n\} \setminus \{\rho\}$:
If $k < \rho$ and $(k,\rho) \notin Q$, then add k to J.
If $k > \rho$ and $(\rho,k) \notin Q$, then add k to J.
3. Output the set J.

For example, if we let $n = 4$, $\rho = 2$, and $Q = \{(1,4),(2,3),(2,4)\}$, then we obtain $J = \{1\}$ since $(1,2) \notin Q$, $(2,3) \in Q$, and $(2,4) \in Q$.

Assumption 1. Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be a bilinear group randomly generated by $\mathcal{G}(1^\lambda)$. Let $g, \hat{g}$ be random generators of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. Let n, ρ, Q, J be defined above. The Assumption 1 for (n, ρ, Q, J) is that if the challenge tuple

$$D = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e),\ g,\ g^a,\ \{g^{b_i}\}_{i=1}^n,\ \{g^{ab_k}\}_{k\in J},\ \hat{g},\ \{(\hat{g}^{b_i c_{i,j}},\ \hat{g}^{b_j c_{i,j}})\}_{(i,j)\in Q}\big) \text{ and } Z$$

are given, no probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = g^{ab_\rho}$ from $Z = Z_1 = g^d$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{\text{A1-}(n,\rho,Q,J)}(\lambda) = \big|\Pr[\mathcal{A}(D,Z_0) = 0] - \Pr[\mathcal{A}(D,Z_1) = 0]\big|$ where the probability is taken over random choices of $a, b_1, \ldots, b_n, d \in \mathbb{Z}_p$.

Assumption 2. Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be a bilinear group randomly generated by $\mathcal{G}(1^\lambda)$. Let $g, \hat{g}$ be random generators of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. Let n, ρ, Q, J be defined above. The Assumption 2 for (n, ρ, Q, J) is that if the challenge tuple

$$\begin{aligned} D = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e),\ g,\ g^a,\ \{g^{b_i}\}_{i=1}^n,\ \{g^{ab_k}\}_{k\in J},\\ \hat{g},\ \{(\hat{g}^{b_i c_{i,j}},\ \hat{g}^{b_j c_{i,j}},\ \hat{g}^{1/(b_i+b_j)})\}_{(i,j)\in Q}\big) \text{ and } Z \end{aligned}$$

are given, no probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = g^{ab_\rho}$ from $Z = Z_1 = g^d$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{\text{A2-}(n,\rho,Q,J)}(\lambda) = \big|\Pr[\mathcal{A}(D,Z_0) = 0] - \Pr[\mathcal{A}(D,Z_1) = 0]\big|$ where the probability is taken over random choices of $a, b_1, \ldots, b_n$, $\{c_{i,j}\}, d \in \mathbb{Z}_p$.

Assumption 3. Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be a bilinear group randomly generated by $\mathcal{G}(1^\lambda)$. Let $g, \hat{g}$ be random generators of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. Let n, ρ, Q be defined above. The Assumption 3 for (n, ρ, Q) is that if the challenge tuple

$$\begin{aligned} D = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e),\ g,\ g^a,\ \{g^{b_i}\}_{i=1}^n,\ \{g^{ab_k}\}_{1\leq k\neq\rho\leq n},\\ \hat{g},\ \{(\hat{g}^{b_i c_{i,j}},\ \hat{g}^{b_j c_{i,j}},\ \hat{g}^{d_i/(b_i+b_j)})\}_{(i,j)\in Q},\ \{\hat{g}^{d_i}\}_{1\leq i\neq\rho\leq n},\ e(g,\hat{g})^{d_\rho}\big) \text{ and } Z \end{aligned}$$

are given, no probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = e(g,\hat{g})^{ad_\rho}$ from $Z = Z_1 = e(g,\hat{g})^{f}$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{A3\text{-}(n,\rho,Q)}(\lambda) = \left|\Pr[\mathcal{A}(D,Z_0)=0] - \Pr[\mathcal{A}(D,Z_1)=0]\right|$ where the probability is taken over random choices of $a, b_1, \ldots, b_n$, $\{c_{i,j}\}, d_1, \ldots, d_n, f \in \mathbb{Z}_p$.

3 MCFE for Set Intersection Cardinality

In this section, we define the syntax and security model of MCFE that calculates the set intersection cardinality. And then we propose an efficient MCFE-SIC scheme by using a bilinear map and analyze the security of our scheme.

3.1 Definition

We define the syntax of MCFE for set intersection cardinality (MCFE-SIC). MCFE-SIC is a special form of FE and supports a function key for calculating the set intersection cardinality in which a ciphertext is associated with a time label T and each client has it own secret key SK_i for encryption. In MCFE-SIC, a trusted center creates client secret keys and public parameters. After that, an individual client associates an item set X_i with a time label T and generate a ciphertext $CT_{i,T}$ by using its secret key SK_i. A third entity who wants to calculate the set intersection cardinality receives a function key DK for client indexes (i,j) from the trusted center. After that, the third entity decrypts the ciphertexts of the i-index client and the j-index client by using the function key, and obtains the value $|X_i \cap X_j|$. The detailed syntax of MCFE-SIC is described as follows.

Definition 3.1 (MCFE for Set Intersection Cardinality)**.** A multi-client functional encryption for set intersection cardinality (MCFE-SIC) scheme for $\mathcal{D}$ and $\mathcal{T}$ consists of four algorithms **Setup**, **GenKey**, **Encrypt**, and **Decrypt**, which are defined as follows:

Setup$(1^\lambda, n)$. The setup algorithm takes as input a security parameter λ and the number of clients n. It outputs a master key MK, client secret keys $(SK_1, \ldots, SK_n)$, and public parameters PP.

GenKey(f, MK, PP). The function key generation algorithm takes as input a function $f = (i,j)$, the master key MK, and public parameters PP. It outputs a function key DK_f.

Encrypt(X_i, T, SK_i, PP). The encryption algorithm takes as input a set $X_i = \{x_{i,1}, \ldots, x_{i,\ell_i}\}$ of items where $x_{i,k} \in \mathcal{D}$, a time period $T \in \mathcal{T}$, a client secret key SK_i, and the public parameters PP. It outputs a ciphertext $CT_{i,T}$.

Decrypt$(CT_{i,T}, CT_{j,T}, DK_f, PP)$. The decryption algorithm takes as input two ciphertexts $CT_{i,T}$ and $CT_{j,T}$ for the same time T, a function key DK_f for a function $f = (i,j)$, and the public parameters PP. It outputs $|X_i \cap X_j|$ where X_i and X_j are associated with $CT_{i,T}$ and $CT_{j,T}$ respectively.

The correctness of MCFE-SIC is defined as follows: For all $MK, (SK_i)_{i=1}^n, PP \leftarrow$ **Setup**$(1^\lambda, n)$, any $DK_f \leftarrow$ **GenKey**(f, MK, PP) of a function $f = (i,j)$, and all $CT_{i,T} \leftarrow$ **Encrypt**(X_i, T, SK_i, PP) and $CT_{j,T} \leftarrow$ **Encrypt** (X_j, T, SK_j, PP) for any X_i, X_j and the same time T, it is required that

- **Decrypt**$(CT_{i,T}, CT_{j,T}, DK_f, PP) = |X_i \cap X_j|$ except with negligible probability.

We define the IND security model of MCFE-SIC. The security model of MCFE was first defined by Goldwasser et al. [14]. For the security model of MCFE-SIC, we use the static IND security model of MCFE-SI defined by Lee and Seo with slight modification [26]. The static IND security model defined by Lee and Seo is a security model in which an attacker fixes function key queries and a list of corrupted clients in advance and submits the target challenge sets X_0^* and X_1^* in advance. At this time, we set a constraint that the cardinality of set intersection exposed in the challenge sets is the same even if many function keys are provided to an attacker. We consider a limited security model in which the cardinality of set intersections and the equality patterns of the challenge ciphertexts are exposed when an attacker decrypts the challenge ciphertexts using function keys.

We first define a function $CSIC((X_k)_{k\in I}, Q)$ for a tuple $(X_k)_{k\in I}$ of item sets X_k and a set $Q = \{(i,j)\}$ that computes the set intersection cardinality of X_i and X_j for each $(i,j) \in Q$ as follows:

$\underline{CSIC((X_k)_{k\in I}, Q)}$ where $Q = \{(i,j)\}$
1. Initialize a set $C = \emptyset$.
2. For each $(i,j) \in Q$:
   Calculate $c = |X_i \cap X_j|$ and add $((i,j),c)$ to C.
3. Output the set C.

Additionally, we define a function $CSIP((X_k)_{k\in I}, Q)$ for a tuple $(X_k)_{k\in I}$ of item sets X_k and a set $Q = \{(i,j)\}$ that computes the set intersection pattern of X_i and X_j for each $(i,j) \in Q$ as follows:

$\underline{CSIPA(i^*, (X_k)_{k\in I}, Q)}$
1. For each $x \in X_{i^*}$, initialize a set $S_x = \emptyset$.
2. For each $(i,j) \in Q$ such that $i = i^*$ or $j = i^*$:
   Calculate $Y = X_i \cap X_j$.
   For each $x \in Y$:
      If $i = i^*$, add j to S_x.
      If $j = i^*$, add i to S_x.
3. Output a pattern multiset $P_{i^*} = \{S_x\}_{x\in X_{i^*}}$.

$\underline{CSIP((X_k)_{k\in I}, Q)}$ where $Q = \{(i,j)\}$
1. For each $i \in I$:
   Calculate P_i by calling $CSIPA(i, (X_k)_{k\in I}, Q)$.
2. Output a tuple $(P_i)_{i\in I}$ of pattern multisets.

For example, if we let $n = 3, (X_1 = \{a,b,c\}, X_2 = \{b,c\}, X_3 = \{c,a\})$, and $Q = \{(1,2),(2,3)\}$, then we have $CSIC((X_k), Q) = \{((1,2),2),((2,3),1)\}$ and $CSIP((X_k), Q) = (P_1 = \{\emptyset, \{2\}, \{2\}\}, P_2 = \{\{1\}, \{1,3\}\}, P_3 = \{\{2\}, \emptyset\})$.

Definition 3.2 (Static-IND Security)**.** The static-IND security of MCFE-SIC with corruptions is defined in the following experiment $\mathbf{EXP}^{ST\text{-}IND}_{MCFE\text{-}SIC,\mathcal{A}}(1^\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ initially submits an index set $\bar{I} \subset [n]$ of corrupted clients. Let $I = \{1,\ldots,n\} \setminus \bar{I}$ be an index set of uncorrupted clients. $\mathcal{A}$ also submits two challenge tuples $(X^*_{0,k})_{k\in I}$ and $(X^*_{1,k})_{k\in I}$ of item sets $X^*_{b,k} = \{x_{b,k,j}\}$, a challenge time period T^*, and a set $Q = \{(i,j)\}$ of function key queries with the three restrictions such that (a) $i,j \in I$ for each $(i,j) \in Q$, (b) $CSIC((X^*_{0,k})_{k\in I}, Q) = CSIC((X^*_{1,k})_{k\in I}, Q)$, and (c) $CSIP((X^*_{0,k})_{k\in I}, Q) = CSIP((X^*_{1,k})_{k\in I}, Q)$.

2. **Setup**: $\mathcal{C}$ generates a master key MK, client secret keys $(SK_i)_{i=1}^n$, and public parameters PP by running **Setup**$(1^\lambda, n)$. It keeps MK and $(SK_i)_{i\in I}$ to itself and gives $(SK_i)_{i\in \bar{I}}$ and PP to $\mathcal{A}$.

3. **Challenge**: $\mathcal{C}$ flips a random bit $\mu \in \{0,1\}$ and obtains a ciphertext CT_{i,T^*} by running **Encrypt**$(X^*_{\mu,i}, T^*, SK_i, PP)$ for each $i \in I$. $\mathcal{C}$ gives the challenge ciphertexts $(CT_{i,T^*})_{i\in I}$ to $\mathcal{A}$

4. **Query**: $\mathcal{A}$ requests function keys and ciphertexts. $\mathcal{C}$ handles these queries as follows:

   - If this is a function key query for a function $f = (i,j) \in Q$, then $\mathcal{C}$ gives a function key DK_f to $\mathcal{A}$ by running **GenKey**(f, MK, PP).
   - If this is a ciphertext query for a client index $k \in I$, an item set X_k, and a time period $T \neq T^*$, then $\mathcal{C}$ gives a ciphertext $CT_{k,T}$ to $\mathcal{A}$ by running **Encrypt**(X_k, T, SK_k, PP).

5. **Guess**: $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$ of μ. $\mathcal{C}$ outputs 1 if $\mu = \mu'$ or 0 otherwise.

An MCFE-SIC scheme is static-IND secure with corruptions if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ defined as $\mathbf{Adv}^{ST\text{-}IND}_{MCFE\text{-}SIC,\mathcal{A}}(\lambda) = \left|\Pr[\mathbf{EXP}^{ST\text{-}IND}_{MCFE\text{-}SIC,\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right|$ is negligible in the security parameter λ.

3.2 Construction

The basic idea of designing an MCFE scheme that computes the set intersection cardinality of two clients is to provide a function key that can check whether ciphertext elements generated by two clients are related to the same item. For this, we can consider to provide a function key $(\hat{g}^{\alpha_i}, \hat{g}^{\alpha_j})$ because ciphertext elements are in the form of $H(T\|x)^{\alpha_i}$ and $H(T\|x)^{\alpha_j}$. In this case, by deriving the same $e(H(T\|x), \hat{g})^{\alpha_i,\alpha_j}$ through the pairing operation, it is possible to compare whether the ciphertext elements are associated to the same item x. However, providing a function key in this simple form has the risk of collusion attack, so we provide a function key $(\hat{g}^{\alpha_i r}, \hat{g}^{\alpha_j r})$ with additional randomization to prevent collusion attack. In this case, only the set intersection of two clients i and j can be compared due to the additionally included random exponent, and comparison with the ciphertexts of other clients is impossible. An MCFE-SIC scheme is described as follows:

Setup$(1^\lambda, n)$. Let n be the maximum number of clients. It first generates a bilinear group $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ of prime order p with random generators $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$. It chooses a hash function $H : \{0,1\}^* \to \mathbb{G}$. Next, it selects random exponents $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p$. It outputs a master key $MK = (\alpha_1, \ldots, \alpha_n)$, client secret keys $(SK_i = \alpha_i)_{i=1}^n$, and public parameters $PP = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, H, n\big)$.

GenKey(f, MK, PP). Let $f = (i,j)$ such that $i < j$ and $MK = (\alpha_1, \ldots, \alpha_n)$. It selects a random exponent $r \in \mathbb{Z}_p$ and outputs a function key $DK_f = \big(K_1 = \hat{g}^{\alpha_i r}, K_2 = \hat{g}^{\alpha_j r}\big)$.

Encrypt(X_i, T, SK_i, PP). Let $X_i = \{x_{i,1}, \ldots, x_{i,\ell_i}\}$ be a set of items where $|X_i| = \ell_i$ and $SK_i = \alpha_i$. For each $k \in [\ell_i]$, it computes $C_{i,k} = H(T\|x_{i,k})^{\alpha_i}$. It chooses a random permutation π and outputs a ciphertext $CT_{i,T} = \big(C_{i,\pi(k)}\big)_{k=1}^{\ell_i}$ by implicitly including i, T.

Decrypt$(CT_{i,T}, CT_{j,T}, DK_f, PP)$. Let $CT_{i,T} = (C_{i,k})_{k=1}^{\ell_i}$ and $CT_{j,T} = (C_{j,k})_{k=1}^{\ell_j}$ be ciphertexts such that $i < j$. Let $DK_f = (K_1, K_2)$ for a function $f = (i,j)$.

1. For each $k \in [\ell_i]$, it computes $E_{i,k} = e(C_{i,k}, K_2)$. For each $k \in [\ell_j]$, it computes $E_{j,k} = e(C_{j,k}, K_1)$.

2. It prepares two sets $E_i = \{E_{i,k}\}_{k=1}^{\ell_i}$ and $E_j = \{E_{j,k}\}_{k=1}^{\ell_j}$ and computes the intersection $S = E_i \cap E_j$ by comparing group elements.
3. It outputs the cardinality of S by counting the number of elements.

3.3 Correctness

We show the correctness of the MCFE-SIC scheme. For this, it is sufficient to show that the same group element is derived by combining a ciphertext element and a function key when the items of two clients are the same. We can derive the following equation when the item x of the client i and the item x' of the client j are the same.

$$e(C_{i,k}, K_2) = e(H(T\|x)^{\alpha_i}, \hat{g}^{\alpha_j r}) = e(H(T\|x), \hat{g})^{\alpha_i \alpha_j r} = e(H(T\|x')^{\alpha_j}, \hat{g}^{\alpha_i r}) = e(C_{j,k'}, K_1).$$

3.4 Security Analysis

We define a function $CIQ((X_k), Q)$ for a tuple (X_k) of item sets and an index set $Q = \{(i,j)\}$ that computes the collected intersection of X_i and X_j for each $(i,j) \in Q$ as follows:

$\underline{CIQ((X_k)_{k\in I}, Q)}$ where $Q = \{(i,j)\}$
1. For each $i \in I$, initialize a set $E_i = \emptyset$.
2. For each $(i,j) \in Q$:
   Calculate $Y = X_i \cap X_j$.
   For each $x \in Y$: Add x to E_i and E_j respectively.
3. Output a tuple $(E_i)_{i\in I}$ of common sets.

Theorem 3.1. *The above MCFE-SIC scheme is static-IND secure with no corruptions in the random oracle model if the Assumption 1 holds.*

Proof. Suppose there exists an adversary that breaks the static-IND security of the MCFE-SIC scheme with no corruptions. We can assume that $I = \{1,\ldots,n\}$ and $\bar{I} = \emptyset$. Let $(X^*_{0,1},\ldots,X^*_{0,n})$ and $(X^*_{1,1},\ldots,X^*_{1,n})$ be the challenge tuples of item sets where $X^*_{b,i} = \{x^*_{b,i,1},\ldots,x^*_{b,i,\ell_i}\}$ and $|X^*_{b,i}| = \ell_i$. Let $Q = \{(i,j)\}$ be the set of function key queries. We derive a tuple $(E^*_1,\ldots,E^*_n)$ by calling $CIQ((X^*_{\mu,k})_{k\in[n]}, Q)$ where μ is the challenge random bit of the security game. To argue that the adversary cannot win this game, we define a sequence of hybrid games $\mathbf{G}_0$, and $\mathbf{G}_1$. The game $\mathbf{G}_i$ is defined as follows:

Game $\mathbf{G}_0$. The first game $\mathbf{G}_0$ is the original security game defined in Definition 3.2.

Game $\mathbf{G}_1$. This game $\mathbf{G}_1$ is similar to the game $\mathbf{G}_0$ except that the challenge ciphertext components $\{C_{i,k}\}$ are generated as random for all $x^*_{\mu,i,k} \notin E^*_i$.

Let $S^{\mathbf{G}_i}_{\mathcal{A}}$ be the event that an adversary wins in a game $\mathbf{G}_i$. From the following lemmas 3.2 and 3.3, we obtain the following result

$$\mathbf{Adv}^{ST\text{-}IND}_{MCFE\text{-}SIC,\mathcal{A}}(\lambda) \leq \left|\Pr[S^{\mathbf{G}_0}_{\mathcal{A}}] - \Pr[S^{\mathbf{G}_1}_{\mathcal{A}}]\right| + \Pr[S^{\mathbf{G}_1}_{\mathcal{A}}] \leq n\ell\mathbf{Adv}^{A1\text{-}(n,\rho,Q,J)}_{\mathcal{B}}(\lambda)$$

where n is the number of clients, ℓ is the maximum size of the challenge item set. This completes our proof. □

Lemma 3.2. *If the Assumption1 for (n,ρ,Q,J) holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage.*

Proof. To prove this lemma, we additionally define hybrid games $\mathbf{H}_{1,0},\mathbf{H}_{1,1},\ldots,\mathbf{H}_{1,\ell_1},\mathbf{H}_{2,1},\ldots,\mathbf{H}_{i,k},\ldots,\mathbf{H}_{n,\ell_n}$ where $\mathbf{H}_{1,0}=\mathbf{G}_0$ and $\mathbf{H}_{n,\ell_n}=\mathbf{G}_1$. The game $\mathbf{H}_{\rho,\delta}$ is defined as follows:

Game $\mathbf{H}_{\rho,\delta}$. This game $\mathbf{H}_{\rho,\delta}$ is almost identical to the game $\mathbf{G}_1$ except the generation of the components $\{C_{i,k}\}$ in the challenge ciphertexts.

- **Case** $(i<\rho)$ or $(i=\rho\wedge k\leq\delta)$: If $x^*_{\mu,i,k}\in E^*_i$, then the component $C_{i,k}$ is generated as normal. Otherwise ($x^*_{\mu,i,k}\notin E^*_i$), the component $C_{i,k}$ is generated as random.
- **Case** $(i=\rho\wedge k>\delta)$ or $(i>\rho)$: The component $C_{i,k}$ is generated as normal.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H}_{\rho,\delta-1}$ and $\mathbf{H}_{\rho,\delta}$ with a non-negligible advantage. Without loss of generality, we assume that $x^*_{\mu,\rho,\delta}\notin E^*_\rho$ since $\mathbf{H}_{\rho,\delta-1}$ and $\mathbf{H}_{\rho,\delta}$ are equal if $x^*_{\mu,\rho,\delta}\in E^*_\rho$. A simulator $\mathcal{B}$ that solves the Assumption 1 for (n,ρ,Q,J) is described as follows:

Init: $\mathcal{A}$ submits challenge tuples $(X^*_{0,1},\ldots,X^*_{0,n})$ and $(X^*_{1,1},\ldots,X^*_{1,n})$, a challenge time period T^*, and a set $Q=\{(i,j)\}$ of function key queries. $\mathcal{B}$ proceeds as follows:

1. From n,ρ,Q, it derives an index set J by calling $ComputeJ(n,\rho,Q)$.
2. It receives a challenge tuple $D=(g,g^a,\{g^{b_i}\}_{i=1}^n,\{g^{ab_k}\}_{k\in J},\hat{g},\{(\hat{g}^{b_ic_{i,j}},\hat{g}^{b_jc_{i,j}})\}_{(i,j)\in Q})$ and Z of the Assumption 1 for (n,ρ,Q,J) where $Z=g^{ab_\rho}$ or $Z=R\in\mathbb{G}$.
3. It flips a random bit $\mu\in\{0,1\}$ internally and derives a tuple $(E^*_1,\ldots,E^*_n)$ by calling $CIQ((X^*_{\mu,k}),Q)$.

Setup: $\mathcal{B}$ sets $PP=((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e),g,\hat{g},H,n)$. It prepares a hash table H-list for the H hash function as empty one. For each $i\in[n]$ and $k\in[\ell_i]$, it updates the H-list as follows:

- Case $i\neq\rho$ or $k\neq\delta$: If $T^*\|x^*_{\mu,i,k}$ does not exist in the H-list, then it adds $(T^*\|x^*_{\mu,i,k},u'_{i,k},g^{u'_{i,k}})$ to the H-list by selecting a random exponent $u'_{i,k}\in\mathbb{Z}_p$.
- Case $i=\rho$ and $k=\delta$: It adds $(T^*\|x^*_{\mu,\rho,\delta},-,g^a)$ to the H-list.

Challenge: $\mathcal{B}$ creates challenge ciphertexts $CT_{1,T^*},\ldots,CT_{n,T^*}$ as follows:

1. For each $i\in[n]$ and $k\in[\ell_i]$, it generates ciphertext elements $C_{i,k}$ depending on the following cases:
   - Case $i<\rho$:
     - If $(x^*_{\mu,i,k}\in E^*_i)\wedge(x^*_{\mu,i,k}=x^*_{\mu,\rho,\delta})$, it retrieves $(T^*\|x^*_{\mu,i,k},-,g^a)$ from the H-list and sets $C_{i,k}=g^{ab_i}$. For this case, we show that g^{ab_i} is given in the assumption. If a function key for (i,ρ) was queried, we have $x^*_{\mu,\rho,\delta}\in E^*_\rho$ by the definition of CIQ. However, we assumed that $x^*_{\mu,\rho,\delta}\notin E^*_\rho$ for this game. Thus a function key for (i,ρ) was not queried and it means that $i\in J$ by the definition of J.
     - If $(x^*_{\mu,i,k}\in E^*_i)\wedge(x^*_{\mu,i,k}\neq x^*_{\mu,\rho,\delta})$, it retrieves $(T^*\|x^*_{\mu,i,k},u'_{i,k},g^{u'_{i,k}})$ from the H-list and creates $C_{i,k}=(g^{b_i})^{u'_{i,k}}$.
     - If $(x^*_{\mu,i,k}\notin E^*_i)$, it retrieves $(T^*\|x^*_{\mu,i,k},u'_{i,k},g^{u'_{i,k}})$ from the H-list and chooses a random $C_{i,k}\in\mathbb{G}$.

- Case $i = \rho$:
  - If $(k < \delta) \wedge (x^*_{\mu,\rho,k} \in E^*_\rho)$, it retrieves $(T^* \| x^*_{\mu,\rho,k}, u'_{\rho,k}, g^{u'_{\rho,k}})$ from the H-list and creates $C_{\rho,k} = (g^{b_\rho})^{u'_{\rho,k}}$ since $x^*_{\mu,\rho,k} \neq x^*_{\mu,\rho,\delta}$.
  - If $(k < \delta) \wedge (x^*_{\mu,\rho,k} \notin E^*_\rho)$, it retrieves $(T^* \| x^*_{\mu,\rho,k}, u'_{\rho,k}, g^{u'_{\rho,k}})$ from the H-list and chooses a random $C_{\rho,k} \in \mathbb{G}$.
  - If $(k = \delta)$, it sets $C_{\rho,\delta} = Z$ since we assumed that $x^*_{\mu,\rho,\delta} \notin E^*_\rho$.
  - If $(k > \delta)$, it retrieves $(T^* \| x^*_{\mu,\rho,k}, u'_{\rho,k}, g^{u'_{\rho,k}})$ from the H-list and creates $C_{\rho,k} = (g^{b_\rho})^{u'_{\rho,k}}$ since $x^*_{\mu,\rho,k} \neq x^*_{\mu,\rho,\delta}$.
- Case $i > \rho$:
  - If $(x^*_{\mu,i,k} = x^*_{\mu,\rho,\delta})$, it retrieves $(T^* \| x^*_{\mu,i,k}, -, g^a)$ from the H-list and sets $C_{i,k} = g^{ab_i}$. For this case, we show that g^{ab_i} is given in the assumption. If a function key for $f = (\rho, i)$ was queried, we have $x^*_{\mu,\rho,\delta} \in E^*_\rho$ by the definition of CIQ. However, we assumed that $x^*_{\mu,\rho,\delta} \notin E^*_\rho$ for this game. Thus a function key for $f = (\rho, i)$ was not queried and it means that $i \in J$ by the definition of J.
  - If $(x^*_{\mu,i,k} \neq x^*_{\mu,\rho,\delta})$, it retrieves $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ from the H-list and creates $C_{i,k} = (g^{b_i})^{u'_{i,k}}$.

2. For each client $i \in [n]$, it chooses a random permutation π_i and sets $CT_{i,T^*} = (C_{i,\pi_i(k)})_{k=1}^{\ell_i}$.

Query: $\mathcal{B}$ handles hash, function key, and ciphertext queries of $\mathcal{A}$ as follows:

- If this is a hash query for a time period T and an item x, then it proceeds as follows: If $T \| x$ exists in the H-list, then it retrieves $(T \| x, -, h)$ from the H-list and gives h to $\mathcal{A}$. Otherwise, it adds $(T \| x, u', g^{u'})$ to the H-list by selecting a random exponent $u' \in \mathbb{Z}_p$ and gives $g^{u'}$ to $\mathcal{A}$.
- If this is a function key query for a function $f = (i, j) \in Q$, then it generates a function key $DK_f = (\hat{g}^{b_i c_{i,j}}, \hat{g}^{b_j c_{i,j}})$ since these elements are given in the assumption.
- If this is a ciphertext query for a client index i, a set $X_i = \{x_{i,1}, \ldots, x_{i,\ell}\}$, and a time period $T \neq T^*$, then it generates a ciphertext as follows: For each $k \in [\ell_i]$, it retrieves $(T \| x_{i,k}, u'_k, g^{u'_k})$ from the H-list and sets $C_{i,k} = (g^{b_i})^{u'_k}$. It chooses a random permutation π and sets $CT_{i,T} = (C_{i,\pi(k)})_{k=1}^{\ell_i}$.

Guess: $\mathcal{A}$ outputs a guess μ'. If $\mu = \mu'$, it outputs 1. Otherwise, it outputs 0. □

Lemma 3.3. *No adversary can win the game $\boldsymbol{G}_1$ with a non-negligible advantage in the random oracle model.*

Proof. Let $\mathcal{A}$ be a statistical adversary. A simulator $\mathcal{B}$ is described as follows:

Init: $\mathcal{A}$ submits challenge tuples $(X^*_{0,1}, \ldots, X^*_{0,n})$ and $(X^*_{1,1}, \ldots, X^*_{1,n})$, a challenge time period T^*, and a set $Q = \{(i, j)\}$ of function key queries. $\mathcal{B}$ proceeds as follows:

1. It flips a random bit $\mu \in \{0, 1\}$ internally and derives a tuple $(E^*_1, \ldots, E^*_n)$ by calling $CIQ((X^*_{\mu,k})_{k \in [n]}, Q)$.

Setup: $\mathcal{B}$ first chooses random exponents $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p$. Next, it sets $(SK_i = \alpha_i)_{i=1}^n$ and $PP = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, H, n)$. It prepares a hash table H-list for the H hash function as empty one.

1. For each $i \in [n]$ and $k \in [\ell_i]$, it updates the H-list as follows: If $T^* \| x^*_{\mu,i,k}$ does not exist in the H-list, then it adds $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ to the H-list by selecting a random exponent $u'_{i,k} \in \mathbb{Z}_p$.

2. It sets $\overline{\mu} = 1 - \mu$. For each $i \in [n]$ and $k \in [\ell_i]$, it also updates the H-list as follows: If $T^* \| x^*_{\overline{\mu},i,k}$ does not exist in the H-list, then it adds $(T^* \| x^*_{\overline{\mu},i,k}, u'_{i,k}, g^{u'_{i,k}})$ to the H-list by selecting a random exponent $u'_{i,k} \in \mathbb{Z}_p$.

Challenge: $\mathcal{B}$ creates challenge ciphertexts $CT_{1,T^*}, \ldots, CT_{n,T^*}$ as follows:

1. For each $i \in [n]$ and $k \in [\ell_i]$, it proceeds as follows: If $x^*_{\mu,i,k} \in E^*_i$, it retrieves $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ from the H-list and sets $C_{i,k} = g^{u'_{i,k}\alpha_i}$. If $x^*_{\mu,i,k} \notin E^*_i$, it chooses a random element $C_{i,k} \in \mathbb{G}$.

2. For each $i \in [n]$, it chooses a random permutation π_i and sets $CT_{i,T^*} = (C_{i,\pi_i(k)})_{k=1}^{\ell_i}$.

Query: $\mathcal{B}$ handles hash, function key, and ciphertext queries of $\mathcal{A}$ as follows:

- If this is a hash query for a time period T and an item x, then it proceeds as follows: If $T \| x$ exists in the H-list, then it retrieves $(T \| x, u', g^{u'})$ from the H-list. Otherwise, it selects a random exponent $u' \in \mathbb{Z}_p$ and adds $(T \| x, u', g^{u'})$ to the H-list. It gives $g^{u'}$ to $\mathcal{A}$.
- If this is a function key query for $f = (i, j) \in Q$, then $\mathcal{B}$ generates DK_f by running **GenKey** since it knows SK_i and SK_j.
- If this is a ciphertext query for a client index i, a set $X_i = \{x_{i,1}, \ldots, x_{i,\ell}\}$, and a time period $T \neq T^*$, then $\mathcal{B}$ generates a ciphertext $CT_{i,T}$ by running **Encrypt** algorithm since it knows SK_i.

Guess: $\mathcal{A}$ outputs a guess μ'. If $\mu = \mu'$, it outputs 1. Otherwise, it outputs 0.

We first show that the simulation described above is correct. Since the simulator knows all the secret key SK_i of individual clients, it is possible to correctly generate function keys and all ciphertexts. When the simulator creates the challenge ciphertext, it creates the correct ciphertext element if $x^*_{\mu,i,k} \in E^*_i$ is established as in the definition of the game $\mathbf{G}_1$, and generates a random element if $x^*_{\mu,i,k} \notin E^*_i$ is established.

Now we show that the advantage of the statistical adversary is zero in the game $\mathbf{G}_1$. To do this, we show that it is possible to change the challenge ciphertext for the challenge bit μ to the challenge ciphertext for the complement bit $1 - \mu$ by modifying the mapping of the random oracle table. Such a change only modifies the mapping of the simulator's random oracle table without modifying the challenge ciphertexts. A detailed description of how to change the random oracle table is given as follows.

1. For each $i \in [n]$, it proceeds as follows:
   (a) It obtains $P_{\mu,i} = \{S_x\}$ by running $CSIPA(i, (X^*_{\mu,k}), Q)$. It also obtains $P_{\overline{\mu},i} = \{S_x\}$ by running $CSIPA(i, (X^*_{\overline{\mu},k}), Q)$.
   (b) It derives a list $XL^*_{\mu,i} = (x^*_{\mu,i,1}, \ldots, x^*_{\mu,i,\ell_i})$ from the challenge item set $X^*_{\mu,i} = \{x^*_{\mu,i,k}\}$ in which each challenge ciphertext element $C^*_{i,k}$ is associated with the item $x^*_{\mu,i,k}$.
   (c) It builds $XL^*_{\overline{\mu},i} = (x_{\overline{\mu},i,1}, \ldots, x^*_{\overline{\mu},i,\ell_i})$ from the challenge item set $X^*_{\overline{\mu},i} = \{x^*_{\overline{\mu},i,k}\}$ by changing the order of items with the condition that the pattern set $S_{x^*_{\mu,i,k}}$ of $x^*_{\mu,i,k}$ is equal to the pattern set $S_{x^*_{\overline{\mu},i,k}}$ of $x^*_{\overline{\mu},i,k}$.
2. It initializes a set $R = \emptyset$. For each $i \in [n]$ and $k \in [\ell_i]$, it takes $x^*_{\mu,i,k}$ and $x^*_{\overline{\mu},i,k}$ from $XL^*_{\mu,i}$ and $XL^*_{\overline{\mu},i}$ respectively, and modifies the H-list as follows:
   (a) If $(x^*_{\mu,i,k} \notin E^*_i) \vee (x^*_{\mu,i,k} = x^*_{\overline{\mu},i,k}) \vee (x^*_{\mu,i,k} \in R) \vee (x^*_{\overline{\mu},i,k} \in R)$, then it skips to the next iteration.

(b) It deletes $(T^*\|x^*_{\mu,i,k}, u'_1, g^{u'_1})$ and $(T^*\|x^*_{\overline{\mu},i,k}, u'_2, g^{u'_2})$ from the H-list, and then adds $(T^*\|x^*_{\overline{\mu},i,k}, u'_1, g^{u'_1})$ and $(T^*\|x^*_{\mu,i,k}, u'_2, g^{u'_2})$ to the H-list.

(c) It adds $x^*_{\mu,i,k}$ and $x^*_{\overline{\mu},i,k}$ to R.

If the random oracle table is changed in the same way as above, the actual elements of the challenge ciphertext is maintained as it is, so the equality pattern of the challenge ciphertext is not changed. Thus, if the challenge tuples of item sets with the same equality pattern are given, it is possible to change the challenge bit without changing the ciphertext through the above process. Therefore, the statistical adversary cannot distinguish the challenge ciphertext. □

Theorem 3.4. *The above MCFE-SIC scheme is static-IND secure with corruptions in the random oracle model if the MCFE-SIC scheme is static-IND secure with no corruptions.*

Proof. To prove this theorem, we use the fact that in the static-IND security model, the two indexes i and j of a function $f = (i, j)$ in a function key query requested by an attacker must be uncorrupted clients. In other words, the simulator of this proof generates the secret keys of corrupted clients $\bar{I}$, and it can handle all other challenge ciphertext, ciphertext, and function key queries requested by the attacker by using the queries of the MCFE-SIC scheme with no corruptions. We omit the detailed description of this simulator. □

3.5 Discussions

Efficiency Analysis. We analyze the efficiency of the MCFE-SIC scheme described above. First, the function key generation algorithm requires two exponentiation operations and a function key consists of two group elements. When the size of the item set is ℓ, the encryption algorithm requires ℓ hash operations and ℓ exponentiation operations, and a ciphertext consists of ℓ group elements. Finally, since the decryption algorithm requires a pairing operation for each individual ciphertext element, it requires 2ℓ pairing operations, and $2\ell \log \ell$ operations for sorting to perform the intersection of pairing elements.

Decentralized Function Key Generation. The function key generation algorithm of our MCFE-SIC scheme should be performed by a trusted center that knows the secret keys of all clients. To reduce trust in the trusted center, it is necessary to decentralize the function key generation so that individual clients are involved to generate function keys without the trusted center. One method is that when creating a function key for a function $f = (i, j)$, two clients with indexes i, j generate partial function keys independently of each other, and the requestor of the function key later combines these partial function keys to derive a complete function key. At this time, in order for the two clients to generate the same random exponent r, a non-interactive key exchange (NIKE) scheme can be used. For more detailed description of this method, refer to the DMCFE-SI scheme in Section 5.

Multi-Party Set Intersection Cardinality. The MCFE-SIC scheme can only process the set intersection cardinality between two clients. To process the set intersection cardinality between three clients, we may consider to provide a function key $(\hat{g}^{\alpha_j\alpha_k r}, \hat{g}^{\alpha_i\alpha_k r}, \hat{g}^{\alpha_i\alpha_j r})$ for the client indexes (i, j, k). However, this method has a problem of exposing information on the set intersection cardinality of clients $(i, j), (j, k)$, and (i, k) as well as the set intersection cardinality of clients (i, j, k). Another way is to select random exponents r_i, r_j, r_k to satisfy $r_i + r_j + r_k = 0$ and provide a function key $(\hat{g}^{r_i/\alpha_i}, \hat{g}^{r_j/\alpha_j}, \hat{g}^{r_k/\alpha_k})$. At this time, the decryption algorithm calculates $e(H(T\|x)^{\alpha_i}, \hat{g}^{r_i/\alpha_i}) = e(H(T\|x), \hat{g})^{r_i}$ for each ciphertext elements of each client. And then it multiply all possible combinations to check that $e(H(T\|x), \hat{g})^{r_i+r_j+r_k} = 1$ holds. This method can prevent the leakage of additional information, but it requires 3ℓ pairing operations and $O(\ell^3)$ multiplication operations since all possible combinations must be considered to calculate the set intersection cardinality.

4 MCFE for Set Intersection

In this section, we define the syntax and security model of MCFE for set intersection. Then, we propose an MCFE-SI scheme with efficient decryption using a bilinear map and analyze the security of our scheme.

4.1 Definition

We define the syntax of MCFE for set intersection (MCFE-SI). The definition of MCFE-SI was introduced by Lee and Seo [26], and it was modified to issue a function key for the set intersection instead of the function key for the set intersection cardinality in MCFE-SIC we introduced in the previous section. Thus, the decryption algorithm of MCFE-SI outputs the set intersection $X_i \cap X_j$ of two item sets X_i and X_j associated with two client ciphertexts $CT_{i,T}$ and $CT_{j,T}$. The detailed syntax of MCFE-SI is described as follows.

Definition 4.1 (MCFE for Set Intersection)**.** A multi-client functional encryption for set intersection (MCFE-SI) scheme for $\mathcal{D}$ and $\mathcal{T}$ consists of four algorithms **Setup**, **GenKey**, **Encrypt**, and **Decrypt**, which are defined as follows:

Setup$(1^\lambda, n)$. The setup algorithm takes as input a security parameter λ and the number of clients n. It outputs a master key MK, client secret keys $(SK_i)_{i=1}^n$, and public parameters PP.

GenKey(f, MK, PP). The key generation algorithm takes as input a function $f = (i, j)$, the master key MK, and public parameters PP. It outputs a function key DK_f.

Encrypt(X_i, T, SK_i, PP). The encryption algorithm takes as input a set $X_i = \{x_{i,1}, \ldots, x_{i,\ell_i}\}$ of items where $x_{i,k} \in \mathcal{D}$, a time period $T \in \mathcal{T}$, the client secret key SK_i, and the public parameters PP. It outputs a ciphertext $CT_{i,T}$.

Decrypt$(CT_{i,T}, CT_{j,T}, DK_f, PP)$. The decryption algorithm takes as input two ciphertexts $CT_{i,T}$ and $CT_{j,T}$ for the same time T, a function key DK_f for a function $f = (i, j)$, and the public parameters PP. It outputs a set $X_i \cap X_j$ where X_i and X_j are associated with $CT_{i,T}$ and $CT_{j,T}$ respectively.

The correctness of MCFE-SI is defined as follows: For all $MK, (SK_i)_{i=1}^n, PP \leftarrow$ **Setup**$(1^\lambda, n)$, any $DK_f \leftarrow$ **GenKey**(f, MK, PP) for a function $f = (i, j)$, and all $CT_{i,T} \leftarrow$ **Encrypt**(X_i, T, SK_i, PP) and $CT_{j,T} \leftarrow$ **Encrypt** (X_j, T, SK_j, PP) for any X_i, X_j and the same time T, it is required that

- **Decrypt**$(CT_{i,T}, CT_{j,T}, DK_f, PP) = X_i \cap X_j$ except with negligible probability.

We define the IND security model of MCFE-SI. The IND security model of MCFE was defined by Goldwasser et al. [14], and Lee and Seo modified this model to define a static IND security model of MCFE-SI [26]. We adopt the same static IND security model defined by Lee and Seo. In the static IND security model, an attacker first submits challenge sets X_0^*, X_1^*, a challenge time period T^*, and all function key queries, and corrupted client indexes with additional constraints. After that, the attacker receives the challenge ciphertext, and can request additional function key and ciphertext queries. Finally, if the attacker correctly guesses the challenge set of the challenge ciphertext, it wins the security game. A more detailed definition of the static IND security model is given as follows.

We first define a function $CSI((X_k)_{k \in I}, Q)$ for a tuple $(X_k)_{k \in I}$ of item sets X_k and a set $Q = \{(i, j)\}$ that computes the set intersection of X_i and X_j for each $(i, j) \in Q$ as follows:

$\underline{CSI((X_k)_{k\in I}, Q)}$ where $Q = \{(i,j)\}$
1. Initialize a set $S = \emptyset$.
2. For each $(i,j) \in Q$:
   Calculate $A = X_i \cap X_j$ and add $((i,j), A)$ to S.
3. Output the set S.

For example, if we let $n = 3, (X_1 = \{a,b,c\}, X_2 = \{b,c\}, X_3 = \{c,a\})$, and $Q = \{(1,2),(2,3)\}$, then we have $CSI((X_k), Q) = \{((1,2),\{b,c\}),((2,3),\{c\})\}$.

Definition 4.2 (Static-IND Security)**.** The static-IND security of MCFE-SI with corruptions is defined in the following experiment $\mathbf{EXP}^{ST\text{-}IND}_{MCFE\text{-}SI,\mathcal{A}}(1^\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ initially submits an index set $\bar{I} \subset [n]$ of corrupted clients. Let $I = \{1,\ldots,n\} \setminus \bar{I}$ be the index set of uncorrupted clients. $\mathcal{A}$ also submits two challenge tuples $(X^*_{0,k})_{k\in I}$ and $(X^*_{1,k})_{k\in I}$ of item sets, a challenge time period T^*, and a set $Q = \{(i,j)\}$ of function key queries with the two restrictions that (1) $i,j \in I$ for each $(i,j) \in Q$ and (2) $CSI((X^*_{0,k})_{k\in I}, Q) = CSI((X^*_{1,k})_{k\in I}, Q)$.

2. **Setup**: $\mathcal{C}$ generates a master key MK, secret keys $(SK_i)_{i=1}^n$, and public parameters PP by running $\mathbf{Setup}(1^\lambda, n)$. It keeps MK and $(SK_i)_{i\in I}$ to itself and gives $(SK_i)_{i\in\bar{I}}$ and PP to $\mathcal{A}$.

3. **Challenge**: $\mathcal{C}$ flips a random bit $\mu \in \{0,1\}$ and obtains a ciphertext CT_{i,T^*} by running $\mathbf{Encrypt}(X^*_{\mu,i}, T^*, SK_i, PP)$ for each $i \in I$. $\mathcal{C}$ gives the challenge ciphertexts $(CT_{i,T^*})_{i\in I}$ to $\mathcal{A}$

4. **Query**: $\mathcal{A}$ requests function keys and ciphertexts. $\mathcal{C}$ handles these queries as follows:

   - If this is a function key query for a function $f = (i,j) \in Q$, then $\mathcal{C}$ gives a function key DK_f to $\mathcal{A}$ by running $\mathbf{GenKey}(f, MK, PP)$.
   - If this is a ciphertext query for a client index $k \in I$, an item set X_k, and a time period $T \neq T^*$, then $\mathcal{C}$ gives a ciphertext $CT_{k,T}$ to $\mathcal{A}$ by running $\mathbf{Encrypt}(X_k, T, SK_k, PP)$.

5. **Guess**: $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$ of μ. $\mathcal{C}$ outputs 1 if $\mu = \mu'$ or 0 otherwise.

An MCFE-SI scheme is static-IND secure with corruptions if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ defined as $\mathbf{Adv}^{ST\text{-}IND}_{MCFE\text{-}SI,\mathcal{A}}(\lambda) = \left|\Pr[\mathbf{EXP}^{ST\text{-}IND}_{MCFE\text{-}SI,\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right|$ is negligible in the security parameter λ.

4.2 Construction

We combine our MCFE-SIC scheme of the previous section and the MCFE-SI scheme of Lee and Seo [26] in order to design an efficient MCFE-SI scheme with improved decryption. The MCFE-SI scheme of Lee and Seo uses an equal-then-derive technique in which if the items of two client ciphertext elements are equal, then a temporal key is derived by combining these ciphertexts and a function key. However, their MCFE-SI scheme has a disadvantage that the decryption algorithm requires approximately ℓ^2 pairing operations because the pairing operation must be performed for all possible combinations of two client ciphertext elements to calculate the set intersection. To improve the decryption performance, we first use our MCFE-SIC scheme to find matching pairs of ciphertext elements corresponding to the set intersection. And then we apply the equal-then-derive method to derive a temporal key to obtain an encrypted item. In this case, the total number of pairing operations can be reduced to 3ℓ.

Let **SKE** = (**GenKey**, **Encrypt**, **Decrypt**) be an SKE scheme. An MCFE-SI scheme is described as follows.

Setup$(1^\lambda, n)$. Let n be the maximum number of clients. It first generates a bilinear group $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ of prime order p with random generators $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$. It chooses two hash functions $H : \{0,1\}^* \to \mathbb{G}$ and $F : \mathbb{G}_T \to \{0,1\}^\lambda$. Next, it selects random exponents $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \in \mathbb{Z}_p$. It outputs a master key $MK = ((\alpha_i, \beta_i))_{i=1}^n$, secret keys $(SK_i = (\alpha_i, \beta_i))_{i=1}^n$ for clients, and public parameters $PP = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, H, F, n\big)$.

GenKey(f, MK, PP). Let $f = (i, j)$ such that $i < j$ and $MK = ((\alpha_i, \beta_i))_{i=1}^n$. It selects a random exponent $r \in \mathbb{Z}_p$ and outputs a function key $DK_f = \big(K_1 = \hat{g}^{\alpha_i r}, K_2 = \hat{g}^{\alpha_j r}, K_3 = \hat{g}^{\beta_i/(\alpha_i+\alpha_j)}\big)$.

Encrypt(X_i, T, SK_i, PP). Let $X_i = \{x_{i,1}, \ldots, x_{i,\ell_i}\}$ be a set of items where $|X_i| = \ell_i$ and $SK_i = (\alpha_i, \beta_i)$.

1. For each $k \in [\ell_i]$, it proceed as follows: It computes $C_{i,k} = H(T \| x_{i,k})^{\alpha_i}$ and derives a temporal key $TK_{i,k} = e(H(T \| x_{i,k}), \hat{g})^{\beta_i}$. It obtains $D_{i,k}$ by running **SKE.Encrypt**$(T \| x_{i,k}, F(TK_{i,k}))$.
2. It chooses a random permutation π and outputs a ciphertext $CT_{i,T} = \big((C_{i,\pi(k)}, D_{i,\pi(k)})\big)_{k=1}^{\ell_i}$ by implicitly including i, T.

Decrypt$(CT_{i,T}, CT_{j,T}, DK_f, PP)$. Let $CT_{i,T} = ((C_{i,k}, D_{i,k}))_{k=1}^{\ell_i}$ and $CT_{j,T} = ((C_{j,k}, D_{j,k}))_{k=1}^{\ell_j}$ be ciphertexts such that $i < j$ for the same T. Let $DK_f = (K_1, K_2, K_3)$ for a function $f = (i, j)$. It first initializes a set $Y = \emptyset$.

1. For each $k \in [\ell_i]$, it computes $E_{i,k} = e(C_{i,k}, K_2)$. For each $k \in [\ell_j]$, it computes $E_{j,k} = e(C_{j,k}, K_1)$.
2. It prepares two sets $E_i = \{E_{i,k}\}_{k=1}^{\ell_i}$ and $E_j = \{E_{j,k}\}_{k=1}^{\ell_j}$ and computes the intersection $S = E_i \cap E_j$ by comparing the group elements.
3. For each $E_k \in S$, it proceeds as follows:
   (a) It finds (C_{i,k_i}, D_{i,k_i}) from $CT_{i,T}$ and (C_{j,k_j}, D_{j,k_j}) from $CT_{j,T}$ such that C_{i,k_i} and C_{j,k_j} are used to derive E_k.
   (b) It computes $TK_k = e(C_{i,k_i} \cdot C_{j,k_j}, K_3)$ and obtains a string $T \| x$ by running **SKE.Decrypt** $(D_{i,k_i}, F(TK_k))$.
   (c) It adds an item x into Y.
4. It outputs the set Y.

4.3 Correctness

We show the correctness of the above MCFE-SI scheme. To this end, we need to show that when the ciphertext elements of two clients are the encryption of the same item, the matching ciphertext elements of the set intersection can be found, and when these matching ciphertext elements are decrypted with a function key, the set intersection item can be obtained. First, we already showed that if client ciphertext elements are the encryption of the same item, then matching ciphertext elements can be found by using a function key through the correctness of the MCFE-SIC scheme. Now, we can confirm that the correct item is decrypted from the matching ciphertext elements since a correct temporal key is derived by the following equation

$$e(C_{i,k} C_{j,k'}, K_3) = e(H(T \| x)^{\alpha_i} H(T \| x)^{\alpha_j}, \hat{g}^{\beta_i/(\alpha_i+\alpha_j)}) = e(H(T \| x), \hat{g})^{\beta_i}.$$

4.4 Security Analysis

Theorem 4.1. *The above MCFE-SI scheme is static-IND secure with no corruptions in the random oracle model if the Assumptions 2 and 3 hold.*

Proof. Suppose there exists an adversary that breaks the static-IND security of the MCFE-SI scheme with no corruptions. We can assume that $I = \{1,\ldots,n\}$ and $\bar{I} = \emptyset$. Let $(X^*_{0,1},\ldots,X^*_{0,n})$ and $(X^*_{1,1},\ldots,X^*_{1,n})$ be the challenge tuples where $X^*_{b,i} = \{x^*_{b,i,1},\ldots,x^*_{b,i,\ell_i}\}$ and $|X^*_{b,i}| = \ell_i$. Let $Q = \{(i,j)\}$ be the indexes set of function key queries. We can derive a tuple $(E^*_1,\ldots,E^*_n)$ by calling $CIQ((X^*_{\mu,k}),Q)$ where μ is the challenge random bit of the security game. To argue that the adversary cannot win this game, we define a sequence of hybrid games $\mathbf{G}_0,\mathbf{G}_1,\mathbf{G}_2$, and $\mathbf{G}_3$. The game $\mathbf{G}_i$ is defined as follows:

Game $\mathbf{G}_0$. The first game $\mathbf{G}_0$ is the original security game defined in Definition 4.2.

Game $\mathbf{G}_1$. This game $\mathbf{G}_1$ is similar to the game $\mathbf{G}_0$ except that the challenge ciphertext components $\{C_{i,k}\}$ are generated as random for all $x^*_{\mu,i,k} \notin E^*_i$.

Game $\mathbf{G}_2$. This game $\mathbf{G}_2$ is slightly changed from the game $\mathbf{G}_1$. That is, the challenge temporal keys $\{TK_{i,k}\}$ are generated as random for all $x^*_{\mu,i,k} \notin E^*_i$.

Game $\mathbf{G}_3$. In the final game $\mathbf{G}_3$, we change the generation of challenge ciphertext components $\{D_{i,k}\}$. That is, the challenge ciphertext components $\{D_{i,k}\}$ are the encryption of random values for all $x^*_{\mu,i,k} \notin E^*_i$. Note that the advantage of the adversary in this game is zero since challenge ciphertext components $\{C_{i,k}\}$ are random and $\{D_{i,k}\}$ are the encryption of random values for all $x^*_{\mu,i,k} \notin E^*_i$.

Let $S^{\mathbf{G}_i}_{\mathcal{A}}$ be the event that an adversary wins in a game $\mathbf{G}_i$. From the following lemmas 4.2, 4.3, and 4.4, we obtain the following result

$$\begin{aligned}\mathbf{Adv}^{ST\text{-}IND}_{MCFE\text{-}SI,\mathcal{A}}(\lambda) &\leq \left|\Pr[S^{\mathbf{G}_0}_{\mathcal{A}}] - \Pr[S^{\mathbf{G}_3}_{\mathcal{A}}]\right| + \Pr[S^{\mathbf{G}_3}_{\mathcal{A}}] \leq \sum_{i=1}^{3}\left|\Pr[S^{\mathbf{G}_{i-1}}_{\mathcal{A}}] - \Pr[S^{\mathbf{G}_i}_{\mathcal{A}}]\right| + \Pr[S^{\mathbf{G}_3}_{\mathcal{A}}] \\ &\leq n\ell\mathbf{Adv}^{A2\text{-}(n,\rho,Q,J)}_{\mathcal{B}}(\lambda) + n\ell\mathbf{Adv}^{A3\text{-}(n,\rho,Q)}_{\mathcal{B}}(\lambda) + n\ell\mathbf{Adv}^{SKE}_{\mathcal{B}}(\lambda)\end{aligned}$$

where n is the number of clients, ℓ is the maximum size of the challenge item set. This completes our proof. □

Lemma 4.2. *If the Assumption 2 for (n,ρ,Q,J) holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage.*

Proof. To prove this lemma, we additionally define hybrid games $\mathbf{H}_{1,0},\mathbf{H}_{1,1},\ldots,\mathbf{H}_{1,\ell_1},\mathbf{H}_{2,1},\ldots,\mathbf{H}_{i,k},\ldots,\mathbf{H}_{n,\ell_n}$ where $\mathbf{H}_{1,0} = \mathbf{G}_0$ and $\mathbf{H}_{n,\ell_n} = \mathbf{G}_1$. The game $\mathbf{H}_{\rho,\delta}$ is defined as follows:

Game $\mathbf{H}_{\rho,\delta}$. This game $\mathbf{H}_{\rho,\delta}$ is almost identical to the game $\mathbf{G}_0$ except the generation of the components $\{C_{i,k}\}$ in the challenge ciphertexts.

- **Case** $(i < \rho)$ or $(i = \rho \wedge k \leq \delta)$: If $x^*_{\mu,i,k} \in E^*_i$, then the component $C_{i,k}$ is generated as normal. Otherwise ($x^*_{\mu,i,k} \notin E^*_i$), the component $C_{i,k}$ is generated as random.
- **Case** $(i = \rho \wedge k > \delta)$ or $(i > \rho)$: The component $C_{i,k}$ is generated as normal.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H}_{\rho,\delta-1}$ and $\mathbf{H}_{\rho,\delta}$ with a non-negligible advantage. Without loss of generality, we assume that $x^*_{\mu,\rho,\delta} \notin E^*_\rho$ since $\mathbf{H}_{\rho,\delta-1}$ and $\mathbf{H}_{\rho,\delta}$ are equal if $x^*_{\mu,\rho,\delta} \in E^*_\rho$. A simulator $\mathcal{B}$ that solves the Assumption 2 for (n,ρ,Q,J) which will be defined later is described as follows:

Init: $\mathcal{A}$ submits challenge tuples $(X^*_{0,1},\ldots,X^*_{0,n})$ and $(X^*_{1,1},\ldots,X^*_{1,n})$, a challenge time period T^*, and a set $Q=\{(i,j)\}$ of function key queries. $\mathcal{B}$ proceeds as follows:

1. From n,ρ,Q, it derives an index set J by calling $ComputeJ(n,\rho,Q)$.
2. It receives a challenge tuple $D=(g,g^a,\{g^{b_i}\}_{i=1}^n,\{g^{ab_k}\}_{k\in J},\hat{g},\{(\hat{g}^{b_i c_{i,j}},\hat{g}^{b_j c_{i,j}},\hat{g}^{1/(b_i+b_j)})\}_{(i,j)\in Q})$ and Z of the Assumption 2 for (n,ρ,Q,J) where $Z=g^{ab\rho}$ or $Z=R\in\mathbb{G}$.
3. It flips a random bit $\mu\in\{0,1\}$ internally and derives a tuple $(E^*_1,\ldots,E^*_n)$ by calling $CIQ((X^*_{\mu,k}),Q)$.

Setup: $\mathcal{B}$ first chooses random exponents $\beta_1,\ldots,\beta_n\in\mathbb{Z}_p$. Next, it sets $PP=((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e),g,\hat{g},H,F,n)$. It prepares a hash table H-list for the H hash function as follows:

1. For each $i\in[n]$ and $k\in[\ell_i]$, it proceeds as follows: If $i\neq\rho$ or $k\neq\delta$, then it selects a random exponent $u'_{i,k}\in\mathbb{Z}_p$ and adds $(T^*\|x^*_{\mu,i,k},u'_{i,k},g^{u'_{i,k}})$ to the H-list. Otherwise ($i=\rho\wedge k=\delta$), it adds $(T^*\|x^*_{\mu,\rho,\delta},-,g^a)$ to the H-list.

Challenge: $\mathcal{B}$ creates challenge ciphertexts $CT_{1,T^*},\ldots,CT_{n,T^*}$ as follows:

1. For each $i\in[n]$ and $k\in[\ell_i]$, it generates ciphertext elements $C_{i,k}$ and $TK_{i,k}$ depending on the following cases:
   - Case $i<\rho$:
     - If $(x^*_{\mu,i,k}\in E^*_i)\wedge(x^*_{\mu,i,k}=x^*_{\mu,\rho,\delta})$, it retrieves $(T^*\|x^*_{\mu,i,k},-,g^a)$ from the H-list, and sets $C_{i,k}=g^{ab_i}$ and creates $TK_{i,k}=e(g^a,\hat{g})^{\beta_i}$. For this case, we show that g^{ab_i} is given in the assumption. If a function key for $f=(i,\rho)$ was queried, we have $x^*_{\mu,\rho,\delta}\in E^*_\rho$ by the definition of CIQ. However, we assumed that $x^*_{\mu,\rho,\delta}\notin E^*_\rho$ for this game. Thus a function key for $f=(i,\rho)$ was not queried and it means that $i\in J$ by the definition of J.
     - If $(x^*_{\mu,i,k}\in E^*_i)\wedge(x^*_{\mu,i,k}\neq x^*_{\mu,\rho,\delta})$, it retrieves $(T^*\|x^*_{\mu,i,k},u'_{i,k},g^{u'_{i,k}})$ from the H-list, and creates $C_{i,k}=(g^{b_i})^{u'_{i,k}}$ and $TK_{i,k}=e(g^{u'_{i,k}},\hat{g})^{\beta_i}$.
     - If $(x^*_{\mu,i,k}\notin E^*_i)$, it retrieves $(T^*\|x^*_{\mu,i,k},u'_{i,k},g^{u'_{i,k}})$ from the H-list, and chooses a random $C_{i,k}\in\mathbb{G}$ and creates $TK_{i,k}=e(g^{u'_{i,k}},\hat{g})^{\beta_i}$.
   - Case $i=\rho$:
     - If $(k<\delta)\wedge(x^*_{\mu,\rho,k}\in E^*_\rho)$, it retrieves $(T^*\|x^*_{\mu,\rho,k},u'_{\rho,k},g^{u'_{\rho,k}})$ from the H-list, and creates $C_{\rho,k}=(g^{b_\rho})^{u'_{\rho,k}}$ and $TK_{\rho,k}=e(g^{u'_{\rho,k}},\hat{g})^{\beta_\rho}$ since $x^*_{\mu,\rho,k}\neq x^*_{\mu,\rho,\delta}$.
     - If $(k<\delta)\wedge(x^*_{\mu,\rho,k}\notin E^*_\rho)$, it retrieves $(T^*\|x^*_{\mu,\rho,k},u'_{\rho,k},g^{u'_{\rho,k}})$ from the H-list, and chooses a random $C_{\rho,k}\in\mathbb{G}$ and creates $TK_{\rho,k}=e(g^{u'_{\rho,k}},\hat{g})^{\beta_\rho}$.
     - If $(k=\delta)$, it sets $C_{\rho,\delta}=Z$ and creates $TK_{\rho,\delta}=e(g^a,\hat{g})^{\beta_\rho}$ since we assumed that $x^*_{\mu,\rho,\delta}\notin E^*_\rho$.
     - If $(k>\delta)$, it retrieves $(T^*\|x^*_{\mu,\rho,k},u'_{\rho,k},g^{u'_{\rho,k}})$ from the H-list, and creates $C_{\rho,k}=(g^{b_\rho})^{u'_{\rho,k}}$ and $TK_{\rho,k}=e(g^{u'_{\rho,k}},\hat{g})^{\beta_\rho}$ since $x^*_{\mu,\rho,k}\neq x^*_{\mu,\rho,\delta}$.

- Case $i > \rho$:
  - If $(x^*_{\mu,i,k} = x^*_{\mu,\rho,\delta})$, it retrieves $(T^* \| x^*_{\mu,i,k}, -, g^a)$ from the H-list, and sets $C_{i,k} = g^{ab_i}$ and creates $TK_{i,k} = e(g^a, \hat{g})^{\beta_i}$. For this case, we show that g^{ab_i} is given in the assumption. If a function key for $f = (\rho, i)$ was queried, we have $x^*_{\mu,\rho,\delta} \in E^*_\rho$ by the definition of CIQ. However, we assumed that $x^*_{\mu,\rho,\delta} \notin E^*_\rho$ for this game. Thus a function key for $f = (\rho, i)$ was not queried and it means that $i \in J$ by the definition of J.
  - If $(x^*_{\mu,i,k} \neq x^*_{\mu,\rho,\delta})$, it retrieves $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ from the H-list, and creates $C_{i,k} = (g^{b_i})^{u'_{i,k}}$ and $TK_{i,k} = e(g^{u'_{i,k}}, \hat{g})^{\beta_i}$.

  Next, it generates a ciphertext element $D_{i,k}$ by running $\mathbf{SKE.Encrypt}(T^* \| x^*_{\mu,i,k}, TK_{i,k})$

2. For each $i \in [n]$, it chooses a random permutation π_i and sets $CT_{i,T^*} = ((C_{i,\pi_i(k)}, D_{i,\pi_i(k)}))_{k=1}^{\ell_i}$.

Query: $\mathcal{B}$ handles hash, function key, and ciphertext queries of $\mathcal{A}$ as follows:

- If this is a hash query for a time period T and an item x, then $\mathcal{B}$ proceeds as follows: If $T \| x$ exists in the H-list, then it retrieves $(T \| x, -, u)$ from H-list and gives u to $\mathcal{A}$. Otherwise, it selects a random exponent $u' \in \mathbb{Z}_p$ and adds $(T \| x, u', g^{u'})$ to the H-list, and then it gives the hash value $g^{u'}$ to $\mathcal{A}$.
- If this is a function key query for a function $f = (i, j) \in Q$, then $\mathcal{B}$ generates $DK_f = \big(\hat{g}^{b_i c_{i,j}}, \hat{g}^{b_j c_{i,j}}, (\hat{g}^{1/(b_i + b_j)})^{\beta_i}\big)$ since these elements are given in the assumption.
- If this is a ciphertext query for a client index i, a set $X_i = \{x_{i,1}, \ldots, x_{i,\ell}\}$, and a time period $T \neq T^*$, then $\mathcal{B}$ generates a ciphertext as follows:
  1. For each $k \in [\ell_i]$, it proceeds as follows: It retrieves $(T \| x_{i,k}, u'_k, g^{u'_k})$ from the H-list, and sets $C_{i,k} = (g^{b_i})^{u'_k}$ and $TK_{i,k} = e(g^{u'_k}, \hat{g})^{\beta_i}$. Next, it obtains $D_{i,k}$ by running $\mathbf{SKE.Encrypt}(T \| x_{i,k}, TK_{i,k})$.
  2. It chooses a random permutation π and sets $CT_{i,T} = ((C_{i,\pi(k)}, D_{i,\pi(k)}))_{k=1}^{\ell_i}$.

Guess: $\mathcal{A}$ outputs a guess μ'. If $\mu = \mu'$, it outputs 1. Otherwise, it outputs 0. □

Lemma 4.3. *If the Assumption 3 for (n, ρ, Q) holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ with a non-negligible advantage.*

Proof. To prove this lemma, we additionally define hybrid games $\mathbf{H}'_{1,0}, \mathbf{H}'_{1,1}, \ldots, \mathbf{H}'_{1,\ell_1}, \ldots, \mathbf{H}'_{i,k}, \ldots, \mathbf{H}'_{n,\ell_n}$ where $\mathbf{H}'_{1,0} = \mathbf{G}_1$ and $\mathbf{H}'_{n,\ell_n} = \mathbf{G}_2$. The game $\mathbf{H}'_{\rho,\delta}$ is defined as follows:

Game $\mathbf{H}'_{\rho,\delta}$. This game $\mathbf{H}'_{\rho,\delta}$ is almost identical to the game $\mathbf{G}_1$ except the generation of temporal keys $\{TK_{i,k}\}$ in the challenge ciphertexts.

- **Case** $(i < \rho)$ or $(i = \rho \wedge k \leq \delta)$: If $x^*_{\mu,i,k} \in E^*_i$, then the temporal key $TK_{i,k}$ is generated as normal. Otherwise $(x^*_{\mu,i,k} \notin E^*_i)$, the temporal key $TK_{i,k}$ is generated as random.
- **Case** $(i = \rho \wedge k > \delta)$ or $(i > \rho)$: The temporal key $TK_{i,k}$ is generated as normal.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H}'_{\rho,\delta-1}$ and $\mathbf{H}'_{\rho,\delta}$ with a non-negligible advantage. Without loss of generality, we assume that $x^*_{\mu,\rho,\delta} \notin E^*_\rho$ since $\mathbf{H}'_{\rho,\delta-1}$ and $\mathbf{H}'_{\rho,\delta}$ are equal if $x^*_{\mu,\rho,\delta} \in E^*_\rho$. A simulator $\mathcal{B}$ that solves the Assumption 3 for (n, ρ, Q) which will be defined later is described as follows:

Init: $\mathcal{A}$ submits challenge tuples $(X^*_{0,1}, \ldots, X^*_{0,n})$ and $(X^*_{1,1}, \ldots, X^*_{1,n})$, a challenge time period T^*, and a set $Q = \{(i, j)\}$ of function key queries. $\mathcal{B}$ proceeds as follows:

1. It receives a challenge tuple $D = (g, g^a, \{g^{b_i}\}_{i=1}^n, \{g^{ab_k}\}_{1\leq k\neq \rho\leq n}, \hat{g}, \{(\hat{g}^{b_i c_{i,j}}, \hat{g}^{b_j c_{i,j}}, \hat{g}^{d_i/(b_i+b_j)})\}_{(i,j)\in Q}, \{\hat{g}^{d_i}\}_{1\leq i\neq \rho\leq n}, e(g,\hat{g})^{d_\rho})$ and Z of the Assumption 3 for (n,ρ,Q) where $Z = e(g,\hat{g})^{ad_\rho}$ or $Z = R \in \mathbb{G}_T$.
2. It flips a random bit $\mu \in \{0,1\}$ internally and derives a tuple $(E_1^*, \ldots, E_n^*)$ by calling $CIQ((X_{\mu,k}^*), Q)$.

Setup: $\mathcal{B}$ sets $PP = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, H, F, n)$. It prepares a hash table H-list for the H hash function as follows:

1. For each $i \in [n]$ and $k \in [\ell_i]$, it proceeds as follows: If $i \neq \rho$ or $k \neq \delta$, then it selects a random exponent $u'_{i,k} \in \mathbb{Z}_p$ and adds $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ to the H-list. Otherwise ($i = \rho \wedge k = \delta$), it adds $(T^* \| x^*_{\mu,\rho,\delta}, -, g^a)$ to the H-list.

Challenge: $\mathcal{B}$ creates challenge ciphertexts $CT_{1,T^*}, \ldots, CT_{n,T^*}$ as follows:

1. For each $i \in [n]$ and $k \in [\ell_i]$, it generates ciphertext elements $C_{i,k}$ and $TK_{i,k}$ depending on the following cases:
   - Case $i < \rho$:
     - If $(x^*_{\mu,i,k} \in E_i^*) \wedge (x^*_{\mu,i,k} = x^*_{\mu,\rho,\delta})$, it retrieves $(T^* \| x^*_{\mu,i,k}, -, g^a)$ from the H-list, and sets $C_{i,k} = g^{ab_i}$ and $TK_{i,k} = e(g^a, \hat{g}^{d_i})$. In this case, g^{ab_i} is given in the assumption since $i \neq \rho$.
     - If $(x^*_{\mu,i,k} \in E_i^*) \wedge (x^*_{\mu,i,k} \neq x^*_{\mu,\rho,\delta})$, it retrieves $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ from the H-list, and sets $C_{i,k} = (g^{b_i})^{u'_{i,k}}$ and $TK_{i,k} = e(g^{u'_{i,k}}, \hat{g}^{d_i})$.
     - If $(x^*_{\mu,i,k} \notin E_i^*)$, it retrieves $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ from the H-list, and selects random $C_{i,k} \in \mathbb{G}$ and $TK_{i,k} \in \mathbb{G}_T$.
   - Case $i = \rho$:
     - If $(k < \delta) \wedge (x^*_{\mu,\rho,k} \in E_\rho^*)$, it retrieves $(T^* \| x^*_{\mu,\rho,k}, u'_{\rho,k}, g^{u'_{\rho,k}})$ from the H-list, and sets $C_{\rho,k} = (g^{b_\rho})^{u'_{\rho,k}}$ and $TK_{\rho,k} = (e(g,\hat{g})^{d_\rho})^{u'_{\rho,k}}$ since $x^*_{\mu,\rho,k} \neq x^*_{\mu,\rho,\delta}$.
     - If $(k < \delta) \wedge (x^*_{\mu,\rho,k} \notin E_\rho^*)$, it retrieves $(T^* \| x^*_{\mu,\rho,k}, u'_{\rho,k}, g^{u'_{\rho,k}})$ from the H-list, and selects random $C_{\rho,k} \in \mathbb{G}$ and random $TK_{\rho,k} \in \mathbb{G}_T$.
     - If $(k = \delta)$, it chooses a random $C_{\rho,\delta} \in \mathbb{G}$ and sets $TK_{\rho,\delta} = Z$ since we assumed that $x^*_{\mu,\rho,\delta} \notin E_\rho^*$.
     - If $(k > \delta) \wedge (x^*_{\mu,\rho,k} \in E_\rho^*)$, it retrieves $(T^* \| x^*_{\mu,\rho,k}, u'_{\rho,k}, g^{u'_{\rho,k}})$ from the H-list, and sets $C_{\rho,k} = (g^{b_\rho})^{u'_{\rho,k}}$ and $TK_{\rho,k} = (e(g,\hat{g})^{d_\rho})^{u'_{\rho,k}}$ since $x^*_{\mu,\rho,k} \neq x^*_{\mu,\rho,\delta}$.
     - If $(k > \delta) \wedge (x^*_{\mu,\rho,k} \notin E_\rho^*)$, it retrieves $(T^* \| x^*_{\mu,\rho,k}, u'_{\rho,k}, g^{u'_{\rho,k}})$ from the H-list, and selects a random $C_{\rho,k} \in \mathbb{G}$ and creates $TK_{\rho,k} = (e(g,\hat{g})^{d_\rho})^{u'_{\rho,k}}$.
   - Case $i > \rho$:
     - If $(x^*_{\mu,i,k} \in E_i^*) \wedge (x^*_{\mu,i,k} = x^*_{\mu,\rho,\delta})$, it retrieves $(T^* \| x^*_{\mu,i,k}, -, g^a)$ from the H-list, and sets $C_{i,k} = g^{ab_i}$ and $TK_{i,k} = e(g^a, \hat{g}^{d_i})$. In this case, g^{ab_i} is given in the assumption since $i \neq \rho$.
     - If $(x^*_{\mu,i,k} \in E_i^*) \wedge (x^*_{\mu,i,k} \neq x^*_{\mu,\rho,\delta})$, it retrieves $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ from the H-list, and sets $C_{i,k} = (g^{b_i})^{u'_{i,k}}$ and $TK_{i,k} = e(g^{u'_{i,k}}, \hat{g}^{d_i})$.
     - If $(x^*_{\mu,i,k} \notin E_i^*)$, it retrieves $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ from the H-list, and selects a random $C_{i,k} \in \mathbb{G}$ and creates $TK_{i,k} = e(g^{u'_{i,k}}, \hat{g}^{d_i})$.

Next, it generates a ciphertext element $D_{i,k}$ by running **SKE.Encrypt**$(T^*\|x^*_{\mu,i,k}, TK_{i,k})$

2. For each $i \in [n]$, it chooses a random permutation π_i and sets $CT_{i,T^*} = ((C_{i,\pi_i(k)}, D_{i,\pi_i(k)}))_{k=1}^{\ell_i}$.

Query: $\mathcal{B}$ handles hash, function key, and ciphertext queries of $\mathcal{A}$ as follows:

- If this is a hash query for a time period T and an item x, then $\mathcal{B}$ proceeds as follows: If $T\|x$ exists in the H-list, then it retrieves $(T\|x, -, u)$ from H-list and gives u to $\mathcal{A}$. Otherwise, it selects a random exponent $u' \in \mathbb{Z}_p$ and adds $(T\|x, u', g^{u'})$ to the H-list, and then it gives the hash value $g^{u'}$ to $\mathcal{A}$.
- If this is a function key query for a function $f = (i, j) \in Q$, then $\mathcal{B}$ generates $DK_f = \left(\hat{g}^{b_i c_{i,j}}, \hat{g}^{b_j c_{i,j}}, \hat{g}^{d_i/(b_i+b_j)}\right)$ since these elements are given in the assumption.
- If this is a ciphertext query for a client index i, a set $X_i = \{x_{i,1}, \ldots, x_{i,\ell}\}$, and a time period $T \neq T^*$, then $\mathcal{B}$ generates a ciphertext as follows:

  1. For each $k \in [\ell_i]$, it proceeds as follows: It retrieves $(T\|x_{i,k}, u'_k, g^{u'_k})$ from the H-list and sets $C_{i,k} = (g^{b_i})^{u'_k}$. Next, it sets $TK_{i,k} = (e(g,\hat{g})^{d_\rho})^{u'_k}$ if $i = \rho$, and it sets $TK_{i,k} = e(g^{u'_k}, \hat{g}^{d_i})$ if $i \neq \rho$. It obtains $D_{i,k}$ by running **SKE.Encrypt**$(T\|x_{i,k}, TK_{i,k})$.
  2. It chooses a random permutation π and creates $CT_{i,T} = ((C_{i,\pi(k)}, D_{i,\pi(k)}))_{k=1}^{\ell_i}$.

Guess: $\mathcal{A}$ outputs a guess μ'. If $\mu = \mu'$, it outputs 1. Otherwise, it outputs 0. □

Lemma 4.4. *If the SKE scheme is one-message secure, then no polynomial-time adversary can distinguish between $\mathbf{G}_2$ and $\mathbf{G}_3$ with a non-negligible advantage.*

Proof. To prove this lemma, we additionally define hybrid games $\mathbf{H}''_{1,0}, \mathbf{H}''_{1,1}, \ldots, \mathbf{H}''_{1,\ell_1}, \mathbf{H}''_{2,1}, \ldots, \mathbf{H}''_{i,k}, \ldots, \mathbf{H}''_{n,\ell_n}$ where $\mathbf{H}''_{1,0} = \mathbf{G}_2$ and $\mathbf{H}''_{n,\ell_n} = \mathbf{G}_3$. The game $\mathbf{H}''_{\rho,\delta}$ is defined as follows:

Game $\mathbf{H}''_{\rho,\delta}$. This game $\mathbf{H}''_{\rho,\delta}$ is almost identical to the game $\mathbf{G}_2$ except the generation of components $\{D_{i,k}\}$ in the challenge ciphertexts.

- **Case** $(i < \rho)$ or $(i = \rho \wedge k \leq \delta)$: If $x^*_{\mu,i,k} \in E^*_i$, then the component $D_{i,k}$ is generated as normal. Otherwise ($x^*_{\mu,i,k} \notin E^*_i$), the component $D_{i,k}$ is generated as the encryption of a random value.
- **Case** $(i = \rho \wedge k > \delta)$ or $(i > \rho)$: The component $D_{i,k}$ is generated as normal.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H}''_{\rho,\delta-1}$ and $\mathbf{H}''_{\rho,\delta}$ with a non-negligible advantage. Without loss of generality, we assume that $x^*_{\mu,\rho,\delta} \notin E^*_\rho$ since $\mathbf{H}''_{\rho,\delta-1}$ and $\mathbf{H}''_{\rho,\delta}$ are equal if $x^*_{\mu,\rho,\delta} \in E^*_\rho$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ submits challenge tuples $(X^*_{0,1}, \ldots, X^*_{0,n})$ and $(X^*_{1,1}, \ldots, X^*_{1,n})$ of item sets, a challenge time period T^*, and a set $Q = \{(i,j)\}$ of function key queries. $\mathcal{B}$ then flips a random bit $\mu \in \{0,1\}$ internally and derives a tuple $(E^*_1, \ldots, E^*_n)$ by calling $CIQ((X^*_{\mu,k}), Q)$.

Setup: $\mathcal{B}$ first chooses random exponents $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \in \mathbb{Z}_p$. Next, it sets $PP = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, H, F, n)$. It prepares a hash table H-list for the H hash function as follows:

1. For each $i \in [n]$ and $k \in [\ell_i]$, it selects a random exponent $u'_{i,k} \in \mathbb{Z}_p$ and adds $(T^*\|x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ to the H-list.

Challenge: $\mathcal{B}$ creates challenge ciphertexts $CT_{1,T^*}, \ldots, CT_{n,T^*}$ as follows:

1. For each $i \in [n]$ and $k \in [\ell_i]$, it generates ciphertext elements $C_{i,k}$ and $TK_{i,k}$ depending on the following cases:

   - Case $x^*_{\mu,i,k} \in E_i$: It retrieves $(T^* \| x^*_{\mu,i,k}, u'_{i,k}, g^{u'_{i,k}})$ from the H-list, and creates $C_{i,k} = g^{u'_{i,k}\alpha_i}$ and $TK_{i,k} = e(g^{u'_{i,k}}, \hat{g})^{\beta_i}$.
   - Case $x^*_{\mu,i,k} \notin E_i$: It selects random $C_{i,k} \in \mathbb{G}$ and random $TK_{i,k} \in \mathbb{G}_T$.

   Next, it also generates a ciphertext element $D_{i,k}$ depending on the following cases:

   - Case $(i < \rho)$ or $(i = \rho \wedge k < \delta)$: If $x^*_{\mu,i,k} \in E^*_i$, it creates $D_{i,k}$ by running **SKE.Encrypt**$(T^* \| x^*_{\mu,i,k}, TK_{i,k})$. Otherwise $(x^*_{\mu,i,k} \notin E^*_i)$, it selects a random $y \in \mathcal{D}$ and creates $D_{i,k}$ by running **SKE.Encrypt** $(T^* \| y, TK_{i,k})$.
   - Case $(i = \rho \wedge k = \delta)$: It selects a random $y \in \mathcal{D}$ and submits challenge message $x^*_{\mu,\rho,\delta}$ and y to the encryption oracle of SKE. Next, it receives a challenge ciphertext CT^*_{SKE} from SKE and sets $D_{\rho,\delta} = CT^*_{SKE}$. Recall that we assumed $x^*_{\mu,\rho,\delta} \notin E^*_\rho$.
   - Case $(i = \rho \wedge k > \delta)$ or $(i > \rho)$: It creates $D_{i,k}$ by running **SKE.Encrypt**$(T^* \| x^*_{\mu,i,k}, TK_{i,k})$.

2. For each $i \in [n]$, it chooses a random permutation π_i and sets $CT_{i,T^*} = ((C_{i,\pi_i(k)}, D_{i,\pi_i(k)}))_{k=1}^{\ell_i}$.

Query: $\mathcal{B}$ handles hash, function key, and ciphertext queries of $\mathcal{A}$ as follows:

- If this is a hash query for a time period T and an item x, then $\mathcal{B}$ proceeds as follows: If $T \| x$ exists in the H-list, then it retrieves $(T \| x, -, u)$ from H-list and gives u to $\mathcal{A}$. Otherwise, it selects a random exponent $u' \in \mathbb{Z}_p$ and adds $(T \| x, u', g^{u'})$ to the H-list, and then it gives the hash value $g^{u'}$ to $\mathcal{A}$.
- If this is a function key query for a function $f = (i, j)$, then $\mathcal{B}$ simply generates DK_f by using $\alpha_i, \alpha_j, \beta_i$.
- If this is a ciphertext query for a client index i, a set X_i, and a time period $T \neq T^*$, then $\mathcal{B}$ simply generates a ciphertext $CT_{i,T}$ by using α_i, β_i.

Guess: $\mathcal{A}$ outputs a guess μ'. If $\mu = \mu'$, it outputs 1. Otherwise, it outputs 0. □

Theorem 4.5. *The above MCFE-SI scheme is static-IND secure with corruptions in the random oracle model if the MCFE-SI scheme is static-IND secure with no corruptions.*

Proof. The proof of this theorem is similar to that of Theorem 3.4. In other words, the simulator of this proof generates the secret keys of corrupted clients by itself, and processes all other queries of an attacker using the queries of the MCFE-SI scheme with no corruptions. We omit the description of more detailed proofs. □

4.5 Discussions

Efficiency Analysis. We analyze the efficiency of the proposed MCFE-SI scheme. First, the function key is composed of two group elements for the set intersection cardinality and one group element for deriving a temporal key. The encryption algorithm requires ℓ map-to-point hash operations, ℓ exponentiation operations, and ℓ pairing operations since it requires operations in proportion to the size of a set. The decryption algorithm requires 2ℓ pairing operations, $\ell \log \ell$ comparison operations for sorting of group elements, and ℓ

pairing operations for deriving temporal keys to decrypt intersection items. Compared to the decryption algorithm of the MCFE-SI scheme of Lee and Seo [26] that requires approximately $O(\ell^2)$ pairing operations, the decryption algorithm of our scheme is more efficient since it only requires $O(\ell)$ pairing operations.

Outsourcing the Decryption of MCFE. If the ciphertexts generated by clients are stored on a cloud server, we can consider outsourcing part of the decryption operation to the cloud server. At this time, since the cloud server is not a trusted entity, we must be careful not to expose the set intersection information of the ciphertext to the cloud server. To this end, a client owning a function key $DK = (K_1, K_2, K_3)$ for indexes (i, j) selects a random exponent z and provides an outsourcing function key $oDK = (K_1, K_2, K_3^z)$ to the cloud server. Then, the cloud server finds ciphertext elements that satisfy the set intersection by using K_1 and K_2, derives outsourced temporal keys $oTK = e(C_{i,k}C_{j,k'}, K_3^z) = e(H(T\|x), \hat{g}^{\beta_i})^z$, and then it passes these keys back to the client. Then, the client raises all outsourced temporal keys to z^{-1} and decrypts corresponding ciphertexts with the temporal keys. At this time, the cloud server obtains information on the set intersection cardinality and information on the equality patterns, but does not obtain the set intersection items.

Multi-Party Set Intersection. In the previous section, we presented a method of extending the MCFE-SIC scheme to support the set intersection cardinality for multiple parties. Using this method, our MCFE-SI scheme can also be extended to support multi-party set intersection. That is, for calculating the set intersection cardinality, random exponents r_i, r_j, and r_k that satisfy $r_i + r_j + r_k = 0$ are selected and key elements $\hat{g}^{r_i/\alpha_i}, \hat{g}^{r_j/\alpha_j}, \hat{g}^{r_k/\alpha_k}$ are created. After that, an additional key element $\hat{g}^{\beta_i/(\alpha_i+\alpha_j+\alpha_k)}$ is provided to derive temporal keys. This method has the disadvantage that it requires $O(\ell^3)$ multiplication operations to find matching ciphertext elements, but it only requires $O(\ell)$ pairing operations.

5 Decentralized MCFE for Set Intersection

In this section, we define the syntax and security model of DMCFE-SI that generates function keys in a distributed way. And we propose an efficient DMCFE-SI scheme and analyze the security of the proposed scheme.

5.1 Definition

We define the syntax of decentralized MCFE-SI (DMCFE-SI). DMCFE-SI is a decentralized version of MCFE-SI in the previous section so that individual clients generate partial function keys instead of a trusted center generating a function key. In DMCFE-SI, individual clients sets their own private key SK_i and public key PK_i using the **ClientSetup** algorithm. And then individual clients generate partial function keys using the **GenPartKey** algorithm, and a third entity combines the partial function keys using the **CombPartKey** algorithm to derive a correct function key. That is, if the third entity wants to obtain a function key for client indexes (i, j), it receives a partial function key pDK_i from the i-index client and a partial function key pDK_j from the j-index client. And then, it combines the two partial function keys to derive the correct function key DK to decrypt a ciphertext. At this point, the encryption and decryption algorithms of DMCFE-SI are the same as those of MCFE-SI. The detailed syntax of DMCFE-SI is described as follows.

Definition 5.1 (Decentralized MCFE for Set Intersection)**.** A decentralized multi-client functional encryption for set intersection (DMCFE-SI) scheme for $\mathcal{D}$ and $\mathcal{T}$ consists of six algorithms **Setup**, **ClientSetup**, **GenPartKey**, **CombPartKey**, **Encrypt**, and **Decrypt**, which are defined as follows:

Setup$(1^\lambda, n)$. The global setup algorithm takes as input a security parameter λ and the number of clients n. It outputs public parameters PP.

ClientSetup(i, PP). The client setup algorithm takes as input an index i of a client and public parameters PP. It outputs a secret key SK_i and a public key PK_i.

GenPartKey(f, SK_i, PK, PP). The partial key generation algorithm takes as input a function f, a secret key SK_i, and a tuple PK of public keys, and public parameters PP. It outputs a partial function key $pDK_{i,f}$.

CombPartKey$(pDK_{i,f}, pDK_{j,f}, PP)$. The partial key combining algorithm takes as input two partial decryption keys $pDK_{i,f}$ and $pDK_{j,f}$ for a function $f = (i, j)$ and public parameters PP. It outputs a function key DK_f.

Encrypt(X_i, T, SK_i, PP). The encryption algorithm takes as input a set $X_i = \{x_{i,1}, \ldots, x_{i,\ell_i}\}$ of items where $x_{i,j} \in \mathcal{D}$, a time period $T \in \mathcal{T}$, a secret key SK_i, and the public parameters PP. It outputs a ciphertext $CT_{i,T}$.

Decrypt$(CT_{i,T}, CT_{j,T}, DK_f, PP)$. The decryption algorithm takes as input two ciphertexts $CT_{i,T}$ and $CT_{j,T}$ for the same time T, a function key DK_f, and the public parameters PP. It outputs a set $X_i \cap X_j$ where X_i and X_j are associated with $CT_{i,T}$ and $CT_{j,T}$ respectively.

The correctness of DMCFE-SI is defined as follows: For any $PP \leftarrow$ **Setup**$(1^\lambda, n)$, all $SK_i, PK_i \leftarrow$ **ClientSetup** (i, PP), and all $CT_{i,T} \leftarrow$ **Encrypt**(X_i, T, SK_i, PP) and $CT_{j,T} \leftarrow$ **Encrypt**(X_j, T, SK_j, PP) for any X_i, X_j and the same time T, it is required that

- **CombPartKey**(**GenPartKey**(f, SK_i, PK, PP), **GenPartKey**$(f, SK_j, PK, PP), PP) = DK_f$.
- **Decrypt**$(CT_{i,T}, CT_{j,T}, DK_f, PP) = X_i \cap X_j$ except with negligible probability.

We define the security model of DMCFE-SI. We define the static IND security model of DMCFE-SI by modifying the static IND security model of MCFE-SI defined in the previous section. This security model of DMCFE-SI is the same as that of MCFE-SI in Section 4.1, except that it allows partial function key queries instead of function key queries. In this security model of DMCFE-SI, partial function key queries requested by an attacker have two limitations. If a partial function key for a function $f = (i, j)$ requested by the attacker belongs to the predefined function key query set, then the attacker can request both a partial function key for a client i and a partial function key for a client j. However, if a partial function key for $f = (i, j)$ does not belong to the predefined function key query set, then the attacker can request only one partial function key for a client i or j. Thus, the attacker of DMCFE-SI allows not only predefined function key queries, but also additional partial function key queries. The more detailed security model of DMCFE-SI is defined as follows.

Definition 5.2 (Static-IND Security)**.** The static-IND security of DMCFE-SI with corruptions is defined in the following experiment $\mathbf{EXP}^{\textit{ST-IND}}_{\textit{DMCFE-SI},\mathcal{A}}(1^\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ initially submits an index set $\bar{I} \subset [n]$ of corrupted clients. Let $I = \{1, \ldots, n\} \setminus \bar{I}$ be the index set of uncorrupted clients. $\mathcal{A}$ also submits two challenge tuples $(X^*_{0,k})_{k \in I}$ and $(X^*_{1,k})_{k \in I}$ of item sets, a challenge time period T^*, and a set $Q = \{(i, j)\}$ of function key queries with the two restrictions that (1) $i, j \in I$ for each $(i, j) \in Q$ and (2) $CSI((X^*_{0,k})_{k \in I}, Q) = CSI((X^*_{1,k})_{k \in I}, Q)$.

2. **Setup**: $\mathcal{C}$ generates public parameters PP by running **Setup**$(1^\lambda, n)$. It also generates secret keys and public keys (SK_i, PK_i) of clients by running **ClientSetup**(i, PP) for each $i \in [n]$. It keeps $(SK_i)_{i \in I}$ to itself and gives $(SK_i)_{i \in \bar{I}}$, $PK = (PK_i)_{i=1}^n$, and PP to $\mathcal{A}$.

3. **Challenge**: $\mathcal{C}$ flips a random bit $\mu \in \{0,1\}$ and obtains a ciphertext CT_{i,T^*} by running **Encrypt**($X^*_{\mu,i}$, T^*, SK_i, PP) for each $i \in I$. $\mathcal{C}$ gives the challenge ciphertexts $(CT_{i,T^*})_{i\in I}$ to $\mathcal{A}$

4. **Query**: $\mathcal{A}$ requests function keys and ciphertexts. $\mathcal{C}$ handles these queries as follows:

   - If this is a partial function key query for a tuple $f = (i, j)$ and a client index k such that $k = i$ or $k = j$, then $\mathcal{C}$ gives a partial function key $pDK_{k,f}$ to $\mathcal{A}$ by running **GenPartKey**(f, SK_k, PK, PP) with the restrictions that (1) if $f \in Q$, then two partial function keys of i and j can be queried and (2) if $f \notin Q$, then only one partial function key of i or j can be queried.
   - If this is a ciphertext query for a client index $k \in I$, an item set X_k, and a time period $T \neq T^*$, then $\mathcal{C}$ gives a ciphertext $CT_{k,T}$ to $\mathcal{A}$ by running **Encrypt**(X_k, T, SK_k, PP).

5. **Guess**: $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$ of μ. $\mathcal{C}$ outputs 1 if $\mu = \mu'$ or 0 otherwise.

A DMCFE-SI scheme is static-IND secure with corruptions if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ defined as $\mathbf{Adv}^{ST\text{-}IND}_{DMCFE\text{-}SI,\mathcal{A}}(\lambda) = \left|\Pr[\mathbf{EXP}^{ST\text{-}IND}_{DMCFE\text{-}SI,\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right|$ is negligible in the security parameter λ.

5.2 Construction

The function key of the MCFE-SI scheme proposed in the previous section consist of K_1 and K_2 for set intersection cardinality and K_3 for deriving a temporal key for set intersection. We first devise a method to decentralize the generation of K_1 and K_2. In order for individual clients to generate these two group elements in an independent way, it is necessary to generate a common random exponent r. To this end, we derive the same shared key K by using a non-interactive key exchange NIKE scheme and we use PRF to derive the exponent r from the shared key K. That is, if an individual client additionally selects a private key γ_i and exposes a public key $h_i = g^{\gamma_i}$, then it can derive a shared key $K = g^{\gamma_i\gamma_j}$ by using a NIKE scheme. Thus, individual clients can generate partial function keys of $\hat{g}^{\alpha_i r}$ and $\hat{g}^{\alpha_j r}$ where $r = PRF(K, 1)$.

Now we devise a method to decentralize the generation of K_3 for derivation of a temporal key. However, it cannot be decentralized by a simple method since it requires the inverse operation of an exponent. In order to decentralize the calculation of the inverse operation while hiding the secret keys of two clients, we introduce a method in which the secret key is encrypted with a one-time pad scheme and a client requesting the partial function key combines the encrypted keys to calculate the inverse operation. That is, individual clients first derive the same shared key $K_{i,j}$ using the NIKE scheme, and derives the same random exponents s and t. Then, each client encrypts its secret key as $E_i = s\alpha_i + t$ and $E_j = s\alpha_j - t$, respectively. At this time, if the i index client additionally provides $\hat{g}^{\beta_i s}$, the client that received E_i and E_j can compute a key $(\hat{g}^{\beta_i s})^{1/(E_i+E_j)}$. Note that, since E_i and E_j have a one-to-one correspondence with random exponents s and t, the information of the secret keys is not exposed.

Let **SKE** = (**GenKey**, **Encrypt**, **Decrypt**) be an SKE scheme. A DMCFE-SI scheme is described as follows.

Setup($1^\lambda, n$). Let n be the maximum number of clients. It first generates a bilinear group $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ of prime order p with random generators $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$. It chooses two hash functions $H : \{0,1\}^* \to \mathbb{G}$ and $F : \mathbb{G}_T \to \{0,1\}^\lambda$. It outputs public parameters $PP = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, H, F, n\big)$.

ClientSetup(i, PP). Let i be the index of a client. It selects random exponents $\alpha_i, \beta_i, \gamma_i \in \mathbb{Z}_p$, and then it outputs a secret key $SK_i = (\alpha_i, \beta_i, \gamma_i)$ and a public key $PK_i = \big(h_i = g^{\gamma_i}\big)$.

GenPartKey(f, SK_k, PK, PP). Let $f = (i, j)$ such that $i < j$. Let $SK_k = (\alpha_k, \beta_k, \gamma_k)$ such that $k = i$ or $k = j$ and $PK = (PK_1, \ldots, PK_n)$.

1. If $k = i$, it retrieves $PK_j = h_j$ from PK and computes a shared key $K_{i,j} = h_j^{\gamma_i}$. Otherwise ($k = j$), it retrieves $PK_i = h_i$ from PK and computes a shared key $K_{i,j} = h_i^{\gamma_j}$. Next, it derives random exponents $r, s, t \in \mathbb{Z}_p$ by running $PRF(K_{i,j}, 1)$, $PRF(K_{i,j}, 2)$, $PRF(K_{i,j}, 3)$ respectively.
2. If $k = i$, it sets $A_2 = \hat{g}^{\beta_i \cdot s}$ and $E = s \cdot \alpha_i + t \mod p$. Otherwise, it sets $A_2 = 1_{\hat{\mathbb{G}}}$ and $E = s \cdot \alpha_j - t \mod p$. It outputs a partial function key $pDK_{k,f} = \left(A_1 = \hat{g}^{\alpha_k \cdot r}, A_2, E\right)$.

CombPartKey$(pDK_{i,f}, pDK_{j,f}, PP)$. Let $f = (i, j)$ such that $i < j$. Let $pDK_{i,f} = (A_1, A_2, E)$ and $pDK_{j,f} = (A_1', A_2', E')$. It selects a random exponent $r \in \mathbb{Z}_p$ and outputs a function key $DK_f = \left(K_1 = (A_1)^r, K_2 = (A_1')^r, K_3 = A_2^{1/(E+E')}\right)$.

Encrypt(X_i, T, SK_i, PP). Let $X_i = \{x_{i,1}, \ldots, x_{i,\ell_i}\}$ be a set of items where $|X_i| = \ell_i$ and $SK_i = (\alpha_i, \beta_i, \gamma_i)$.

1. For each $k \in [\ell_i]$, it proceed as follows: It computes $C_{i,k} = H(T \| x_{i,k})^{\alpha_i}$ and derives a temporal key $TK_{i,k} = e(H(T \| x_{i,k}), \hat{g})^{\beta_i}$. It obtains $D_{i,k}$ by running **SKE.Encrypt**$(T \| x_{i,k}, F(TK_{i,k}))$.
2. It chooses a random permutation π and outputs a ciphertext $CT_{i,T} = \left((C_{i,\pi(k)}, D_{i,\pi(k)})\right)_{k=1}^{\ell_i}$ by implicitly including i, T.

Decrypt$(CT_{i,T}, CT_{j,T}, DK_f, PP)$. Let $CT_{i,T} = ((C_{i,k}, D_{i,k}))_{k=1}^{\ell_i}$ and $CT_{j,T} = ((C_{j,k}, D_{j,k}))_{k=1}^{\ell_j}$ be ciphertexts such that $i < j$ for the same T. Let $DK_f = (K_1, K_2, K_3)$ where $f = (i, j)$. It first initializes a set $Y = \emptyset$.

1. For each $k \in [\ell_i]$, it computes $E_{i,k} = e(C_{i,k}, K_2)$. For each $k \in [\ell_j]$, it computes $E_{j,k} = e(C_{j,k}, K_1)$.
2. It prepares two sets $E_i = \{E_{i,k}\}_{k=1}^{\ell_i}$ and $E_j = \{E_{j,k}\}_{k=1}^{\ell_j}$ and computes the intersection $S = E_i \cap E_j$ by comparing the group elements.
3. For each $E_k \in S$, it proceeds as follows:
   (a) It finds (C_{i,k_i}, D_{i,k_i}) from $CT_{i,T}$ and (C_{j,k_j}, D_{j,k_j}) from $CT_{j,T}$ such that C_{i,k_i} and C_{j,k_j} are used to derive E_k.
   (b) It computes $TK_k = e(C_{i,k_i} \cdot C_{j,k_j}, K_3)$ and obtains a string $T \| x$ by running **SKE.Decrypt** $(D_{i,k_i}, F(TK_k))$.
   (c) It adds an item x into Y.
4. It outputs the set Y.

5.3 Correctness

We show the correctness of the DMCFE-SI scheme. First, two clients i and j can obtain the same shared key $K_{i,j}$ from the correctness of the Diffie-Hellman non-interactive key exchange scheme. And two clients i and j can derive the same random exponents r, s, and t since PRF is a deterministic function. Now, when a combing client combines the partial function key elements generated by using the same random exponents r, s, and t, it can derive a function key by the following equation

$$A_1 = \hat{g}^{\alpha_i r},\ A_1' = \hat{g}^{\alpha_j r},\ A_2^{1/(E+E')} = \left(\hat{g}^{\beta_i \cdot s}\right)^{1/(s\alpha_i + t + s\alpha_j - t)} = \left(\hat{g}^{\beta_i \cdot s}\right)^{1/(s\alpha_i + s\alpha_j)} = \hat{g}^{\beta_i/(\alpha_i + \alpha_j)}.$$

Since the correct function key is derived from the partial function key, it is guaranteed that the set intersection is correctly calculated from the ciphertexts of two clients from the correctness of the MCFE-SI scheme.

5.4 Security Analysis

Theorem 5.1. *The above DMCFE-SI scheme is static-IND secure with no corruptions in the random oracle model if the PRF scheme is secure and the Assumptions 2 and 3 hold.*

Proof. Suppose there exists an adversary that breaks the static-IND security of the DMCFE-SI scheme with no corruptions. We can assume that $I = \{1,\ldots,n\}$ and $\bar{I} = \emptyset$. Let $(X^*_{0,1},\ldots,X^*_{0,n})$ and $(X^*_{1,1},\ldots,X^*_{1,n})$ be the challenge tuples where $X^*_{b,i} = \{x^*_{b,i,1},\ldots,x^*_{b,i,\ell_i}\}$ and $|X^*_{b,i}| = \ell_i$. Let $Q = \{(i,j)\}$ be the indexes set of function key queries. We can derive a tuple $(E^*_1,\ldots,E^*_n)$ by calling $CIQ((X^*_{\mu,k}),Q)$ where μ is the challenge random bit of the security game. To argue that the adversary cannot win this game, we define a sequence of hybrid games $\mathbf{G}_0,\mathbf{G}_1,\mathbf{G}_2$, and $\mathbf{G}_3$. The game $\mathbf{G}_i$ is defined as follows:

Game $\mathbf{G}_0$. The first game $\mathbf{G}_0$ is the original security game defined in Definition 5.2.

Game $\mathbf{G}_1$. In this game $\mathbf{G}_1$, when processing partial function key queries, we change all shared keys $\{K_{i,j}\}$ derived by non-interactive key agreement to random elements.

Game $\mathbf{G}_2$. In this game, we modify the previous game $\mathbf{G}_1$ to generate random exponents r,s,t by using the a truly random function instead of using a pseudo-random function when processing partial function key queries.

Game $\mathbf{G}_3$. This game $\mathbf{G}_3$ is similar to the game $\mathbf{G}_2$ except that the challenge ciphertext components $\{C_{i,k}\}$ are generated as random for all $x^*_{\mu,i,k} \notin E^*_i$.

Game $\mathbf{G}_4$. This game $\mathbf{G}_4$ is slightly changed from the game $\mathbf{G}_3$. That is, the challenge temporal keys $\{TK_{i,k}\}$ are generated as random for all $x^*_{\mu,i,k} \notin E^*_i$.

Game $\mathbf{G}_5$. In the final game $\mathbf{G}_5$, we change the generation of challenge ciphertext components $\{D_{i,k}\}$. That is, the challenge ciphertext components $\{D_{i,k}\}$ are the encryption of random values for all $x^*_{\mu,i,k} \notin E^*_i$. Recall that the advantage of the adversary in this game is zero since challenge ciphertext components $\{C_{i,k}\}$ are random and $\{D_{i,k}\}$ are the encryption of random values for all $x^*_{\mu,i,k} \notin E^*_i$.

Let $S^{\mathbf{G}_i}_{\mathcal{A}}$ be the event that an adversary wins in a game $\mathbf{G}_i$. From the following lemmas 5.2, 5.3, 5.4, 5.5, and 5.6, we obtain the following result

$$\begin{aligned}\mathbf{Adv}^{ST\text{-}IND}_{DMCFE\text{-}SI,\mathcal{A}}(\lambda) \leq& \left|\Pr[S^{\mathbf{G}_0}_{\mathcal{A}}] - \Pr[S^{\mathbf{G}_5}_{\mathcal{A}}]\right| + \Pr[S^{\mathbf{G}_5}_{\mathcal{A}}] \leq \sum_{i=1}^{5}\left|\Pr[S^{\mathbf{G}_{i-1}}_{\mathcal{A}}] - \Pr[S^{\mathbf{G}_i}_{\mathcal{A}}]\right| + \Pr[S^{\mathbf{G}_5}_{\mathcal{A}}] \\ \leq& \mathbf{Adv}^{XDH}_{\mathcal{B}}(\lambda) + n^2\mathbf{Adv}^{PRF}_{\mathcal{B}}(\lambda) + n\ell\mathbf{Adv}^{A2\text{-}(n,\rho,Q,J)}_{\mathcal{B}}(\lambda) + n\ell\mathbf{Adv}^{A3\text{-}(n,\rho,Q)}_{\mathcal{B}}(\lambda) + \\ & n\ell\mathbf{Adv}^{SKE}_{\mathcal{B}}(\lambda)\end{aligned}$$

where n is the number of clients, ℓ is the maximum size of the challenge item set. This completes our proof. □

Lemma 5.2. *If the XDH assumption holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage.*

Proof. To prove this lemma, we introduce a multi-XDH assumption that is modified from the XDH assumption. Let $(p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e)$ be a bilinear group and $g,\hat{g}$ be random generators of $\mathbb{G},\hat{\mathbb{G}}$ respectively. The multi-XDH assumption is that if the challenge tuple $D = \big((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e),\ g,\ g^{a_1},\ldots,g^{a_n},\ \hat{g}\big)$ and Z

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = (g^{a_1 a_2}, \ldots, g^{a_1 a_n}, \ldots, g^{a_i a_j}, \ldots, g^{a_{n-1} a_n})_{1 \le i < j \le n}$ from $Z = Z_1 = (g^{c_{1,2}}, \ldots, g^{c_{i,j}} \ldots, g^{c_{n-1,n}})_{1 \le i < j \le n}$ with more than a negligible advantage where the probability is taken over random choices of $a_1, \ldots, a_n, \{c_{i,j}\} \in \mathbb{Z}_p$.

The multi-XDH assumption is actually the same as the XDH assumption by using the random self-reducibility of the XDH assumption. We omit the detailed proof of this lemma since the proof of randomly changing all shared keys is simply processed by using the multi-XDH assumption. □

Lemma 5.3. *If the PRF is secure, then no polynomial-time adversary can distinguish between $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ with a non-negligible advantage.*

Proof. To prove this lemma, we play additional hybrid games that convert pseudo-random functions into truly random functions one by one. When the number of clients is n, the maximum number of shared keys is $n(n-1)/2$, so the hybrid games consist of a maximum of $n^2/2$. Note that the exponents r, s, and t derived by a truly random function are distributed as random values. We omit the detailed proof of this lemma. □

Lemma 5.4. *If the Assumption 2 for (n, ρ, Q, J) holds, then no polynomial-time adversary can distinguish between $\boldsymbol{G}_2$ and $\boldsymbol{G}_3$ with a non-negligible advantage.*

Proof. The proof of this lemma is almost the same as Lemma 4.2 except for client public key generation and partial function key query processing. To perform the proof, we define a number of additional hybrid games as in Lemma 4.2 and show the indistinguishability of individual hybrid games. The simulator of this lemma generates public parameters, challenge ciphertexts, and challenge ciphertexts in the same manner as in Lemma 4.2. Note that function key query processing in Lemma 4.2 is unnecessary for this lemma. In the proof of individual hybrid games, the simulator handles additional client public key generation and partial function key query processing.

In the setup phase, the simulator selects a random exponent $\gamma_i \in \mathbb{Z}_p$ for each client and sets $h_i = g^{\gamma_i}$ as the corresponding client public key. The public key generated in this way has the same distribution as that of the original game.

In the query phase, the simulator handles a partial function key query for a function $f = (i, j)$ and a client index k as follows:

- Case $f = (i, j) \in Q$: It first sets a function key $DK_f = \left(K_1 = \hat{g}^{b_i c_{i,j}}, K_2 = \hat{g}^{b_j c_{i,j}}, K_3 = (\hat{g}^{1/(b_i + b_j)})^{\beta_i}\right)$ since these elements are given in the assumption. Next, it selects random exponents $r', s', t' \in \mathbb{Z}_p$. If $k = i$, then it creates $pDK_{i,f} = \left(A_1 = K_1^{r'}, A_2 = K_3^{s'}, E = s' + t' \mod p\right)$. Otherwise ($k = j$), it creates $pDK_{j,f} = \left(A'_1 = K_2^{r'}, A'_2 = 1_{\hat{\mathbb{G}}}, E' = -t' \mod p\right)$.

  Now we show that the distribution of the generated partial function keys has the same distribution as that of the original game. We implicitly define the random exponents of the partial function key as follows:

$$r = c_{i,j} r', \; s = \frac{1}{(b_i + b_j)} s', \; t = \frac{b_j}{(b_i + b_j)} s' + t'.$$

  Then, we can show that the elements of the partial function key are correctly distributed by the fol-

lowing equations:

$$A_1 = \hat{g}^{b_i r} = \hat{g}^{b_i c_{i,j} r'} = K_1^{r'},\ A_1' = \hat{g}^{b_j r} = \hat{g}^{b_j c_{i,j} r'} = K_2^{r'},\ A_2 = \hat{g}^{\beta_i s} = \hat{g}^{\beta_i \cdot s'/(b_i+b_j)} = K_3^{s'},$$
$$E = sb_i + t = \frac{s'}{(b_i+b_j)} b_i + \frac{b_j}{(b_i+b_j)} s' + t' = s' + t',$$
$$E' = sb_j - t = \frac{s'}{(b_i+b_j)} b_j - \frac{b_j}{(b_i+b_j)} s' - t' = -t'.$$

- Case $f = (i,j) \notin Q$: It first selects random exponents $r', s', t' \in \mathbb{Z}_p$. If $k = i$, then it creates $pDK_{i,f} = \big(A_1 = \hat{g}^{r'}, A_2 = \hat{g}^{s'}, E = t' \mod p\big)$. Otherwise ($k = j$), it creates $pDK_{j,f} = \big(A_1' = \hat{g}^{r'}, A_2' = 1_{\hat{\mathbb{G}}}, E' = t' \mod p\big)$.

  Now we should show that the distribution of the partial function keys generated in this way has the same distribution as that of the original game. Note that in the case of $f \notin Q$, an attacker can obtain only one of $pDK_{i,f}$ or $pDK_{j,f}$ due to the constraints of the security model. First, in the case of $k = i$, if we define the random exponents as follows, then we can see that the elements of the partial function key are correctly distributed by the following equations:

$$r = \frac{1}{b_i} r',\ s = \frac{1}{\beta_i} s',\ t = -\frac{b_i}{\beta_i} s' + t',$$

$$A_1 = \hat{g}^{b_i r} = \hat{g}^{b_i \cdot r'/b_i} = \hat{g}^{r'},\ A_2 = \hat{g}^{\beta_i s} = \hat{g}^{\beta_i \cdot s'/\beta_i} = \hat{g}^{s'},$$
$$E = sb_i + t = \frac{1}{\beta_i} s' b_i - \frac{b_i}{\beta_i} s' + t' = t'.$$

  Next, in the case of $k = j$, if we define the random exponents as follows, then we can see that the elements of the partial function key are correctly distributed by the following equations:

$$r = \frac{1}{b_j} r',\ s = \frac{1}{\beta_i} s',\ t = \frac{b_j}{\beta_i} s' - t',$$

$$A_1' = \hat{g}^{b_j r} = \hat{g}^{b_j \cdot r'/b_j} = \hat{g}^{r'},\ E' = sb_j - t = \frac{s'}{\beta_i} b_j - \frac{b_j}{\beta_i} s' + t' = t'.$$

This completes our proof. □

Lemma 5.5. *If the Assumption 3 for (n, ρ, Q, J) holds, then no polynomial-time adversary can distinguish between $\boldsymbol{G}_3$ and $\boldsymbol{G}_4$ with a non-negligible advantage.*

Proof. The proof of this lemma is the same as that of Lemma 4.3 by removing the function key query and adding additional client public key generation and partial function key query. In order to perform the proof, we define additional hybrid games, identical to Lemma 4.3, and perform indistinguishability proof of individual hybrid games. In the proof of individual hybrid games, a simulator proceeds client public key generation and partial function key query processing as the similar manner as in Lemma 5.4. We omit the detailed proof. □

Lemma 5.6. *If the SKE scheme is one-message secure, then no polynomial-time adversary can distinguish between $\boldsymbol{G}_4$ and $\boldsymbol{G}_5$ with a non-negligible advantage.*

Proof. The proof of this lemma is almost the same by removing the function key generation from the proof of Lemma 4.4, and adding client public key generation and partial function key query processing. A simulator can easily handle client public key generation and partial function key query by using α_i, β_i, and γ_i selected by the simulator. We omit the detailed description of this proof. □

Theorem 5.7. *The above DMCFE-SIC scheme is static-IND secure with corruptions in the random oracle model if the DMCFE-SIC scheme is static-IND secure with no corruptions.*

Proof. The proof of this theorem is almost the same as Theorem 4.5 by replacing the function key query with a partial function key query. In other words, the simulator of this theorem generates the secret keys of corrupted clients by itself, and partial function key queries requested by an attacker are also processed by using the queries of the DMCFE-SI scheme with no corruption. Since all other parts of this proof are the same as Theorem 4.5, we will omit the detailed proof. □

5.5 Discussions

Efficiency Analysis. The encryption and decryption algorithms of our DMCFE-SI scheme has the same performance as those of the MCFE-SI scheme in the previous section because they are the same. The partial function key generation algorithm requires three exponentiations and three PRF operations to generate random exponents. And the partial function key combination algorithm requires one inverse and one exponentiation operations. Therefore, generating and combining partial function key algorithms are very efficient.

Public Verification of Function Keys. A client that performs the partial function key combination algorithm may need to check whether the derived function key is correct or not. In order to publicly verify the function key, it is necessary to additionally expose public keys for private keys of individual clients. In other words, individual clients publishes a public key $(g^{\alpha_i}, e(g,\hat{g})^{\beta_i}, g^{\gamma_i})$ for their private key $(\alpha_i, \beta_i, \gamma_i)$. Since the function key is composed of $(\hat{g}^{\alpha_i r}, \hat{g}^{\alpha_j r}, \hat{g}^{\beta_i/(\alpha_i+\alpha_j)})$, it is possible to verify the function key by checking the following equations. $e(g^{\alpha_j}, \hat{g}^{\alpha_i r}) = e(g^{\alpha_i}, \hat{g}^{\alpha_j r}) \wedge e(g^{\alpha_i} g^{\alpha_j}, \hat{g}^{\beta_i/(\alpha_i+\alpha_j)}) = e(g,\hat{g})^{\beta_i}$. Note that it is secure for a client to expose $g^{\alpha_i}, e(g,\hat{g})^{\beta_i}$ in the public key since these elements are already included in the two assumptions used to prove the security of the DMCFE-SI scheme.

Decentralized Three-Party Set Intersection. Previously, we could extend the MCFE-SIC and MCFE-SI schemes to support the set intersection between multiple parties. Here, we extend our DMCFE-SI scheme to support multi-party set intersection. In the case of the DMCFE-SI scheme, the function key generation is divided into partial function key generation and partial function key combination algorithms. Therefore, it is necessary to modify the partial function key generation algorithm to support the multi-party set intersection. The partial function key generation algorithm needs to derive a shared key through non-interactive key exchange between entities involved in the set intersection. Fortunately, three-party non-interactive key exchange is possible by using the pairing operation. In other words, we first derive a shared key $K_{i,j,k} = e(g^{\gamma_i}, \hat{g}^{\gamma_j})^{\gamma_k}$ for three clients (i,j,k). We then select random exponents r_1, r_2, s, t_1, t_2 and set $r_3 = -r_1 - r_2, t_3 = -t_1 - t_2$. Then the partial key of the client i is $(g^{\alpha_i r_1}, g^{\beta_i s}, E_i = s\alpha_i + t_1)$, and the partial key of the client j is $(g^{\alpha_j r_2}, 1, E_j = s\alpha_j + t_2)$, and the partial key of the client k is $(g^{\alpha_k r_3}, 1, E_k = s\alpha_k + t_3)$. In this case, the correct function key $(\hat{g}^{\beta_i s})^{1/(E_i+E_j+E_k)} = \hat{g}^{\beta_i/(\alpha_i+\alpha_j+\alpha_k)}$ is derived from the partial function keys.

6 Generic Group Model

In this section, we describe the master theorem for analyzing complexity assumptions, and analyze that our three complexity assumptions hold in the generic group model of Shoup [28].

6.1 Master Theorem

We use the master theorems of by Lee and Seo [26] to analyze the complexity assumptions introduced in the previous section. Their master theorems are the extension of the well-known master theorems [6, 22] to asymmetric bilinear groups.

Let $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ be asymmetric bilinear groups of prime order p. The bilinear map is defined as $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. In the generic group model, a random group element of $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ is represented as a random variable P_i, R_i, T_i respectively where P_i, R_i, T_i are chosen uniformly in $\mathbb{Z}_p$. We say that a random variable has degree d if the maximum degree of any variable is d. The definition of dependence and independence is given as follows:

Definition 6.1. Let $P = \{P_1, \ldots, P_u\}$, $R = \{R_1, \ldots, R_w\}$, and $T = \{T_1, \ldots, T_v\}$ be random variables over $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ respectively. We let Z_0, Z_1 be random variables over $\mathbb{G}$. We say that Z_b is dependent on P if there exists constants $\alpha, \{\beta_i\}$ such that $\alpha \cdot Z_b = \sum_{i=1}^{u} \beta_i \cdot P_i$ where $\alpha \neq 0$. We say that Z_b is independent of P if Z_b is not dependent on P. We say that $\{e(Z_b, R_i)\}_i$ is dependent on $P \cup R \cup T$ if there exist constants $\{\alpha_i\}, \{\beta_{i,j}\}, \{\gamma_i\}$ such that $\sum_{i=1}^{w} \alpha_i \cdot e(Z_b, R_i) = \sum_{i=1}^{u} \sum_{j=1}^{w} \beta_{i,j} \cdot e(P_i, R_j) + \sum_{i=1}^{v} \gamma_i \cdot T_i$ where $\alpha_i \neq 0$ for at least one i. We say that $\{e(Z_b, R_i)\}_i$ is independent of $P \cup R \cup T$ if $\{e(Z_b, R_i)\}_i$ is not dependent on $P \cup R \cup T$.

Theorem 6.1 ([26]). *Let $P = \{P_1, \ldots, P_u\}$, $R = \{R_1, \ldots, R_w\}$, and $T = \{T_1, \ldots, T_v\}$ be random variables over $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ respectively. We let Z_0, Z_1 be random variables over $\mathbb{G}$. We consider the following experiment in the generic group model:*

> *An algorithm $\mathcal{A}$ is given $P = \{P_1, \ldots, P_u\}$, $R = \{R_1, \ldots, R_w\}$, and $T = \{T_1, \ldots, T_v\}$. A random bit b is chosen and Z_b is given to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a bit b' and succeeds if $b = b'$. The advantage of $\mathcal{A}$ is defined as $|\Pr[b = b'] - \frac{1}{2}|$.*

If Z_b is independent of P for all $b \in \{0,1\}$, and $\{e(Z_b, R_j)\}_j$ is independent of $P \cup R \cup T$ for all $b \in \{0,1\}$, then any algorithm $\mathcal{A}$ issuing at most q instructions has an advantage at most $3(q+2l)^2 d/p$ where $l = \max\{u, w, v\}$.

Definition 6.2. Let $P = \{P_1, \ldots, P_u\}$, $R = \{R_1, \ldots, R_w\}$, and $T = \{T_1, \ldots, T_v\}$ be random variables over $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ respectively. We let Z_0, Z_1 be random variables over $\mathbb{G}_T$. We say that Z_b is dependent on $P \cup R \cup T$ if there exist constants $\alpha, \{\beta_{i,j}\}, \{\gamma_i\}$ such that $\sum_{i=1}^{w} \alpha \cdot Z_b = \sum_{i=1}^{u} \sum_{j=1}^{w} \beta_{i,j} \cdot e(P_i, R_j) + \sum_{i=1}^{v} \gamma_i \cdot T_i$ where $\alpha \neq 0$. We say that Z_b is independent of $P \cup R \cup T$ if Z_b is not dependent on $P \cup R \cup T$.

Theorem 6.2 ([26]). *Let $P = \{P_1, \ldots, P_u\}$, $R = \{R_1, \ldots, R_w\}$, and $T = \{T_1, \ldots, T_v\}$ be random variables over $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ respectively. We let Z_0, Z_1 be random variables over $\mathbb{G}_T$. We consider the same experiment as in Theorem 6.1. If Z_b is independent of $P \cup R \cup T$ for all $b \in \{0,1\}$, then any algorithm $\mathcal{A}$ issuing at most q instructions has an advantage at most $3(q+2l)^2 d/p$ where $l = \max\{u, w, v\}$.*

6.2 Analysis of Assumption 1 for (n, ρ, Q, J)

We analyze the security of the Assumption 1 for (n, ρ, Q, J) in the generic group model by using Theorem 6.1. The Assumption 1 is described as follows:

$$D = \left(g, g^a, \{g^{b_k}\}_{k=1}^{n}, \{g^{ab_k}\}_{k \in J}, \hat{g}, \{(\hat{g}^{b_i c_{i,j}}, \hat{g}^{b_j c_{i,j}})\}_{(i,j) \in Q}\right),\ Z_0 = g^{ab_\rho},\ Z_1 = g^d.$$

The Assumption 1 is described again as the following set of random variables.

$$P = \{1, A\} \cup \{B_k\}_{k=1}^{n} \cup \{AB_k\}_{k \in J},\ R = \{1\} \cup \{B_i C_{i,j}, B_j C_{i,j}\}_{(i,j) \in Q},\ T = \{\},$$
$$Z_0 = AB_\rho,\ Z_1 = D.$$

To apply the master theorem to this assumption, we must show that the random variables Z_0 and Z_1 are independent of $P \cup R \cup T$ by following Definition 6.1. We can easily show that Z_1 is independent of P and $\{e(Z_1, R_j)\}$ is also independent of $P \cup R \cup T$ by using the fact that the random variable D in Z_1 does not exist in P, R, T. Next, we can also show that Z_0 is independent of P by using the fact that $\rho \notin J$ from the definition of J. To show that $\{e(Z_0, R_j)\}$ is independent of $P \cup R \cup T$, we derive two sets $\{e(Z_0, R_j)\}$ and $\{e(P_i, R_j)\}$ as follows:

$$\begin{aligned}
\{e(Z_0, R_j)\} =& \{AB_\rho\} \cup \{AB_\rho B_i C_{i,j}, AB_\rho B_j C_{i,j}\}_{(i,j) \in Q}, \\
\{e(P_i, R_j)\} =& \{1, A\} \cup \{B_k\}_{1 \le k \le n} \cup \{AB_k\}_{k \in J} \cup \\
& \{B_i C_{i,j}, B_j C_{i,j}\}_{(i,j) \in Q} \cup \{AB_i C_{i,j}, AB_j C_{i,j}\}_{(i,j) \in Q} \cup \\
& \{B_k B_i C_{i,j}, B_k B_j C_{i,j}\}_{(i,j) \in Q, 1 \le k \le n} \cup \{AB_k B_i C_{i,j}, AB_k B_j C_{i,j}\}_{(i,j) \in Q, k \in J}.
\end{aligned}$$

The set $\{e(Z_0, R_j)\}$ consists of three component types: AB_ρ, $AB_\rho B_i C_{i,j}$, and $AB_\rho B_j C_{i,j}$. Since these component types are independent of each other, we can analyze these types separately.

- First, we show that AB_ρ is independent of $\{e(P_i, R_j)\}$. At this time, since AB_ρ includes random variables A and B_ρ, only $\{AB_k\}$ can have a dependency. However, AB_ρ is independent because of $\rho \notin J$.
- Next, we show that $AB_\rho B_i C_{i,j}$ is independent of $\{e(P_i, R_j)\}$. The subsets of $\{e(P_i, R_j)\}$ that contain the random variables $A, B_\rho, B_i, C_{i,j}$ are $\{AB_k B_i C_{i,j}\}$. However, the index k cannot be the index ρ because of $\rho \notin J$. Thus $AB_\rho B_i C_{i,j}$ is independent.
- We can also show that $AB_\rho B_j C_{i,j}$ is independent similarly.

Therefore, we have that $\{e(Z_0, R_j)\}$ is independent of $P \cup R \cup T$.

6.3 Analysis of Assumption 2 for (n, ρ, Q, J)

We analyze the security of the Assumption 2 for (n, ρ, Q, J) in the generic group model by using Theorem 6.1. However, we cannot directly apply the theorem to the assumption because the assumption contains negative exponents. To solve this negative exponent problem, we set $\hat{h} = \hat{g}^{\prod_{(i,j) \in Q}(b_i + b_j)}$ and use $\hat{h}$ instead of $\hat{g}$. In this case, the Assumption 2 is described again as follows:

$$D = \left(g, g^a, \{g^{b_k}\}_{k=1}^{n}, \{g^{ab_k}\}_{k \in J}, \hat{h}, \{\hat{h}^{b_i c_{i,j}}, \hat{h}^{b_j c_{i,j}}, \hat{h}^{1/(b_i + b_j)}\}_{(i,j) \in Q}\right),\ Z_0 = g^{ab_\rho},\ Z_1 = g^d.$$

Let $\eta = \prod_{(i,j) \in Q}(B_i + B_j)$ be a random variable where the maximum degree of η is $n(n-1)/2$. The Assumption 2 is described again as the following set of random variables.

$$P = \{1, A\} \cup \{B_k\}_{k=1}^{n} \cup \{AB_k\}_{k \in J},$$
$$R = \{\eta\} \cup \{\eta B_i C_{i,j}, \eta B_j C_{i,j}, \eta/(B_i + B_j)\}_{(i,j) \in Q},\ T = \{\},$$
$$Z_0 = AB_\rho,\ Z_1 = D.$$

To apply the master theorem to this assumption, we must show that the random variables Z_0 and Z_1 are independent of $P \cup R \cup T$ by following Definition 6.1. We can easily show that Z_1 is independent of P and $\{e(Z_1, R_j)\}$ is also independent of $P \cup R \cup T$ by using the fact that the random variable D in Z_1 does not exist in P, R, T. Next, we can also show that Z_0 is independent of P by using the fact that $\rho \notin J$ from the definition of J. To show that $\{e(Z_0, R_j)\}$ is independent of $P \cup R \cup T$, we derive two sets $\{e(Z_0, R_j)\}$ and $\{e(P_i, R_j)\}$ as follows:

$$
\begin{aligned}
\{e(Z_0, R_j)\} =& \{\eta AB_\rho\} \cup \{\eta AB_\rho B_i C_{i,j}, \eta AB_\rho B_j C_{i,j}, \eta AB_\rho/(B_i + B_j)\}_{(i,j)\in Q}, \\
\{e(P_i, R_j)\} =& \{\eta, \eta A\} \cup \{\eta B_k\}_{1\le k\le n} \cup \{\eta AB_k\}_{k\in J} \cup \\
& \{\eta B_i C_{i,j}, \eta B_j C_{i,j}\}_{(i,j)\in Q} \cup \{\eta AB_i C_{i,j}, \eta AB_j C_{i,j}\}_{(i,j)\in Q} \cup \\
& \{\eta B_k B_i C_{i,j}, \eta B_k B_j C_{i,j}\}_{(i,j)\in Q, 1\le k\le n} \cup \{\eta AB_k B_i C_{i,j}, \eta AB_k B_j C_{i,j}\}_{(i,j)\in Q, k\in J} \cup \\
& \{\eta/(B_i + B_j)\}_{(i,j)\in Q} \cup \{\eta A/(B_i + B_j)\}_{(i,j)\in Q} \cup \\
& \{\eta B_k/(B_i + B_j)\}_{(i,j)\in Q, 1\le k\le n} \cup \{\eta AB_k/(B_i + B_j)\}_{(i,j)\in Q, k\in J}.
\end{aligned}
$$

The set $\{e(Z_0, R_j)\}$ consists of four component types: ηAB_ρ, $\eta AB_\rho B_i C_{i,j}$, $\eta AB_\rho B_j C_{i,j}$, and $\eta AB_\rho/(B_i + B_j)$. Since these component types are independent of each other, we can analyze these types separately.

- First, we show that ηAB_ρ is independent of $\{e(P_i, R_j)\}$. At this time, since ηAB_ρ includes random variables η, A, and B_ρ, only $\{\eta AB_k\}$ can have a dependency. However, ηAB_ρ is independent because of $\rho \notin J$.

- We show that $\eta AB_\rho B_i C_{i,j}$ is independent of $\{e(P_i, R_j)\}$. The subsets of $\{e(P_i, R_j)\}$ that contain the random variables $A, B_\rho, B_i, C_{i,j}$ are $\{\eta AB_k B_i C_{i,j}\}$. However, $\eta AB_\rho B_i C_{i,j}$ is independent because of $\rho \notin J = \{k\}$.

- We can also show that $\eta AB_\rho B_j C_{i,j}$ is independent similarly.

- Next, we show that $\eta AB_\rho/(B_i + B_j)$ is independent of $\{e(P_i, R_j)\}$. The subsets of $\{e(P_i, R_j)\}$ that contain the random variables η, A are $\{\eta A\}, \{\eta AB_k\}, \{\eta A/(B_i + B_j)\}$, and $\{\eta AB_k/(B_i + B_j)\}$. Here, the subset $\{\eta AB_k\}$ need not be considered because of $\rho \notin J$. The subset $\{\eta A/(B_i + B_j)\}$ does not need to be considered because it does not contain B_ρ. Now using the remaining subsets $\{\eta A = \eta A(B_i + B_j)/(B_i + B_j)\}$ and $\{\eta AB_k/(B_i + B_j)\}$, we may try to compose a linear equation with $\eta AB_\rho/(B_i + B_j)$. Here, the index k cannot be the index ρ because of $\rho \notin J$. Thus the only way to create a linear equation is to derive

  $$\frac{\eta AB_\rho}{(B_\rho + B_k)} = \frac{\eta A(B_\rho + B_k)}{(B_\rho + B_k)} - \frac{\eta AB_k}{(B_\rho + B_k)}$$

  when $(\rho, k) \in Q$. To satisfy the above equation, it is required that $k \in J$ when $(\rho, k) \in Q$. However, if $(\rho, k) \in Q$, we have $k \notin J$ according to the definition of J. Thus $\eta AB_\rho/(B_i + B_j)$ is independent because $AB_k \notin P$ when $(\rho, k) \in Q$.

Therefore, we have that $\{e(Z_0, R_j)\}$ is independent of $P \cup R \cup T$.

6.4 Analysis of Assumption 3 for (n, ρ, Q)

We analyze the security of the Assumption 3 for (n, ρ, Q) in the generic group model by using Theorem 6.2. However, we cannot directly apply the theorem to the assumption because the assumption contains negative

exponents. To solve this negative exponent problem, we set $\hat{h} = \hat{g}^{\prod_{(i,j)\in Q}(b_i+b_j)}$ and use $\hat{h}$ instead of $\hat{g}$. In this case, the Assumption 3 is described again as follows:

$$D = \big(g, g^a, \{g^{b_i}\}_{i=1}^n, \{g^{ab_k}\}_{1\le k\neq\rho\le n}, \hat{h}, \{\hat{h}^{b_i c_{i,j}}, \hat{h}^{b_j c_{i,j}}, \hat{h}^{d_i/(b_i+b_j)}\}_{(i,j)\in Q}, \{\hat{h}^{d_i}\}_{1\le i\neq\rho\le n}, e(g,\hat{h})^{d_\rho}\big),$$
$$Z_0 = e(g,\hat{h})^{ad_\rho},\ Z_1 = e(g,\hat{h})^{f}.$$

Let $\eta = \prod_{(i,j)\in Q}(B_i + B_j)$ be a random variable where the maximum degree of η is $n(n-1)/2$. The Assumption 3 is described again as the following set of random variables.

$$P = \{1, A\} \cup \{B_k\}_{k=1}^n \cup \{AB_k\}_{1\le k\neq\rho\le n},$$
$$R = \{\eta\} \cup \{\eta B_i C_{i,j}, \eta B_j C_{i,j}, \eta D_i/(B_i+B_j)\}_{(i,j)\in Q} \cup \{\eta D_i\}_{1\le i\neq\rho\le n},\ T = \{\eta D_\rho\},$$
$$Z_0 = \eta A D_\rho,\ Z_1 = \eta F.$$

To apply the master theorem to this assumption, we must show that the random variables Z_0 and Z_1 are independent of $P \cup R \cup T$ by following Definition 6.2. We can easily show that Z_1 is independent of $P \cup R \cup T$ by using the fact that the random variable F in Z_1 does not exist in P, R, T. To show that Z_0 is independent of $P \cup R \cup T$, we derive the set $\{e(P_i, R_j)\}$ as follows:

$$\begin{aligned}\{e(P_i,R_j)\} =& \{\eta, \eta A\} \cup \{\eta B_k\}_{i=k}^n \cup \{\eta AB_k\}_{1\le k\neq\rho\le n} \cup \{\eta D_i, \eta AD_i\}_{1\le i\neq\rho\le n} \cup \\ & \{\eta B_k D_i\}_{1\le i\neq\rho\le n, 1\le k\le n} \cup \{\eta AB_k D_i\}_{1\le i\neq\rho\le n, 1\le k\le n} \cup \\ & \{\eta B_i C_{i,j}, \eta B_j C_{i,j}\}_{(i,j)\in Q} \cup \{\eta AB_i C_{i,j}, \eta AB_j C_{i,j}\}_{(i,j)\in Q} \cup \\ & \{\eta B_k B_i C_{i,j}, \eta B_j B_k C_{i,j}\}_{(i,j)\in Q, 1\le k\neq\rho\le n} \cup \{\eta AB_k B_i C_{i,j}, \eta AB_k B_j C_{i,j}\}_{(i,j)\in Q, 1\le k\neq\rho\le n} \cup \\ & \{\eta D_i/(B_i+B_j)\}_{(i,j)\in Q} \cup \{\eta AD_i/(B_i+B_j)\}_{(i,j)\in Q} \cup \\ & \{\eta B_k D_i/(B_i+B_j)\}_{(i,j)\in Q, 1\le k\neq\rho\le n} \cup \{\eta AB_k D_i/(B_i+B_j)\}_{(i,j)\in Q, 1\le k\neq\rho\le n}.\end{aligned}$$

We show that $Z_0 = \eta AD_\rho$ is independent of $\{e(P_i,R_j)\} \cup T$. The subsets of $\{e(P_i,R_j)\}$ that contain the random variables A, D_ρ are $\{\eta AD_i/(B_i+B_j)\}$ and $\{\eta AB_k D_i/(B_i+B_j)\}$. Here, the subset $\{\eta AD_i/(B_i+B_j)\}$ does not need to be considered because it lacks (B_i+B_j). By using the remaining subset $\{\eta AB_k D_i/(B_i+B_j)\}$, we may try to compose a linear equation with ηAD_ρ. The only way to create a linear equation is to derive

$$\eta AD_\rho = \frac{\eta AB_{k_1} D_\rho}{(B_\rho + B_j)} + \frac{\eta AB_{k_2} D_\rho}{(B_\rho + B_j)}$$

when $(\rho, j) \in Q$, $k_1 = \rho$, and $k_2 = j$. To satisfy the above equation, it is required that $k_1 = \rho$ where k_1 is an index for $\{AB_k\}$. However, we have $k_1 \neq \rho$ from the restriction of the Assumption 3. Therefore Z_0 is independent of $P \cup R \cup T$.

7 Conclusion

In this paper, we proposed various MCFE schemes that support set intersection operations and proved the security of our schemes by using the newly introduced complexity assumptions. Our first MCFE-SIC scheme supports the computation of set intersection cardinality and can efficiently find matching ciphertext elements by using a pairing operation. Our second MCFE-SI scheme supports the set intersection operation and the pairing operation only requires 2ℓ times in the decryption. Our third DMCFE-SI scheme decentralizes the

generation of function keys by removing a trusted center. Using our MCFE-SI schemes, it is possible to construct an effective contact tracing system that preserves privacy of people.

We leave some interesting problems related to this study. The first problem is to devise an MCFE-SI scheme that is secure under standard assumptions. Since all of our MCFE-SI schemes have disadvantages that they are secure under complex and dynamic assumptions, it is an important problem to prove the security under weaker assumptions. The second problem is to devise an MCFE-SI scheme that can efficiently compute the set intersection between n patients and m users. If our MCFE-SI scheme is directly used, the computation requires $2nm\ell$ pairing operations with additional comparison operations. Thus if we can improve the performance, it can be used for more efficient contact tracing.

References

[1] Apple and google privacy-preserving contact tracing. `https://covid19.apple.com/contacttracing`, 2020.

[2] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *Public-Key Cryptography - PKC 2015*, volume 9020 of *Lecture Notes in Computer Science*, pages 733–751. Springer, 2015.

[3] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input inner-product functional encryption from pairings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 601–626. Springer, 2017.

[4] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016*, volume 9816 of *Lecture Notes in Computer Science*, pages 333–362. Springer, 2016.

[5] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015*, volume 9452 of *Lecture Notes in Computer Science*, pages 470–491. Springer, 2015.

[6] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

[7] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[8] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography - TCC 2011*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[9] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, *Theory of Cryptography - TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer, 2007.

[10] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 703–732. Springer, 2018.

[11] Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated PSI cardinality with applications to contact tracing. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020*, volume 12493 of *Lecture Notes in Computer Science*, pages 870–899. Springer, 2020.

[12] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

[13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS 2013*, pages 40–49. IEEE Computer Society, 2013.

[14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2014.

[15] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC 2013*, pages 555–564. ACM, 2013.

[16] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2012.

[17] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security - CCS 2006*, pages 89–98. ACM, 2006.

[18] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Ran Canetti, editor, *Theory of Cryptography - TCC 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 155–175. Springer, 2008.

[19] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed System Security Symposium - NDSS 2012*. The Internet Society, 2012.

[20] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In Stuart I. Feldman and Michael P. Wellman, editors, *ACM Conference on Electronic Commerce - EC-99*, pages 78–86. ACM, 1999.

[21] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - FC 2014*, volume 8437 of *Lecture Notes in Computer Science*, pages 195–215. Springer, 2014.

[22] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.

[23] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829. ACM, 2016.

[24] Kwangsu Lee. Efficient multi-client functional encryption for conjunctive equality and range queries. Cryptology ePrint Archive, Report 2020/822, 2020. `http://eprint.iacr.org/2020/822`.

[25] Kwangsu Lee and Dong Hoon Lee. Two-input functional encryption for inner products from bilinear maps. *IEICE Transactions*, 101-A(6):915–928, 2018.

[26] Kwangsu Lee and Minhye Seo. Functional encryption for set intersection in the multi-client setting. Cryptology ePrint Archive, Report 2020/1154, 2020. `http://eprint.iacr.org/2020/1154`.

[27] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium*, pages 797–812. USENIX Association, 2014.

[28] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[29] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *IEEE Data Eng. Bull.*, 43(2):95–107, 2020.

[30] Tim van de Kamp, David Stritzl, Willem Jonker, and Andreas Peter. Two-client and multi-client functional encryption for set intersection. In Julian Jang-Jaccard and Fuchun Guo, editors, *Information Security and Privacy - ACISP 2019*, volume 11547 of *Lecture Notes in Computer Science*, pages 97–115. Springer, 2019.