Efficient Multi-Client Functional Encryption for Conjunctive Equality and Range Queries

Kwangsu Lee*

Abstract

In multi-client functional encryption (MC-FE) for predicate queries, clients generate ciphertexts of plaintexts $x_1, \ldots, x_n$ binding with a time period T and store them on a cloud server, and the cloud server receives a function key corresponding to a predicate f from a trusted center and learns whether $f(x_1, \ldots, x_n) = 1$ or not by running the decryption algorithm on the multiple ciphertexts of the same time period. MC-FE for predicates can be used for a network event or medical data monitoring system based on time series data gathered by multiple clients. In this paper, we propose efficient MC-FE schemes that support conjunctive equality or range queries on encrypted data in the multi-client settings. First, we propose an efficient multi-client hidden vector encryption (MC-HVE) scheme in bilinear groups and prove the selective security with static corruptions. Our MC-HVE scheme is very efficient since a function key is composed of four group elements, a ciphertext consists of $O(\ell)$ group elements where ℓ is the size of a plaintext, and the decryption algorithm only requires four pairing operations. Second, we propose an efficient multi-client range query encryption (MC-RQE) scheme and prove the selective weak security with static corruptions. Our MC-RQE scheme which uses a binary tree is efficient since a ciphertext consists of $O(\log D)$ group elements and a function key consists of $O(n \log D)$ group elements where D is the maximum value of a range.

Keywords: Functional encryption, Predicate encryption, Multi-client setting, Hidden vector encryption, Bilinear maps.

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

Contents

1 Introduction

Functional encryption (FE) for predicate queries is a generalization of public-key encryption that can perform queries on encrypted data [14,27]. In FE for predicates, a sender creates a ciphertext CT by encrypting an input x, and then a receiver who has a function key SK_f associated with a predicate f can check whether $f(x) = 1$ or not by running the decryption algorithm without revealing the input x. FE for predicates can be viewed as a special type of general functional encryption (FE) that allows a function key holder to compute a specific function $f(x)$ on encrypted data without learning anything else about x [12]. FE for predicates can be widely used in situations where querying for encrypted data is required while preserving the privacy of original data such as network log auditing, payment gateway, email filtering, and so on. An initial study on FE for predicate queries began by supporting the equality query on encrypted data [10]. In addition, it has been shown that an FE scheme that supports arbitrary circuits can be constructed by using indistinguishability obfuscation at the theoretical level and this FE scheme also can be an FE scheme for arbitrary predicates [19].

In FE for predicates, many clients can generate ciphertexts by using public parameters since it is a public-key system, but the decryption algorithm of FE for predicates can only handle one ciphertext per each operation [14, 27]. If the decryption algorithm can process the query operation on a large number of ciphertexts rather than a single ciphertext, it would be possible to apply FE for predicates to a wider range of applications. One possible application is a network monitoring system that detects risks based on data gathered from multiple network devices. In this case, the distributed network devices independently encrypt the collected data and store the encrypted data in a cloud server. Then, the cloud server queries the abnormal behavior based on the gathered encrypted data for monitoring. For this kind of applications, we can construct a multi-client functional encryption (MC-FE) scheme for predicate queries by using previous MC-FE schemes. However, the previous MC-FE schemes have some problems such that they are based on inefficient indistinguishability obfuscation [22] or they only support a limited number of function key queries because of inner product operations [17].

We would like to build an efficient MC-FE scheme for predicate queries in which multiple clients generate ciphertexts independently and the decryption algorithm can proceed on these multiple ciphertexts. As pointed out above, we can use the MC-FE scheme that supports arbitrary circuits, but this scheme is inefficient due to indistinguishability obfuscation. We can consider extending a previous predicate encryption (PE) scheme to an MC-FE scheme for predicate queries, but it is not easy for the decryption algorithm to process multiple ciphertexts in a single decryption operation because of independent random values in ciphertexts. One possible idea for designing an efficient MC-FE scheme for predicate queries is to modify a hidden vector encryption (HVE) scheme, which is a special PE scheme that supports conjunctive equality [14], to support multiple clients. Kamp et al. [41] proposed the first MC-HVE scheme in bilinear groups by following this approach. However, their MC-HVE scheme is inefficient because the number of pairing operations linearly depends on the number of clients and the number of messages, and the security of their scheme is only proven in the generic group model. Thus, it is an interesting problem to build an efficient MC-HVE scheme with reduction proof under reasonable assumptions or to propose an efficient MC-FE scheme for more expressive predicate queries.

Recently, Chotard et al. [17] proposed an MC-FE for inner product (MC-FE-IP) scheme that supports inner product operations on encrypted data. Since HVE is a special form of FE and Katz et al. [27] proposed a transformation from an inner product predicate encryption (IPE) scheme into an HVE scheme, one can try to construct an MC-HVE scheme from an MC-FE-IP scheme. However, it is not easy to build an MC-HVE scheme from an MC-FE-IP scheme since their transformation only works for IPE, not FE-

IP[1]. The reason is that an MC-HVE scheme derived from an MC-FE-IP scheme exposes the inner product results in the decryption process and an attacker can distinguish a challenge ciphertext by using the exposed partial information of ciphertexts if he can obtain enough function keys. A detailed attack is described as follows: Let x_0, x_1 be challenge messages and y_1, y_2 be function attributes of HVE such that $x_0, x_1 \notin \{y_1, y_2\}$. By applying the transformation of Katz et al., the challenge message x_b is encoded into $(rx_b, -r)$ where r is random and the function attributes y_1, y_2 are encoded into $(1, y_1)$ and $(1, y_2)$ respectively. An adversary can obtain $g^{r(x_b - y_1)}$ and $g^{r(x_b - y_2)}$ by decrypting the challenge ciphertext with two function keys. Next, the adversary derives $g^{r(y_1 - y_2)}$ and distinguishes the challenge ciphertext by checking whether $(g^{r(y_1-y_2)})^{(x_0-y_1)/(y_1-y_2)} = g^{r(x_b-y_1)}$ or not. Thus the adversary can easily break the security of this method with two function keys. Note that function key queries for y_1 and y_2 are allowed in HVE since $f_{y_i}(x_0) = 0 = f_{y_i}(x_1)$ by the definition $f_y(x) = 0$ if $x \neq y$, but function key queries for $(1, y_1)$ and $(1, y_2)$ are not allowed in FE-IP since $f_{(1,y_i)}((rx_0, -r)) = \langle (rx_0, -r), (1, y_i) \rangle \neq \langle (rx_1, -r), (1, y_i) \rangle = f_{(1,y_i)}((rx_1, -r))$. Recall that FE-IP avoid this subtle issue by limiting the number of independent function key queries in the security model. Unlike FE-IP, FE for predicate queries only reveals 0 or 1 in the decryption process, so it is allowed for the attacker to request many independent function keys for a predicate f with the constraint $f(X_0^*) = f(X_1^*) \in \{0, 1\}$ where X_0^*, X_1^* are challenge message vectors. Therefore, it is difficult to design an MC-HVE scheme that can allow many independent function keys by using an MC-FE-IP scheme.

1.1 Our Results

In this work, we propose efficient MC-FE schemes that support conjunctive equality or range queries and prove the selective security of the proposed schemes. Our results are summarized as follows.

Multi-Client Hidden Vector Encryption. We first propose an efficient and secure multi-client hidden vector encryption (MC-HVE) scheme that supports conjunctive equality queries in asymmetric bilinear groups. In MC-HVE, each ciphertext is associated with a message vector $\vec{x}_i = (x_{i,1}, \ldots, x_{i,\ell})$ with a time period T and a function key is associated with function attributes $Y = (\vec{y}_1, \ldots, \vec{y}_n)$ where $\vec{y}_i = (y_{i,1}, \ldots, y_{i,\ell})$. A conjunctive equality predicate for ciphertexts with the same time period and a function key outputs 1 if $x_{i,j} = y_{i,j}$ for all i and j except wildcard positions. Our MC-HVE scheme is very efficient than the previous MC-HVE scheme of Kamp et al. [41] because the decryption algorithm requires only four pairing operations and a function key consists of just four group elements. Our MC-HVE scheme is proven to be selectively secure with static corruptions in the random oracle model. Note that it provides the selective strong security that allows an attacker to query any function key for a predicate f that satisfies $f(X_0^*) = f(X_1^*)$ where X_0^*, X_1^* are challenge message vectors. Additionally, our MC-HVE scheme can be proven without random oracles if the maximum value of a time period is limited to a polynomial value by increasing the size of public parameters. Furthermore, our MC-HVE scheme also can be extended to support more expressive queries by following the HVE encoding method of Boneh and Waters [14].

Multi-Client Range Query Encryption. Next, we propose a multi-client range query encryption (MC-RQE) scheme that supports conjunctive range queries on multiple ciphertexts by combining a binary tree with the simple version of our MC-HVE scheme. The simplest way to support range queries is to apply the HVE encoding method of Boneh and Waters [14] to the MC-HVE scheme. However, this encoding method is inefficient since a ciphertext is composed of $O(nD)$ group elements where n is the number of clients and D is the maximum value of the range. In contrast, our MC-RQE scheme is more efficient because a ciphertext

[1]IPE and FE-IP are similar in that a ciphertext is associated with an attribute vector $\vec{x}$ and a function key is associated with an attribute vector $\vec{y}$, and decryption is related to the inner product operation $\langle \vec{x}, \vec{y} \rangle$. However, the decryption of IPE outputs a binary value indicating that $\langle \vec{x}, \vec{y} \rangle = 0$ or not, whereas the decryption of FE-IP outputs the inner product value $\langle \vec{x}, \vec{y} \rangle$ itself.

is composed of $n \log D$ group elements and a function key is composed of $O(n \log D)$ group elements. We prove that our MC-RQE scheme provides the selective security with static corruptions under well-known static assumptions in the random oracle model. Our MC-RQE scheme only achieves the selective weak security which allows for an attacker to query any function key satisfying $f(X_0^*) = f(X_1^*) = 0$.

1.2 Our Techniques

In order to design a private-key HVE scheme, we can consider a simple method that encrypts plaintexts by using a pseudo-random function (PRF). However, this PRF-based construction has a problem such that the additional information of plaintexts is leaked because the outputs of PRF on the same plaintexts are the same. To solve this leakage problem, we introduce additional random values and combine them with a cyclic group to form a ciphertext as (g^t, g^{ft}) and a function key as (v^{fr}, v^r) after calculating $f = PRF(z,x)$ for a message x. However, when this method is extended to support multiple clients, it is not easy to check the equality of a message in a ciphertext and an attribute in a function key since individual clients use different random exponents t_1 and t_2.

To overcome this problem of extending the simple HVE scheme to support multiple clients, we use a method of using a hash function H to synchronize the random values of all clients. For example, two clients generate ciphertexts $H(T)^{f_1}$ and $H(T)^{f_2}$, respectively for a time period T, and a center generates a function key $(v^{(f_1+f_2)r}, v^r)$ where $f_1 = PRF(z_1,x_1)$ and $f_2 = PRF(z_2,x_2)$. In this case, even though two clients perform encryption independently, they are forced to use the same random element $H(T) = g^t$. In addition, we modify the function key of this MC-HVE scheme to include additional random values to prove the (strong) security under static assumptions. That is, the ciphertexts are formed as $(H(T)^{f_1}, H(T)^{w'_{1,1}}, H(T)^{w'_{1,2}})$, $(H(T)^{f_2}, H(T)^{w'_{2,1}}, H(T)^{w'_{2,2}})$ respectively, and the function key is formed as $(v^{(f_1+f_2)r_1} w_1^{r_2} w_2^{r_3}, v^{r_1}, v^{r_2}, v^{r_3})$ where $w_1 = v^{w'_{1,1}+w'_{2,1}}$, $w_2 = v^{w'_{1,2}+w'_{2,2}}$, and r_1, r_2, r_3 are random values. The details of our design principle are given in Section 3.1.

The idea of designing an MC-RQE scheme that supports efficient range queries is to express the range of values using a binary tree. That is, the value of a range is represented by the path of a binary tree, and a range is represented by the minimum set of internal nodes that cover all leaf nodes corresponding to the range in the binary tree. At this time, each node of the binary tree is set to be associated with the ciphertext and the function key elements of the simple version of our MC-HVE scheme. In addition, we modify this scheme to apply a secret sharing scheme to prevent collusion attacks that derive a new function key by combining different function keys.

1.3 Applications

Secure Network Event Monitoring. Hacking attacks on computer systems tend to change from simple attacks targeting a single system to wide-scale organized attacks targeting multiple systems. Thus, it is difficult to grasp such recent network hacking attacks by only monitoring the events of an individual network system. It is necessary to collect information from many network devices and judge a hacking event on the basis of the collected information [32]. One possible solution is for network devices of many companies to collect information at each time period and entrust it to an external security surveillance cloud server. This cloud server can then analyze the collected information to identify hacking attacks. In this case, these companies want to maintain the privacy of the network information they gathered because they do not trust the external surveillance cloud server. By using efficient MC-HVE or MC-RQE scheme, it is possible to build a secure network event monitoring system which allows the cloud server to monitor the network status while preserving the privacy of the information collected by individual devices.

Privacy-Preserving Medical Data Monitoring. With the development of small medical devices, it is possible to collect medical data of a patient by using these devices that monitor various types of medical information [33]. To diagnose the medical condition of the patient, medical devices collect specialized medical data and store this data in a central cloud server managed by a hospital. The cloud server can then monitor the periodic status of the patient or determine an emergency situation based on the medical data collected by the devices of each patient. However, patients do not want their personal medical data to be exposed for privacy reason. One method to perform health monitoring with ensuring privacy is to use an efficient MC-HVE or MC-RQE scheme. That is, medical devices of a patient periodically store encrypted medical data in a cloud server, and the cloud server performs a query operation on the encrypted data to check the current status of the patient.

1.4 Related Work

Searchable Encryption. Searchable symmetric encryption (SSE) is symmetric-key encryption that allows a data owner to outsource encrypted data to a cloud server and then allow keyword searches on the encrypted data. A practical SSE scheme for keyword searches on encrypted data was first proposed by Song et al. [40]. Later, an extended SSE scheme that supports conjunctive keyword searches was proposed by Golle et al. [23]. To improve the performance of search queries, Cash et al. [16] proposed a highly efficient SSE scheme that can handle a large database with supporting conjunctive search queries. A multi-client SSE scheme was presented by Curtmola et al. [18] which one client outsources the encrypted data to a cloud server and other clients can submit search queries on the encrypted data. Although the previous SSE schemes are very efficient, most SSE schemes have additional leakage. Cash et al. [15] presented leakage-abuse attacks on most SSE schemes and showed the danger of many SSE schemes. Recently, Lai et al. [28] proposed an SSE scheme that combines a symmetric-key HVE scheme with Bloom filter indexing to support conjunctive keyword searches with reduced leakage.

Functional Encryption. Functional encryption (FE) is a generalization of public-key encryption that allows fine grained control on disclosure of encrypted data. In FE, a ciphertext CT is associated with an input x, a function key SK_f is associated with a function f, and decryption outputs $f(x)$ but nothing else about x is revealed. The concept of FE was introduced by Boneh, Sahai, and Waters [12, 13] and they showed that public-key cryptographic primitives such as identity-based encryption (IBE) [11], attribute-based encryption (ABE) [26, 35], and predicate encryption (PE) [14, 27] are special cases of FE[2]. The first FE scheme for general circuits was proposed by Garg et al. [19] by using indistinguishability obfuscation, public-key encryption, and non-interactive zero-knowledge proof. In the bounded collusion model, FE for general circuits can be constructed from public-key encryption or identity-based encryption [6, 24, 34]. By extending the concept of FE, Goldwasser et al. [22] proposed a multi-input functional encryption (MI-FE) scheme that handles multiple ciphertexts in a single decryption process and a multi-client functional encryption (MC-FE) scheme that performs the decryption on multiple ciphertexts generated by multiple clients with the same label. Recently, efficient FE schemes that support inner products and quadratic functions were proposed [2, 5, 7, 8]. These FE schemes for inner products and quadratic functions also can be extended to the multi-input and multi-client settings [1, 3, 4, 7, 17, 30].

Predicate Encryption. A special type of functional encryption is predicate encryption (PE) in which a ciphertext is associated with an input $x = (v, m)$ where v is a hidden attribute and m is a message, and a

[2]For instance, if a ciphertext is associated with an input $x = (\varphi, m)$ where φ is an access policy and m is a message, a function key is associated with a function f_u for an attribute u such that the function f_u is defined as $f_u((\varphi, m)) := m$ if and only if $\varphi(u) = 1$, then this FE is ciphertext-policy ABE.

function key is associated with a predicate f, and decryption outputs the message m if $f(v) = 1$ is satisfied. Predicate-only encryption (POE) is a specific form of PE in which a binary value $f(v) \in \{0,1\}$ is only revealed in the decryption process. PE was originally devised for public-key searchable encryption, and anonymous IBE is one of the simplest PE [10]. Boneh and Waters [14] introduced the concept of hidden vector encryption (HVE) that supports conjunctive equality queries with a wildcard character and proposed an efficient HVE scheme in bilinear groups. Katz et al. [27] proposed an inner-product predicate encryption (IPE) scheme that supports more expressive inner product queries. After that, various PE schemes have been proposed to provide better efficiency or additional functionality [9, 29, 36, 39]. Shi et al. [37] proposed a multi-dimensional range query on encrypted data (MRQED) scheme that supports efficient conjunctive range queries, and showed that it provides weak attribute hiding security, named the match-revealing security. Later, Lu [31] proposed a symmetric-key PE scheme for range queries that provides strong attribute hiding security. Gay et al. [20] showed that an MRQED scheme can be built in lattices. Gorbunov et al. [25] showed that it is possible to design a PE scheme that supports arbitrary circuits using lattices, but it only guarantees weak attribute hiding security. Recently, Kamp et al. [41] proposed a multi-client HVE (MC-HVE) scheme that supports conjunctive equality queries on multiple clients and analyzed its security in the generic group model.

2 Preliminaries

In this section, we first define asymmetric bilinear groups, complexity assumptions, and pseudo-random functions. Next, we define the syntax and the security of multi-client functional encryption for predicates.

2.1 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 $\hat{\mathbb{G}}$.

2.2 Complexity Assumptions

We introduce two complexity assumptions in asymmetric bilinear groups. The symmetric external Diffie-Hellman (SXDH) assumption is that the decisional Diffie-Hellman (DDH) assumption holds in two cyclic groups $\mathbb{G}$, $\hat{\mathbb{G}}$ in asymmetric bilinear groups. The asymmetric 3-party Diffie-Hellman (A3DH) assumption is the asymmetric bilinear group version of the composite 3-party Diffie-Hellman (C3DH) assumption, which was introduced by Boneh and Waters [14] to prove the security of their HVE scheme in composite order bilinear groups. The generalization of this C3DH assumption was used to prove the security of predicate encryption schemes [36, 39].

Assumption 1 (Symmetric eXternal Diffie-Hellman, SXDH)**.** 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. The decisional Diffie-Hellman (DDH) assumption in $\mathbb{G}$ is that if the challenge tuple

$$D = \left((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, g^a, g^b, \hat{g}\right) \text{ and } Z$$

are given, no probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = g^{ab}$ from $Z = Z_1 = g^c$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{DDH}(\lambda) = \left|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\right|$ where the probability is taken over random choices of $a, b, c \in \mathbb{Z}_p$. The SXDH assumption is that the DDH assumption holds in both $\mathbb{G}$ and $\hat{\mathbb{G}}$.

Assumption 2 (Asymmetric 3-party Diffie-Hellman, A3DH)**.** 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. The A3DH assumption is that if the challenge tuple

$$D = \left((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, g^a, g^b, g^{ab}, g^{abc}, \hat{g}, \hat{g}^a, \hat{g}^b\right) \text{ and } Z$$

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = g^c$ from $Z = Z_1 = g^d$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{A3DH}(\lambda) = \left|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\right|$ where the probability is taken over random choices of $a, b, c, d \in \mathbb{Z}_p$.

2.3 Pseudo-Random Function

A pseudo-random function (PRF) is an efficiently computable function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ where $\mathcal{K}$ is the key space, $\mathcal{X}$ is the domain, and $\mathcal{Y}$ is the 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 is secure if for all efficient adversaries $\mathcal{A}$ the advantage $\mathbf{Adv}_{\mathcal{A}}^{PRF}(\lambda) = \left|\Pr[\mathcal{A}^{F(k,\cdot)} = 1] - \Pr[\mathcal{A}^{f(\cdot)} = 1]\right|$ is negligible.

2.4 Multi-Client Functional Encryption for Predicates

Multi-client functional encryption (MC-FE) for predicates is a symmetric-key version of multi-input functional encryption (MI-FE) introduced by Goldwasser et al. [22] in which each client holds an encryption key, a ciphertext is additionally associated with a time period, and the output of a function f is restricted to be $\{0, 1\}$. A system that uses an MC-FE scheme for predicates consists of a trusted center that generates function keys, a number of clients that create ciphertexts on plaintexts, and a cloud server that performs decryption operations on encrypted plaintexts. First, the trusted center performs the setup algorithm to generate a master key MK, each encryption key EK_i for each client, and the public parameter PP. Each client generates a ciphertext CT_i by encrypting a message x_i for a time period T and transmits the ciphertext to the cloud server. The cloud server receives a function key SK_f for a predicate f from the trusted center. If the ciphertexts $CT_1, \ldots, CT_n$ sent by the clients are generated at the same time period T, then the cloud server can obtain $f(x_1, \ldots, x_n) \in \{0, 1\}$ by running the decryption algorithm. The detailed syntax of the MC-FE scheme for predicates is described as follows.

Definition 2.1 (Multi-Client Functional Encryption for Predicates)**.** A multi-client functional encryption (MC-FE) scheme for predicates 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, encryption keys $EK_1, \ldots, EK_n$ for all clients, and public parameters PP.

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

Encrypt(x_i, T, EK_i, PP). The encryption algorithm takes as input a message x_i, a time period T, an encryption key EK_i for a client index i, and public parameters PP. It outputs a ciphertext $CT_{i,T}$.

Decrypt$(CT_{1,T}, \ldots, CT_{n,T}, SK_f, PP)$. The decryption algorithm takes as input ciphertexts $CT_{1,T}, \ldots, CT_{n,T}$ for the same time period T which are associated with messages $X = (x_1, \ldots, x_n)$, a function key SK_f for a predicate f, and public parameters PP. It outputs $f(X) \in \{0,1\}$.

The correctness property of the MC-FE scheme for predicates is defined as follows: For all $MK, EK_1, \ldots, EK_n$, PP generated by **Setup**, any SK_f generated by **GenKey** for any predicate f, and all $CT_{1,T}, \ldots, CT_{n,T}$ generated by **Encrypt** for any list of messages $X = (x_1, \ldots, x_n)$ with the same time period T, it is required that

- **Decrypt**$(CT_{1,T}, \ldots, CT_{n,T}, SK_f, PP) = f(X)$ except with negligible probability.

We define the selective security model with static corruptions of MC-FE for predicates by following the security model of MC-FE defined by Goldwasser et al. [22]. In this security model, an attacker first specifies corrupted clients and all challenge ciphertext queries in which each ciphertext query is specified by two challenge messages X_0, X_1 and a challenge time period T, and then it receives the encryption keys of corrupted clients and public parameters. After that, the attacker can query a function key for a predicate f with a constraint $f(X_0) = f(X_1)$. And the attacker also queries a challenge ciphertext for challenge messages X_0, X_1 and a time period T that were submitted initially and receives challenge ciphertexts that are the encryption of one of the two challenge messages. Finally, if the challenge message is correctly guessed, the attacker wins this game. The detailed definition of this selective security is described as follows.

Definition 2.2 (Selective Security)**.** The selective multiple-challenge IND-security with static corruptions of an MC-FE scheme for predicates is defined in the following experiment $\mathbf{EXP}_{\mathcal{A}}^{SE\text{-}IND}(\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ initially submits the set of corrupted client indexes $\bar{I} \subset \{1, \ldots, n\}$. Let $I = \{1, \ldots, n\} \setminus \bar{I}$ be the set of uncorrupted client indexes. $\mathcal{A}$ additionally submits all challenge ciphertext queries. Each challenge ciphertext query is specified with two challenge messages $X_0 = \{x_{0,i}\}_{i \in I}$, $X_1 = \{x_{1,i}\}_{i \in I}$, and a challenge time period T with the restriction that the time period should be distinct between each query. $\mathcal{C}$ flips a random coin $\mu \in \{0,1\}$.

2. **Setup**: $\mathcal{C}$ generates a master key MK, encryption keys $EK_1, \ldots, EK_n$, and public parameters PP by running **Setup**$(1^\lambda, n)$. It keeps MK, $\{EK_i\}_{i \in I}$ to itself and gives PP, $\{EK_i\}_{i \in \bar{I}}$ to $\mathcal{A}$.

3. **Query & Challenge**: $\mathcal{A}$ adaptively requests function keys or challenge ciphertexts. $\mathcal{C}$ handles these queries as follows:

   - If this is a function key query for a predicate f with the restriction that $f(X_0, \cdot) = f(X_1, \cdot)$ for each challenge X_0, X_1, then $\mathcal{C}$ gives a function key SK_f to $\mathcal{A}$ by running **GenKey**(f, MK, PP) where $\cdot$ indicates the messages of corrupted clients.
   - If this is a challenge ciphertext query for challenge messages X_0, X_1, and a time period T that were already submitted in the initialization step, then $\mathcal{C}$ gives challenge ciphertexts $\{CT_{i,T}\}$ to $\mathcal{A}$ by running **Encrypt**$(x_{\mu,i}, T, EK_i, PP)$ for each i.

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

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

Handling multiple challenge ciphertexts in the selective security proof is rather complicated. To facilitate the proof of the selective security, we define the selective single-challenge security that considers single-challenge ciphertext. Fortunately, it is known through existing studies that the single-challenge security and the multiple-challenge security are the same by using a simple hybrid argument [1, 22].

Definition 2.3 (Selective Single-challenge Security)**.** The selective single-challenge IND-security with static corruptions of an MC-FE scheme for predicates is defined in the following experiment $\mathbf{EXP}_{\mathcal{A}}^{SE\text{-}1\text{-}IND}(\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ initially submits the set of corrupted client indexes $\bar{I} \subset \{1,\ldots,n\}$. Let $I = \{1,\ldots,n\} \setminus \bar{I}$ be the set of uncorrupted client indexes. $\mathcal{A}$ also submits two challenge messages $X_0^* = \{x_{0,i}^*\}_{i\in I}$ and $X_1^* = \{x_{1,i}^*\}_{i\in I}$, and a challenge time period T^*. $\mathcal{C}$ flips a random coin $\mu \in \{0,1\}$.

2. **Setup**: $\mathcal{C}$ generates a master key MK, encryption keys $EK_1,\ldots,EK_n$, and public parameters PP by running $\mathbf{Setup}(1^\lambda, n)$. It keeps MK, $\{EK_i\}_{i\in I}$ to itself and gives PP, $\{EK_i\}_{i\in\bar{I}}$ to $\mathcal{A}$.

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

   - If this is a function key query for a predicate f with the restriction that $f(X_0^*,\cdot) = f(X_1^*,\cdot)$, then $\mathcal{C}$ gives a function key SK_f to $\mathcal{A}$ by running $\mathbf{GenKey}(f, MK, PP)$ where $\cdot$ indicates the messages of corrupted clients.
   - If this is a ciphertext query for a client index $i \in I$, a message x_i, and a time period T with the restriction that $T \neq T^*$, then $\mathcal{C}$ gives a ciphertext $CT_{i,T}$ to $\mathcal{A}$ by running $\mathbf{Encrypt}(x_i, T, EK_i, PP)$.

4. **Challenge**: $\mathcal{C}$ gives challenge ciphertexts $\{CT_{i,T^*}\}_{i\in I}$ to $\mathcal{A}$ by running $\mathbf{Encrypt}(x_{\mu,i}^*, T^*, EK_i, PP)$ for each $i \in I$.

5. **Query**: $\mathcal{A}$ additionally requests function keys or ciphertexts. $\mathcal{C}$ handles these queries in a similar way to the previous query step.

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

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

Remark 1. An MC-FE scheme for predicates is selectively single-challenge weak IND-secure with static corruptions if each function key query for a predicate f is restricted to $f(X_0^*) = f(X_1^*) = 0$.

3 Multi-Client Hidden Vector Encryption

In this section, we present an efficient MC-HVE scheme in asymmetric bilinear groups and analyze the security of our scheme under static assumptions.

3.1 Design Principle

To understand the design principle of our MC-HVE scheme, we consider a simple example in which two clients encrypt messages x_1 and x_2, respectively. We first note that an MC-HVE scheme is a private-key setting since each client uses an independent encryption key. A simple method to achieve a conjunctive equality operation in the symmetric-key setting is to use a pseudo-random function (PRF). That is, individual clients encrypt messages x_1, x_2 as $f_1 = PRF(z_1, x_1), f_2 = PRF(z_2, x_2)$ by using a PRF and encryption keys z_1, z_2, and a trusted center generates a function key for the list of attributes $Y = (y_1, y_2)$ as $f_Y = f_1 + f_2 = PRF(z_1, y_1) + PRF(z_2, y_2)$. In this case, it is easy to check the conjunctive equality of PRF values by simply comparing the PRF values. However, this method has a problem such that ciphertexts leak the partial information of messages since the outputs of the PRF are deterministic. In addition, it is uncertain how to bind a ciphertext with a time period T to prevent mix-and-match attacks.

In order to solve the partial information leakage problem, the PRF values should not be directly exposed to an attacker, and the ciphertexts should be randomized by including additional random values. To do this, we introduce a cyclic group $\mathbb{G}$ and form ciphertexts for each client as $(g^t, g^{f_1 t})$ and $(g^t, g^{f_2 t})$ in which the PRF values are placed in the exponent of an element $g \in \mathbb{G}$ and an additional random exponent t is multiplied. If the DDH assumption holds in the cyclic group, then these ciphertexts do not reveal the partial information since these elements are not distinguished from random elements. A function key can also be constructed in the form of $(v^{(f_1+f_2)r}, v^r)$ using an additional random exponent r to prevent partial information leakage. The problem of this method is that the handling of conjunctival equality queries is difficult since ciphertexts and function keys are randomized. Fortunately, if an asymmetric bilinear group is used, it is possible to compute $e(g^t, v^{(f_1+f_2)r}) = e(g^{(f_1+f_2)t}, v^r)$ to check whether the conjunctive equality is satisfied or not.

However, in the multi-client setting, two clients generate ciphertexts by using different random exponents t_1 and t_2 rather than the same random exponent t. Because of this different randomness, it is difficult to perform the conjunctive equality query by using the pairing operation since the exponent values $f_1 t_1$ and $f_2 t_2$ are not aggregated into $(f_1 + f_2)(t_1 + t_2)$. To overcome this problem, we adopt the method of using a hash function which was used to build a synchronized aggregate signature scheme [21] and to design a privacy-preserving data aggregation scheme [38]. That is, a hash value $H(T)$ for a time period T is used to create a ciphertext instead of using g^t. In this case, client ciphertexts consist of $H(T)^{f_1}, H(T)^{f_2}$ and a function key has the same form as before $(v^{(f_1+f_2)r}, v^r)$. Here, two clients can agree the same random element by computing $H(T)$ for the same time period T, and the attacker cannot obtain the partial information of the message without solving the discrete logarithm of $H(T) = g^t$.

In the above construction, if the attacker does not request function key queries, then the construction can be secure because the structure of ciphertexts is related to the DDH assumption. However, it is not easy to prove the security of the above construction in the selective security model since an attacker can query not only non-matching function key queries but also matching function key queries. In particular, in the security model, the attacker is allowed to query on an arbitrary predicate f satisfying $f(X_0^*) = f(X_1^*)$. That is, not only the non-matching function key query of $f(X_0^*) = f(X_1^*) = 0$ but also the matching function key query of $f(X_0^*) = f(X_1^*) = 1$ should be allowed. To prove the selective security, we modify the above construction to include additional two random exponents in a function key. That is, a function key is formed as $(v^{(f_1+f_2)r_1} w_1^{r_2} w_2^{r_3}, v^{r_1}, v^{r_2}, v^{r_3})$ and two ciphertexts of clients are formed as $(H(T)^{f_1}, H(T)^{w'_{1,1}}, H(T)^{w'_{1,2}})$ and $(H(T)^{f_2}, H(T)^{w'_{2,1}}, H(T)^{w'_{2,2}})$ where $w_1 = v^{w'_{1,1}+w'_{2,1}}$ and $w_2 = v^{w'_{1,2}+w'_{2,2}}$. This method of adding additional random exponents to the function key is widely used in the construction of previous HVE schemes [14, 29, 39]. Specifically, we use the proof technique of Shi and Waters [39], used for the construction of an efficient delegatable HVE scheme, by associating two random exponents r_2, r_3 in a correlated way to handle the matching function key queries in the security proof.

3.2 Predicate for Conjunctive Equality with Wildcards

Let Σ be a finite set of attributes and let $*$ be a special symbol not in Σ. We define $\Sigma_* = \Sigma \cup \{*\}$ where the star $*$ plays the role of a wildcard character. Let $Y = (\vec{y}_1, \ldots, \vec{y}_n)$ be a list of vectors where $\vec{y}_i = (y_{i,1}, \ldots, y_{i,\ell}) \in \Sigma_*^\ell$ and $X = (\vec{x}_1, \ldots, \vec{x}_n)$ be a list of vectors where $\vec{x}_i = (x_{i,1}, \ldots, x_{i,\ell}) \in \Sigma^\ell$. We define a predicate f_Y over $\Sigma^{n \times \ell}$ for conjunctive equality with wildcards as

$$f_Y(X) = \begin{cases} 1 & \text{if } (x_{i,j} = y_{i,j}) \vee (y_{i,j} = *) \text{ for all } i \in [n], j \in [\ell] \\ 0 & \text{otherwise.} \end{cases}$$

3.3 Construction

Let PRF be a pseudo-random function. Our MC-HVE scheme that supports conjunctive equality queries with wildcards is described as follows.

MC-HVE.Setup$(1^\lambda, n)$**:** Let λ be the security parameter and n be the number of clients. It first obtains a bilinear group $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ of prime order p by running $\mathcal{G}(1^\lambda)$. Let $g, \hat{g}$ be random generators of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. It chooses random PRF keys $\{z_i\}_{i=1}^n$ and a random element $\hat{v} \in \hat{\mathbb{G}}$. It selects random exponents $\{\omega_{i,1}, \omega_{i,2}\}_{i=1}^n$ and calculates $\omega_1 = \sum_{i=1}^n \omega_{i,1}, \omega_2 = \sum_{i=1}^n \omega_{i,2}$. It selects a hash function $H : \{0,1\}^* \to \mathbb{G}$ from the family of hash functions. It outputs a master key $MK = \big(\{z_i\}_{i=1}^n, \hat{v}, \hat{w}_1 = \hat{v}^{\omega_1}, \hat{w}_2 = \hat{v}^{\omega_2}\big)$, encryption keys $\big\{EK_i = (z_i, \omega_{i,1}, \omega_{i,2})\big\}_{i=1}^n$ for all clients, and public parameters

$$PP = \Big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, n, \ell, H\Big).$$

MC-HVE.GenKey(Y, MK, PP)**:** Let $Y = (\vec{y}_1, \ldots, \vec{y}_n)$ where $\vec{y}_i = (y_{i,1}, \ldots, y_{i,\ell}) \in \Sigma_*^\ell$ and $MK = (\{z_i\}_{i=1}^n, \hat{v}, \hat{w}_1, \hat{w}_2)$. It first derives $z_{i,j} = PRF(z_i, j)$ for all i and j. Let S be the set of index tuples (i, j) that are not wildcard positions in Y. It calculates $f_Y = \sum_{(i,j) \in S} PRF(z_{i,j}, y_{i,j})$ by using MK. It chooses random exponents $r_1, r_2, r_3 \in \mathbb{Z}_p$ and outputs a function key by implicitly including the wildcard positions of Y as

$$SK_Y = \Big(K_0 = \hat{v}^{f_Y \cdot r_1} \hat{w}_1^{r_2} \hat{w}_2^{r_3},\ K_1 = \hat{v}^{r_1},\ K_2 = \hat{v}^{r_2},\ K_3 = \hat{v}^{r_3}\Big).$$

MC-HVE.Encrypt$(\vec{x}_i, T, EK_i, PP)$**:** Let $\vec{x}_i = (x_{i,1}, \ldots, x_{i,\ell}) \in \Sigma^\ell$ and $EK_i = (z_i, \omega_{i,1}, \omega_{i,2})$ for a client index i. It first derives $z_{i,j} = PRF(z_i, j)$ for all $j \in [\ell]$. It calculates $f_{i,j} = PRF(z_{i,j}, x_{i,j})$ for all $j \in [\ell]$. It outputs a ciphertext by implicitly including i and T as

$$CT_{i,T} = \Big(\big\{C_{i,j,1} = H(T)^{f_{i,j}}\big\}_{j=1}^\ell,\ C_{i,2} = H(T)^{\omega_{i,1}},\ C_{i,3} = H(T)^{\omega_{i,2}}\Big).$$

MC-HVE.Decrypt$(CT_{1,T}, \ldots, CT_{n,T}, SK_Y, PP)$**:** Let $CT_{i,T} = (\{C_{i,j,1}\}_{j=1}^\ell, C_{i,2}, C_{i,3})$ for a time period T and $SK_Y = (K_0, K_1, K_2, K_3)$ for a list of vectors $Y = (\vec{y}_1, \ldots, \vec{y}_n)$. Let S be the set of index tuples (i, j) that are not wildcard positions in Y. It checks the following equation

$$e(H(T), K_0) \stackrel{?}{=} e(\prod_{(i,j) \in S} C_{i,j,1}, K_1) \cdot e(\prod_{i=1}^n C_{i,2}, K_2) \cdot e(\prod_{i=1}^n C_{i,3}, K_3).$$

If this check succeeds, it outputs 1. Otherwise, it outputs 0.

3.4 Correctness

We show that our MC-HVE scheme satisfies the correctness property. If the messages of client's ciphertexts for the same time period and the attributes of a function key are the same except for the wildcard positions, the following equation is satisfied.

$$\begin{aligned}
e(H(T),K_0) &= e(H(T),\hat{v}^{f_y\cdot r_1}\hat{w}_1^{r_2}\hat{w}_2^{r_3}) \\
&= e\big(H(T),\hat{v}^{\sum_{(i,j)\in S} PRF(z_{i,j},y_{i,j})\cdot r_1}\hat{v}^{\sum_{i=1}^{n}\omega_{i,1}r_2}\hat{v}^{\sum_{i=1}^{n}\omega_{i,2}r_3}\big) \\
&= e\big(H(T)^{\sum_{(i,j)\in S} PRF(z_{i,j},x_{i,j})},\hat{v}^{r_1}\big)\cdot e\big(H(T)^{\sum_{i=1}^{n}\omega_{i,1}},\hat{v}^{r_2}\big)\cdot e\big(H(T)^{\sum_{i=1}^{n}\omega_{i,2}},\hat{v}^{r_3}\big) \\
&= e\big(\prod_{(i,j)\in S} C_{i,j,1},K_1\big)\cdot e\big(\prod_{i=1}^{n} C_{i,2},K_2\big)\cdot e\big(\prod_{i=1}^{n} C_{i,3},K_3\big).
\end{aligned}$$

In contrast, if the messages in ciphertexts and function key are different at least one position, then the above equation is satisfied only with negligible probability since the outputs of PRF will be different when the inputs are different.

3.5 Security Analysis

In this section, we show that our proposed MC-HVE scheme provides the selective single-challenge security. To briefly describe the proof, we first show that our scheme is selectively single-challenge secure when all clients are not corrupted. We next show that this scheme also provides the selective single-challenge security even if statically fixed clients are corrupted.

The basic idea of proving the selective single-challenge security with no corruptions of our MC-HVE scheme is to change all challenge ciphertext elements of the message positions having different challenge message values to random elements. In this case, an attacker cannot win the security game because it cannot obtain any useful information to distinguish X_0^* from X_1^*. In order to implement this idea in the proof, it is essential to devise a method to simulate function keys for any predicate f_Y satisfying $f_Y(X_0^*) = f_Y(X_1^*)$ requested by the attacker without knowing the master key. Specifically, it is needed for a simulator to generate both non-matching function keys such that $f_Y(X_0^*) = f_Y(X_1^*) = 0$ and matching function keys such that $f_Y(X_0^*) = f_Y(X_1^*) = 1$.

In order to handle these non-matching function keys and matching function keys in the security proof, we would like to use the proof methods of the previous HVE schemes [14, 29, 39]. One problem is that the previous HVE schemes are defined in the public-key setting, but our MC-HVE scheme is defined in the private-key setting. To overcome this problem, we observe that changing the pseudo-random function (PRF) to truly random function (TRF) in the security proof allows the simulator to program the TRF outputs of uncorrupted clients as desired by using the lazy sampling technique. Thus, we fix a target challenge message in the selective security game and carefully program the TRF outputs to be similar to the proof setting of the previous HVE schemes. Additionally, we modify the proof methods of previous HVE schemes to be suitable for the multi-client setting. The detailed security proof of our MC-HVE scheme is given as follows.

Theorem 3.1. *The above MC-HVE scheme is selectively single-challenge IND-secure with no corruptions in the random oracle model if the SXDH and A3DH assumptions hold and the PRF is secure.*

Proof. Suppose there exists an adversary that breaks the selective single-challenge IND-security game with no corruptions. The adversary initially submits two challenge messages $X_0^* = (\vec{x}_{0,1}^*,\ldots,\vec{x}_{0,n}^*)$, $X_1^* = (\vec{x}_{1,1}^*,\ldots,\vec{x}_{1,n}^*)$, and challenge time T^* where $\vec{x}_{\mu,i} = (x_{\mu,i,1}^*,\ldots,x_{\mu,i,\ell}^*) \in \Sigma^\ell$. Let E be the set of index tuple

(i,j) such that $x^*_{0,i,j} = x^*_{1,i,j}$ and $\overline{E}$ be the set of index tuple (i,j) such that $x^*_{0,i,j} \neq x^*_{1,i,j}$. To argue that the adversary cannot win this security game, we define a sequence of hybrid games which are defined as follows:

Game $\mathbf{G}_0$. This game $\mathbf{G}_0$ denotes the selective single-challenge game which is defined in Section 2.3 where $\overline{I} = \emptyset$ for no corruptions.

Game $\mathbf{G}_1$. This game $\mathbf{G}_1$ is almost identical to the game $\mathbf{G}_0$ except the generation of function keys. Let S be the set of index tuple (i,j) that are not wildcard positions in function attributes $Y = (\vec{y}_1, \ldots, \vec{y}_n)$ where $\vec{y}_i = (y_{i,1}, \ldots, y_{i,\ell})$. Any function key query requested by the adversary should be one of the following types:

- A function key query is **Type-1** if it satisfies $f_Y(X^*_0) = f_Y(X^*_1) = 1$. In this query, we have $y_{i,j} = x^*_{0,i,j} = x^*_{1,i,j}$ for all index tuple $(i,j) \in S$ since $S \cap \overline{E} = \emptyset$.
- A function key query is **Type-2** if it satisfies $f_Y(X^*_0) = f_Y(X^*_1) = 0$. In this query, we have $y_{i,j} \neq x^*_{0,i,j}$ and $y_{i,j} \neq x^*_{1,i,j}$ for some index tuple $(i,j) \in S$.

In this game, a simulator initially chooses a random value $\pi \in \mathbb{Z}_p$. To handle a type-1 function key query, the simulator chooses two random exponents r_1, r_3 and sets $r_2 = \pi r_3$ in a correlated way by using the fixed π. To handle a type-2 function key query, the simulator chooses three independent random exponents r_1, r_2, and r_3.

Game $\mathbf{G}_2$. In this game $\mathbf{G}_2$, we replace the pseudo-random function $PRF(z_{i,j}, x)$ with the truly random function $TRF_{i,j}(x)$ for all i, j. This change can be easily done by the security of PRF.

Game $\mathbf{G}_3$. This game $\mathbf{G}_3$ is similar to the game $\mathbf{G}_2$ except the generation of the challenge ciphertext. In this game, the simulator slightly changes the generation of challenge ciphertext elements as $C_{i,2} = H(T^*)^{\omega_{i,1}} g^{\rho_i}$ and $C_{i,3} = H(T^*)^{\omega_{i,2}} g^{\phi_i}$ for all client index $i \in [n]$ with random exponents ρ_i and ϕ_i that satisfy $\sum_{i=1}^n \rho_i \cdot \pi + \sum_{i=1}^n \phi_i = 0$. Note that even the adversary that has a type-1 function key cannot distinguish the changed challenge ciphertext elements since it gets $\sum_{i=1}^n \rho_i \cdot \pi + \sum_{i=1}^n \phi_i = 0$ during the decryption process.

Game $\mathbf{G}_4$. This final game $\mathbf{G}_4$ differs from the game $\mathbf{G}_3$ in that for all index tuple $(i,j) \in \overline{E}$, the challenge ciphertext element $C_{i,j,1}$ is generated as a random element. In this game, the challenge ciphertext gives no information about the challenge message X^*_μ. Therefore, the advantage of the adversary in this game is zero.

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, 3.3, 3.4, and 3.5, we obtain the following result

$$\begin{aligned}\mathbf{Adv}^{SE\text{-}1\text{-}IND}_{\mathcal{A}}(\lambda) &\leq \left|\Pr[S^{\mathbf{G}_0}_{\mathcal{A}}] - \Pr[S^{\mathbf{G}_4}_{\mathcal{A}}]\right| + \Pr[S^{\mathbf{G}_4}_{\mathcal{A}}] \leq \sum_{i=1}^{4} \left|\Pr[S^{\mathbf{G}_{i-1}}_{\mathcal{A}}] - \Pr[S^{\mathbf{G}_i}_{\mathcal{A}}]\right| + \Pr[S^{\mathbf{G}_4}_{\mathcal{A}}] \\ &\leq \mathbf{Adv}^{SXDH}_{\mathcal{B}}(\lambda) + n\ell \cdot \mathbf{Adv}^{PRF}_{\mathcal{B}}(\lambda) + (1 + n\ell) \cdot \mathbf{Adv}^{A3DH}_{\mathcal{B}}(\lambda)\end{aligned}$$

where $n\ell$ is the size of the challenge message. This completes our proof. □

Lemma 3.2. *If the SXDH 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 new complexity assumption by extending the DDH assumption. The multi-DDH assumption is that if the challenge tuple $D = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, \hat{g}^a, \hat{g}^{b_1}, \ldots, \hat{g}^{b_n}\big)$ and $\vec{Z} = (Z_1, \ldots, Z_n)$ are given, no PPT algorithm $\mathcal{A}$ can distinguish $\vec{Z} = \vec{Z}_0 = (\hat{g}^{ab_1}, \ldots, \hat{g}^{ab_n})$ from $\vec{Z} = \vec{Z}_1 = (\hat{g}^{c_1}, \ldots, \hat{g}^{c_n})$ with more than a negligible advantage where $a, b_1, \ldots, b_n, c_1, \ldots, c_n$ are randomly chosen in $\mathbb{Z}_p$. The multi-DDH assumption holds if the DDH assumption holds from the random self-reducibility of the DDH assumption [17].

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the multi-DDH assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, \hat{g}^a, \hat{g}^{b_1}, \ldots, \hat{g}^{b_q})$ and $\vec{Z} = (Z_1, \ldots, Z_q)$ where $\vec{Z} = (\hat{g}^{ab_1}, \ldots, \hat{g}^{ab_q})$ or $\vec{Z} = (\hat{g}^{c_1}, \ldots, \hat{g}^{c_q})$ where q is the maximum number of type-1 queries. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ submits challenge message lists $X_0^* = (\vec{x}_{0,1}, \ldots, \vec{x}_{0,n})$, $X_1^* = (\vec{x}_{1,1}, \ldots, \vec{x}_{1,n}) \in \Sigma^{n\ell}$, and a challenge time period T^*. $\mathcal{B}$ then flips a random coin μ internally to fix X_μ^* as the target message list.

Setup: $\mathcal{B}$ first chooses random PRF keys $\{z_i\}_{i=1}^n$. It also selects random exponents $v', \{\omega_{i,1}, \omega_{i,2}\}_{i=1}^n \in \mathbb{Z}_p$, calculates $\omega_1 = \sum_{i=1}^n \omega_{i,1}, \omega_2 = \sum_{i=1}^n \omega_{i,2}$, and then implicitly sets $\hat{v} = \hat{g}^{v'}, \hat{w}_1 = \hat{g}^{v'\omega_1}, \hat{w}_2 = \hat{g}^{v'\omega_2}$. Next, it sets MK, $\{EK_i\}_{i=1}^n$, and PP by using these selected elements.

Challenge: $\mathcal{B}$ creates challenge ciphertexts $CT_{1,T^*}, \ldots, CT_{n,T^*}$ for X_μ^* by running $\mathbf{Encrypt}(\vec{x}_{\mu,i}^*, T^*, EK_i, PP)$ for all client index $i \in [n]$ by using EK_i.

Query: $\mathcal{A}$ adaptively requests hash, function key, and ciphertext queries. $\mathcal{B}$ handles these queries as follows:

If this is a hash query for a time period T, then $\mathcal{B}$ proceeds as follows: If T was queried before, then it retrieves $(T, h, -)$ from a hash list and returns h. Otherwise, it selects a random element $h \in \mathbb{G}$, stores $(T, h, -)$ to a hash list, and then returns h.

If this is a function key query for a list of vectors $Y = (\vec{y}_1, \ldots, \vec{y}_n)$, then $\mathcal{B}$ generates a function key depending on the type of function key queries as follows:

- **Case** Type-1: Let k be the index of type-1 function key queries. It calculates $f_y = \sum_{(i,j)\in S} PRF(z_{i,j}, y_{i,j})$ by using MK. Next, it chooses a random exponent $r_1 \in \mathbb{Z}_p$ and creates a function key depending on the index k as

$$K_0 = \hat{v}^{f_y r_1} (Z_k)^{v'\omega_1} (\hat{g}^{b_k})^{v'\omega_2},\ K_1 = \hat{v}^{r_1},\ K_2 = (Z_k)^{v'},\ K_3 = (\hat{g}^{b_k})^{v'}.$$

  If $Z_k = \hat{g}^{ab_k}$, then $\mathcal{B}$ plays the game $\mathbf{G}_1$ since this function key is correctly distributed with setting $r_2 = ab_k, r_3 = b_k, \pi = a$ as

$$K_0 = \hat{v}^{f_y r_1} (\hat{g}^{ab_k})^{v'\omega_1} (\hat{g}^{b_k})^{v'\omega_2} = \hat{v}^{f_y r_1} \hat{w}_1^{ab_k} \hat{w}_2^{b_k} = \hat{v}^{f_y r_1} \hat{w}_1^{\pi r_3} \hat{w}_2^{r_3},$$
$$K_1 = \hat{v}^{r_1},\ K_2 = (\hat{g}^{ab_k})^{v'} = \hat{v}^{ab_k} = \hat{v}^{\pi r_3},\ K_3 = (\hat{g}^{b_k})^{v'} = \hat{v}^{b_k} = \hat{v}^{r_3}.$$

  Otherwise ($Z_k = \hat{g}^{c_k}$), $\mathcal{B}$ plays the game $\mathbf{G}_0$ since it implicitly sets $r_2 = c_k, r_3 = b_k$.

- **Case** Type-2: It simply creates a function key by running the **GenKey** algorithm since it knows MK.

If this is a ciphertext query for a client index i, a message vector $\vec{x}_i$, and a time period T, then $\mathcal{B}$ generate a ciphertext by running the **Encrypt** algorithm since it knows EK_i.

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

Lemma 3.3. *If the PRF scheme is secure, then no polynomial-time adversary can distinguish between $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ with a non-negligible advantage.*

Proof. This proof is relatively straightforward from the security of PRF. That is, we can use additional hybrid games that change a PRF to a truly random function one by one. Note that there are at most $n\ell$ number of $z_{i,j}$ in the security proof. We omit the details of this proof. □

Lemma 3.4. *If the A3DH assumption holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_2$ and $\mathbf{G}_3$ with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_2$ and $\mathbf{G}_3$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the A3DH assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e),g,g^a,g^b,g^{ab},g^{abc},\hat{g},\hat{g}^a,\hat{g}^b)$ and Z where $Z = g^c$ or $Z = R$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows.

Init: $\mathcal{A}$ submits challenge message lists $X_0^* = (\vec{x}_{0,1}^*,\ldots,\vec{x}_{0,n}^*)$, $X_1^* = (\vec{x}_{1,1}^*,\ldots,\vec{x}_{1,n}^*) \in \Sigma^{n\ell}$, and a challenge time period T^* where $\vec{x}_{\mu,i}^* = (x_{\mu,i,1}^*,\ldots,x_{\mu,i,\ell}^*)$. $\mathcal{B}$ flips a random coin $\mu \in \{0,1\}$ to fix X_μ^* as the target challenge message list.

Setup: $\mathcal{B}$ proceeds the following steps:

1. It first selects random exponents $\{\omega'_{i,1},\omega'_{i,2},\rho'_i,\phi'_i\}_{i=1}^n \in \mathbb{Z}_p$ and defines $\{\omega_{i,1},\omega_{i,2}\}_{i=1}^n$ as follows:

$$\omega_{i,1} := \omega'_{i,1} + (1/ab)\rho'_i,\ \ \omega_{i,2} := \omega'_{i,2} + (1/ab)\phi'_i.$$

   Next, it calculates $\omega'_1 = \sum_{i=1}^n \omega'_{i,1}, \omega'_2 = \sum_{i=1}^n \omega'_{i,2}, \rho' = \sum_{i=1}^n \rho'_i, \phi' = \sum_{i=1}^n \phi'_i$ and implicitly sets $\hat{v} = \hat{g}^{ab}, \hat{w}_1 = (\hat{g}^{ab})^{\omega'_1}\hat{g}^{\rho'}, \hat{w}_2 = (\hat{g}^{ab})^{\omega'_2}\hat{g}^{\phi'}$. It also sets $\pi = -\phi'/\rho'$ for type-1 function key queries.

2. We now define the value of TRF. Let $x_{i,j}$ be a message for a client i and $x_{\mu,i,j}^*$ be the challenge message for the same client index i. If $x_{i,j} = x_{\mu,i,j}^*$, then it defines $TRF_{i,j}(x_{i,j}) := f'_{i,j}$ by selecting a fixed random $f'_{i,j}$ for $x_{i,j}$. Otherwise ($x_{i,j} \neq x_{\mu,i,j}^*$), it defines $TRF_{i,j}(x_{i,j}) := (1+1/a)f'_{i,j}$ by selecting a fixed random $f'_{i,j}$ for $x_{i,j}$. That is, it implicitly defines

$$\hat{v}^{TRF_{i,j}(x_{i,j})} := (\hat{g}^{ab})^{f'_{i,j}} \text{ if } x_{i,j} = x_{\mu,i,j}^*,\ \ \hat{v}^{TRF_{i,j}(x_{i,j})} := (\hat{g}^{ab}\hat{g}^{b})^{f'_{i,j}} \text{ if } x_{i,j} \neq x_{\mu,i,j}^*.$$

3. For the target message list $X_\mu^* = (\vec{x}_{\mu,1}^*,\ldots,\vec{x}_{\mu,n}^*)$ where $\vec{x}_{\mu,i}^* = (x_{\mu,i,1}^*,\ldots,x_{\mu,i,\ell}^*)$, it selects a random $f'_{i,j}$ and defines $TRF_{i,j}(x_{\mu,i,j}^*) := f'_{i,j}$ for all $i \in [n], j \in [\ell]$.

4. It initializes a hash table H-list and publishes $PP = ((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e),g,\hat{g},n,\ell,H)$.

Challenge: $\mathcal{B}$ retrieves $\{f'_{i,j}\}$ which are defined as $TRF_{i,j}(x_{\mu,i,j}^*) := f'_{i,j}$ for the challenge messages X_μ^*. Next, it implicitly sets $H(T^*) = g^{abc}$ and creates challenge ciphertexts for the time period T^* as

$$\left(\left\{C_{i,j,1} = (g^{abc})^{f'_{i,j}}\right\}_{j=1}^{\ell},\ C_{i,2} = (g^{abc})^{\omega'_{i,1}}Z^{\rho'_i},\ C_{i,3} = (g^{abc})^{\omega'_{i,2}}Z^{\phi'_i}\right) \text{ for all } i \in [n].$$

If $Z = g^c$, then it plays the game $\mathbf{G}_2$ by the following equations

$$C_{i,2} = (g^{abc})^{\omega'_{i,1}+(1/ab)\rho'_i} = (g^{abc})^{\omega'_{i,1}}(g^c)^{\rho'_i},\ C_{i,3} = (g^{abc})^{\omega'_{i,2}+(1/ab)\phi'_i} = (g^{abc})^{\omega'_{i,2}}(g^c)^{\phi'_i}.$$

Otherwise ($Z = R = g^d$), it implicitly sets $\rho_i = (-c+d)\rho'_i$, $\phi_i = (-c+d)\phi'_i$ and plays $\mathbf{G}_3$ by the following equations

$$\begin{aligned} C_{i,2} &= H(T^*)^{\omega_{i,1}}g^{\rho_i} = (g^{abc})^{\omega'_{i,1}+(1/ab)\rho'_i}g^{(-c+d)\rho'_i} = (g^{abc})^{\omega'_{i,1}}(g^d)^{\rho'_i}, \\ C_{i,3} &= H(T^*)^{\omega_{i,2}}g^{\phi_i} = (g^{abc})^{\omega'_{i,2}+(1/ab)\phi'_i}g^{(-c+d)\phi'_i} = (g^{abc})^{\omega'_{i,2}}(g^d)^{\phi'_i}. \end{aligned}$$

Note that we have $\sum_{i=1}^{n} \rho_i \cdot \pi + \sum_{i=1}^{n} \phi_i = 0$ since $\pi = -\phi'/\rho' = -\sum_{i=1}^{n} \phi'_i / \sum_{i=1}^{n} \rho'_i$.

Query: $\mathcal{A}$ adaptively requests hash, function key, and ciphertext queries. $\mathcal{B}$ handles these queries as follows. If this is a hash query for a time period T, then $\mathcal{B}$ proceeds as follows: If T was queried before, then it retrieves $(T, h, -)$ from a hash list and returns h. Otherwise, it performs:

- **Case** $T = T^*$: It sets $H(T^*) = g^{abc}$ and stores $(T^*, H(T^*), -)$ to H-list. It then returns g^{abc}.
- **Case** $T \neq T^*$: It selects a random exponent $h' \in \mathbb{Z}_p$ and stores $(T, H(T) = (g^{ab})^{h'}, h')$ to H-list. It then returns $(g^{ab})^{h'}$.

If this is a function key query for a list of vectors $Y = (\vec{y}_1, \ldots, \vec{y}_n)$ where $\vec{y}_i = (y_{i,1}, \ldots, y_{i,\ell})$, then $\mathcal{B}$ generates a function key as follows: It first defines two index sets A and $\overline{A}$ for the list of vectors Y as $A = \{(i,j) : (y_{i,j} \neq *) \wedge (y_{i,j} = x^*_{\mu,i,j})\}$ and $\overline{A} = \{(i,j) : (y_{i,j} \neq *) \wedge (y_{i,j} \neq x^*_{\mu,i,j})\}$ where $i \in [n]$ and $j \in [\ell]$. It then creates a function key depending on the type of this function key as follows:

- **Case** Type-1: It implicitly calculates

$$f_Y = \sum_{(i,j)\in S} TRF_{i,j}(x_{i,j}) = \sum_{(i,j)\in A} f'_{i,j} = f'_A$$

by retrieving $f'_{i,j}$ chosen at the setup since $\overline{A} = \emptyset$ where $f'_A = \sum_{(i,j)\in A} f'_{i,j}$. It chooses random exponents $r'_1, r'_3 \in \mathbb{Z}_p$ and implicitly defines the randomness as

$$r_1 := r'_1/ab,\ r_2 := \phi' r'_3/b,\ r_3 := -\rho' r'_3/b$$

where r_2 and r_3 are correlated as $r_2 = \pi r_3$ with the fixed $\pi = -\phi'/\rho'$. Next, it creates a function key by using the randomness as

$$K_0 = \hat{g}^{f'_A r'_1} (\hat{g}^a)^{\omega'_1 \phi' r'_3 - \omega'_2 \rho' r'_3},\ K_1 = \hat{g}^{r'_1},\ K_2 = (\hat{g}^a)^{\phi' r'_3},\ K_3 = (\hat{g}^a)^{-\rho' r'_3}.$$

This type-1 function key is correctly distributed by the following equation

$$K_0 = (\hat{g}^{ab})^{f'_A r'_1/ab} \left(\hat{g}^{ab\omega'_1 + \rho'}\right)^{\phi' r'_3/b} \left(\hat{g}^{ab\omega'_2 + \phi'}\right)^{-\rho' r'_3/b} = \hat{g}^{f'_A r'_1} (\hat{g}^a)^{\omega'_1 \phi' r'_3 - \omega'_2 \rho' r'_3}.$$

- **Case** Type-2: It implicitly calculates

$$\begin{aligned} f_Y &= \sum_{(i,j)\in S} TRF_{i,j}(x_{i,j}) = \sum_{(i,j)\in A} f'_{i,j} + \sum_{(i,j)\in \overline{A}} (1 + 1/a) f'_{i,j} \\ &= \sum_{(i,j)\in S} f'_{i,j} + \sum_{(i,j)\in \overline{A}} (1/a) f'_{i,j} = f'_S + (1/a) f'_{\overline{A}} \end{aligned}$$

by retrieving $f'_{i,j}$ for the set A and selecting a fixed random $f'_{i,j}$ for the set $\overline{A}$ since $\overline{A} \neq \emptyset$ where $f'_S = \sum_{(i,j)\in S} f'_{i,j}$ and $f'_{\overline{A}} = \sum_{(i,j)\in \overline{A}} f'_{i,j}$. It chooses random exponents $r'_1, r'_2, r'_3 \in \mathbb{Z}_p$ and implicitly defines the randomness as

$$r_1 := r'_1/b + \phi' r'_3/ab,\ r_2 := \phi' r'_2/b,\ r_3 := -f'_{\overline{A}} r'_3/a - \rho' r'_2/b$$

where r_1, r_2, and r_3 are independent random values since $f'_{\overline{A}} \neq 0$ except with negligible probability. Next, it creates a function key by using the defined randomness as

$$\begin{aligned} K_0 &= (\hat{g}^a)^{f'_S r'_1} \hat{g}^{f'_S \phi' r'_3} \hat{g}^{f'_{\overline{A}} r'_1} (\hat{g}^a)^{\omega'_1 \phi' r'_2} (\hat{g}^b)^{-\omega'_2 f'_{\overline{A}} r'_3} (\hat{g}^a)^{-\omega'_2 \rho' r'_2}, \\ K_1 &= (\hat{g}^a)^{r'_1} \hat{g}^{\phi' r'_3},\ K_2 = (\hat{g}^a)^{\phi' r'_2},\ K_3 = (\hat{g}^b)^{-f'_{\overline{A}} r'_3} (\hat{g}^a)^{-\rho' r'_2}. \end{aligned}$$

This type-2 function key is correctly distributed by the following equation

$$\begin{aligned}
K_0 =& (\hat{g}^{ab})^{f_Y\cdot(r'_1/b+\phi' r'_3/ab)} \left(\hat{g}^{ab\omega'_1+\rho'}\right)^{\phi' r'_2/b} \left(\hat{g}^{ab\omega'_2+\phi'}\right)^{-f'_A r'_3/a-\rho' r'_2/b} \\
=& \left(\hat{g}^{ab f'_S}\hat{g}^{b f'_A}\right)^{r'_1/b+\phi' r'_3/ab} \left(\hat{g}^{ab\omega'_1}\right)^{\phi' r'_2/b} \left(\hat{g}^{ab\omega'_2}\right)^{-f'_A r'_3/a-\rho' r'_2/b} \left(\hat{g}^{\phi'}\right)^{-f'_A r'_3/a} \\
=& \left(\hat{g}^{ab f'_S}\right)^{r'_1/b+\phi' r'_3/ab} \left(\hat{g}^{b f'_A}\right)^{r'_1/b} \left(\hat{g}^{ab\omega'_1}\right)^{\phi' r'_2/b} \left(\hat{g}^{ab\omega'_2}\right)^{-f'_A r'_3/a-\rho' r'_2/b} \\
=& (\hat{g}^{a})^{f'_S r'_1} \hat{g}^{f'_S\phi' r'_3} \hat{g}^{f'_A r'_1} (\hat{g}^{a})^{\omega'_1\phi' r'_2} (\hat{g}^{b})^{-\omega'_2 f'_A r'_3} (\hat{g}^{a})^{-\omega'_2\rho' r'_2}.
\end{aligned}$$

If this is a ciphertext query for a client index i, a message vector $\vec{x}_i = (x_{i,1},\ldots,x_{i,\ell})$, and a time period $T \neq T^*$, then $\mathcal{B}$ retrieves $(T, H(T) = (g^{ab})^{h'}, h')$ from the H-list and generates a ciphertext as follows: For each $j \in [\ell]$, it performs:

- If $x_{i,j} = x^*_{\mu,i,j}$, it creates $C_{i,j,1} = (g^{ab})^{h' f'_{i,j}}$ since $TRF_{i,j}(x_{i,j}) := f'_{i,j}$. Otherwise ($x_{i,j} \neq x^*_{\mu,i,j}$), it creates $C_{i,j,1} = (g^{ab}g^{b})^{h' f'_{i,j}}$ since $TRF_{i,j}(x_{i,j}) := (1+1/a) f'_{i,j}$.

Next, it generates a ciphertext as $\left(\left\{C_{i,j,1}\right\}_{j=1}^{\ell}, C_{i,2} = (g^{ab})^{h'\omega'_{i,1}} g^{h'\rho'_i}, C_{i,3} = (g^{ab})^{h'\omega'_{i,2}} g^{h'\phi'_i}\right)$.

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

To complete this proof, we show that $\{\rho_i = (-c+d)\rho'_i, \phi_i = (-c+d)\phi'_i\}_{i=1}^n$ are random that satisfy $\sum_{i=1}^n \rho_i\pi + \sum_{i=1}^n \phi_i = 0$. First, we can see that $\{\rho'_i, \phi'_i\}_{i=1}^n$ are well hidden by the setting $\omega_{i,1} = \omega'_{i,1} + (1/ab)\rho'_i, \omega_{i,2} = \omega'_{i,2} + (1/ab)\phi'_i$ since $\omega'_{i,1}, \omega'_{i,2}$ are completely random. Next, since $\pi = -\sum_{i=1}^n \phi'_i / \sum_{i=1}^n \rho'_i$, we can easily see that ρ_i, ϕ_i satisfy $\sum_{i=1}^n \rho_i\pi + \sum_{i=1}^n \phi_i = \sum_{i=1}^n (-c+d)\rho'_i\pi + \sum_{i=1}^n (-c+d)\phi'_i = 0$. □

Lemma 3.5. *If the A3DH assumption holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_3$ and $\mathbf{G}_4$ with a non-negligible advantage.*

Proof. We define a sequence of hybrid games $\mathbf{H}_{3,(1,0)}, \mathbf{H}_{3,(1,1)}, \ldots, \mathbf{H}_{3,(1,\ell)}, \mathbf{H}_{3,(2,1)}, \ldots, \mathbf{H}_{3,(i_\delta,j_\delta)}, \ldots, \mathbf{H}_{3,(n,\ell)}$ for indexes $i_\delta \in [n]$ and $j_\delta \in [\ell]$ where $\mathbf{H}_{3,(1,0)} = \mathbf{G}_3$. For notational simplicity, we also define $\mathbf{H}_{3,(i_\delta-1,\ell)} = \mathbf{H}_{3,(i_\delta,0)}$. Recall that two sets $E, \overline{E}$ were defined as $E = \{(i,j) : x^*_{0,i,j} = x^*_{1,i,j}\}$ and $\overline{E} = \{(i,j) : x^*_{0,i,j} \neq x^*_{1,i,j}\}$ where $x^*_{0,i,j} \in X^*_0$ and $x^*_{1,i,j} \in X^*_1$. For notational simplicity, we define a comparison $(i,j) \leq (i_\delta, j_\delta)$ as $(i < i_\delta)$ or $(i = i_\delta \wedge j \leq j_\delta)$. The game $\mathbf{H}_{3,(i_\delta,j_\delta)}$ is defined as follows:

Game $\mathbf{H}_{3,(i_\delta,j_\delta)}$ In this game $\mathbf{H}_{3,(i_\delta,j_\delta)}$, we slightly change the generation of challenge ciphertext elements in $\overline{E}$. The simulator generates challenge ciphertext elements $\{C_{i,j,1}\}$ as follows:

- If $(i,j) \in E$, then it creates $C_{i,j,1}$ normally.
- If $(i,j) \in \overline{E}$ and $(i,j) \leq (i_\delta, j_\delta)$, then it creates $C_{i,j,1}$ as a random element in $\mathbb{G}$.
- If $(i,j) \in \overline{E}$ and $(i,j) > (i_\delta, j_\delta)$, then it creates $C_{i,j,1}$ normally.

The simulator generates challenge ciphertext elements $C_{i,2}, C_{i,3}$ as the same way as $\mathbf{G}_3$. It is obvious that $\mathbf{H}_{3,(n,\ell)} = \mathbf{G}_4$.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H}_{3,(i_\delta,j_\delta-1)}$ and $\mathbf{H}_{3,(i_\delta,j_\delta)}$ with a non-negligible advantage. Without loss of generality, we assume that $(i_\delta, j_\delta) \in \overline{E}$ since $\mathbf{H}_{3,(i_\delta,j_\delta-1)} = \mathbf{H}_{3,(i_\delta,j_\delta)}$ if $(i_\delta, j_\delta) \in E$. A simulator $\mathcal{B}$ that solves the A3DH assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e), g, g^a, g^b, g^{ab}, g^{abc}, \hat{g}^a, \hat{g}^b)$ and Z where $Z = g^c$ or $Z = R$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows.

Init: $\mathcal{A}$ submits challenge message lists $X_0^* = (\vec{x}_{0,1}^*, \ldots, \vec{x}_{0,n}^*)$, $X_1^* = (\vec{x}_{1,1}^*, \ldots, \vec{x}_{1,n}^*) \in \Sigma^{n\ell}$, and a challenge time period T^* where $\vec{x}_{\mu,i}^* = (x_{\mu,i,1}^*, \ldots, x_{\mu,i,\ell}^*)$ for $\mu \in \{0,1\}$ and $i \in [n]$. $\mathcal{B}$ flips a random coin $\mu \in \{0,1\}$ to fix X_μ^* as the target message list.

Setup: $\mathcal{B}$ proceeds the following steps:

1. It first selects random exponents $\{\omega'_{i,1}, \omega'_{i,2}, \rho'_i, \phi'_i\}_{i=1}^n \in \mathbb{Z}_p$ and defines $\{\omega_{i,1}, \omega_{i,2}\}_{i=1}^n$ as follows:

$$\omega_{i,1} := \omega'_{i,1} + (1/ab)\rho'_i, \;\; \omega_{i,2} := \omega'_{i,2} + (1/ab)\phi'_i.$$

   Next, it calculates $\omega'_1 = \sum_{i=1}^n \omega'_{i,1}, \omega'_2 = \sum_{i=1}^n \omega'_{i,2}, \rho' = \sum_{i=1}^n \rho'_i, \phi' = \sum_{i=1}^n \phi'_i$ and implicitly sets $\hat{v} = \hat{g}^{ab}, \hat{w}_1 = \hat{g}^{ab\omega'_1}\hat{g}^{\rho'}, \hat{w}_2 = \hat{g}^{ab\omega'_2}\hat{g}^{\phi'}$. It also sets $\pi = -\phi'/\rho'$ for type-1 function key queries.

2. We define the value of TRF. Let $x_{i,j}$ be a message for a client i and $x_{\mu,i,j}^*$ be the challenge message for the same client index i.

   - **Case** $(i,j) \neq (i_\delta, j_\delta)$: If $x_{i,j} = x_{\mu,i,j}^*$, then it defines $TRF_{i,j}(x_{i,j}) := f'_{i,j}$ by selecting a fixed random $f'_{i,j}$ for $x_{i,j}$. Otherwise ($x_{i,j} \neq x_{\mu,i,j}^*$), it defines $TRF_{i,j}(x_{i,j}) := (1+1/a)f'_{i,j}$ by selecting a fixed random $f'_{i,j}$ for $x_{i,j}$. That is, it implicitly defines

$$\hat{v}^{TRF_{i,j}(x_{i,j})} := \left(\hat{g}^{ab}\right)^{f'_{i,j}} \text{ if } x_{i,j} = x_{\mu,i,j}^*, \;\; \hat{v}^{TRF_{i,j}(x_{i,j})} := \left(\hat{g}^{ab}\hat{g}^{b}\right)^{f'_{i,j}} \text{ if } x_{i,j} \neq x_{\mu,i,j}^*.$$

   - **Case** $(i,j) = (i_\delta, j_\delta)$: If $x_{i,j} = x_{\mu,i,j}^*$, then it defines $TRF_{i,j}(x_{i,j}) := (1/ab)f'_{i,j}$ by selecting a fixed random $f'_{i,j}$ for $x_{i,j}$. Otherwise ($x_{i,j} \neq x_{\mu,i,j}^*$), it defines $TRF_{i,j}(x_{i,j}) := (1/a+1/ab)f'_{i,j}$ by selecting a fixed random $f'_{i,j}$ for $x_{i,j}$. That is, it implicitly defines

$$\hat{v}^{TRF_{i,j}(x_{i,j})} := \left(\hat{g}\right)^{f'_{i,j}} \text{ if } x_{i,j} = x_{\mu,i,j}^*, \;\; \hat{v}^{TRF_{i,j}(x_{i,j})} := \left(\hat{g}^{b}\hat{g}\right)^{f'_{i,j}} \text{ if } x_{i,j} \neq x_{\mu,i,j}^*.$$

3. For the target message list $X_\mu^* = (\vec{x}_{\mu,1}^*, \ldots, \vec{x}_{\mu,n}^*)$ where $\vec{x}_{\mu,i}^* = (x_{\mu,i,1}^*, \ldots, x_{\mu,i,\ell}^*)$, it selects a random $f'_{i,j}$ for all $i \in [n], j \in [\ell]$, and then defines $TRF_{i,j}(x_{\mu,i,j}^*) := (1/ab)f'_{i,j}$ if $(i,j) = (i_\delta, j_\delta)$ and $TRF_{i,j}(x_{\mu,i,j}^*) := f'_{i,j}$ if $(i,j) \neq (i_\delta, j_\delta)$.

4. It initializes a hash table H-list and publishes $PP = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, n, \ell, H)$.

Challenge: $\mathcal{B}$ sets $H(T^*) = g^{abc}$ and outputs a challenge ciphertext as follows: For each (i,j) where $i \in [n]$ and $j \in [\ell]$, it performs:

- **Case** $(i,j) < (i_\delta, j_\delta)$: If $(i,j) \in \overline{E}$, then it chooses a random element $P_{i,j,1} \in \mathbb{G}$ and sets $C_{i,j,1} = P_{i,j,1}$. Otherwise, it creates $C_{i,j,1} = (g^{abc})^{f'_{i,j}}$ since $TRF_{i,j}(x_{\mu,i,j}^*) := f'_{i,j}$.

- **Case** $(i,j) = (i_\delta, j_\delta)$: It creates $C_{i,j,1} = Z^{f'_{i,j}}$ since $(i_\delta, j_\delta) \in \overline{E}$ and $TRF_{i,j}(x_{\mu,i,j}^*) := (1/ab)f'_{i,j}$.

- **Case** $(i,j) > (i_\delta, j_\delta)$: It creates $C_{i,j,1} = (g^{abc})^{f'_{i,j}}$ since $TRF_{i,j}(x_{\mu,i,j}^*) := f'_{i,j}$.

Next, it chooses a random element $P = g^s \in \mathbb{G}$ and generates the challenge ciphertext by implicitly setting $\rho_i = (-c+s)\rho'_i, \phi_i = (-c+s)\phi'_i$ as

$$\left(\left\{C_{i,j,1}\right\}_{j=1}^{\ell}, C_{i,2} = (g^{abc})^{\omega'_{i,1}} P^{\rho'_i}, C_{i,3} = (g^{abc})^{\omega'_{i,2}} P^{\phi'_i}\right) \text{ for all } i \in [n].$$

If Z is a valid A3DH tuple, then $\mathcal{B}$ plays $\mathbf{H}_{3,(i_\delta, j_\delta - 1)}$. Otherwise, it plays $\mathbf{H}_{3,(i_\delta, j_\delta)}$.

Query: $\mathcal{A}$ adaptively requests hash, function key, and ciphertext queries. $\mathcal{B}$ handles these queries as follows. If this is a hash query for a time period T, then $\mathcal{B}$ proceeds as follows: If T was queried before, then it retrieves $(T, h, -)$ from a hash list and returns h. Otherwise, it performs:

- **Case** $T = T^*$: It sets $H(T^*) = g^{abc}$ and stores $(T^*, H(T^*), -)$ to H-list. It then returns g^{abc}.
- **Case** $T \neq T^*$: It selects a random exponent $h' \in \mathbb{Z}_p$ and stores $(T, H(T) = (g^{ab})^{h'}, h')$ to H-list. It then returns $(g^{ab})^{h'}$.

If this is a function key query for a list of vectors $Y = (\vec{y}_1, \ldots, \vec{y}_n)$ where $\vec{y}_i = (y_{i,1}, \ldots, y_{i,\ell})$, then $\mathcal{B}$ generates a function key as follows: It first defines two index sets A and $\overline{A}$ for the list of vectors Y as $A = \{(i,j) : (y_{i,j} \neq *) \wedge (y_{i,j} = x^*_{\mu,i,j})\}$ and $\overline{A} = \{(i,j) : (y_{i,j} \neq *) \wedge (y_{i,j} \neq x^*_{\mu,i,j})\}$ where $i \in [n]$ and $j \in [\ell]$. It then creates a function key depending on the type of this function key as follows:

- **Case** Type-1: In this case, we have $(i_\delta, j_\delta) \notin S$ since $(i_\delta, j_\delta) \in \overline{E}$ and $S \cap \overline{E} = \emptyset$. It implicitly calculates

$$f_Y = \sum_{(i,j)\in S} TRF_{i,j}(x_{i,j}) = \sum_{(i,j)\in A} f'_{i,j} = f'_A$$

  by retrieving $f'_{i,j}$ chosen at the setup since $\overline{A} = \emptyset$ where $f'_A = \sum_{(i,j)\in A} f'_{i,j}$. It chooses random exponents $r'_1, r'_3 \in \mathbb{Z}_n$ and implicitly defines the randomness as

$$r_1 := r'_1/ab,\ r_2 := \phi' r'_3/b,\ r_3 := -\rho' r'_3/b.$$

  Next, it creates a function key by using the defined randomness as

$$K_0 = \hat{g}^{f_Y \cdot r'_1} (\hat{g}^a)^{\omega'_1 \phi' r'_3 - \omega'_2 \rho' r'_3},\ K_1 = \hat{g}^{r'_1},\ K_2 = (\hat{g}^a)^{\phi' r'_3},\ K_3 = (\hat{g}^a)^{-\rho' r'_3}.$$

  This function key is correctly distributed since it is the same as that of Lemma 3.4.

- **Case** Type-2 and $(i_\delta, j_\delta) \notin S$: It implicitly calculates

$$\begin{aligned} f_Y =& \sum_{(i,j)\in S} TRF_{i,j}(x_{i,j}) = \sum_{(i,j)\in A} f'_{i,j} + \sum_{(i,j)\in \overline{A}} (1 + 1/a) f'_{i,j} \\ =& \sum_{(i,j)\in S} f'_{i,j} + \sum_{(i,j)\in \overline{A}} (1/a) f'_{i,j} = f'_S + f'_{\overline{A}} \end{aligned}$$

  by retrieving $f'_{i,j}$ for the set A and selecting a fixed random $f'_{i,j}$ for the set $\overline{A}$ since $\overline{A} \neq \emptyset$ where $f'_S = \sum_{(i,j)\in S} f'_{i,j}$ and $f'_{\overline{A}} = \sum_{(i,j)\in \overline{A}} f'_{i,j}$. It chooses random exponents $r'_1, r'_2, r'_3 \in \mathbb{Z}_n$ and implicitly defines the randomness as

$$r_1 := r'_1/b + \phi' r'_3/ab,\ r_2 := \phi' r'_2/b,\ r_3 := -f'_{\overline{A}} r'_3/a - \rho' r'_2/b$$

  where r_1, r_2, and r_3 are independent random values since $\sum_{(i,j)\in \overline{A}} f'_{i,j} \neq 0$ except with negligible probability. Next, it creates a function key by using the randomness as

$$\begin{aligned} K_0 =& (\hat{g}^a)^{f'_S r'_1} \hat{g}^{f'_S \phi' r'_3} \hat{g}^{f'_{\overline{A}} r'_1} (\hat{g}^a)^{\omega'_1 \phi' r'_2} (\hat{g}^b)^{-\omega'_2 f'_{\overline{A}} r'_3} (\hat{g}^a)^{-\omega'_2 \rho' r'_2}, \\ K_1 =& (\hat{g}^a)^{r'_1} \hat{g}^{\phi' r'_3},\ K_2 = (\hat{g}^a)^{\phi' r'_2},\ K_3 = (\hat{g}^b)^{-f'_{\overline{A}} r'_3} (\hat{g}^a)^{-\rho' r'_2}. \end{aligned}$$

  This function key is correctly distributed since it is the same as that of Lemma 3.4.

- **Case** Type-2 and $(i_\delta, j_\delta) \in S$: Let $S_\delta = S \setminus \{(i_\delta, j_\delta)\}$, $A_\delta = A \setminus \{(i_\delta, j_\delta)\}$, and $\overline{A}_\delta = \overline{A} \setminus \{(i_\delta, j_\delta)\}$. It implicitly calculates

$$\begin{aligned} f_Y =& \sum_{(i,j)\in S} TRF_{i,j}(x_{i,j}) \\ =& \sum_{(i,j)\in A_\delta} f'_{i,j} + \sum_{(i,j)\in \overline{A}_\delta} (1 + 1/a) f'_{i,j} + (\tau/a + 1/ab) f'_{i_\delta, j_\delta} \\ =& \sum_{(i,j)\in S_\delta} f'_{i,j} + \sum_{(i,j)\in \overline{A}_\delta} (1/a) f'_{i,j} + (\tau/a + 1/ab) f'_{i_\delta, j_\delta} \\ =& f'_{S_\delta} + (1/a) f'_{\overline{A}_\delta} + (\tau/a + 1/ab) f'_{i_\delta, j_\delta} \end{aligned}$$

by retrieving $f'_{i,j}$ for the set A and selecting a fixed random $f'_{i,j}$ for the set $\overline{A}$ since $\overline{A} \neq \emptyset$ where $f'_{S_\delta} = \sum_{(i,j)\in S_\delta} f'_{i,j}$ and $f'_{\overline{A}_\delta} = \sum_{(i,j)\in \overline{A}_\delta} f'_{i,j}$ and $\tau = 0$ if $x_{i_\delta,j_\delta} = x^*_{\mu,i_\delta,j_\delta}$ and $\tau = 1$ otherwise.

It chooses random exponents $r'_1, r'_2, r'_3 \in \mathbb{Z}_n$ and implicitly defines the randomness as

$$\begin{aligned}
r_1 :=& \phi' r'_1/b + \phi' r'_3/ab,\ r_2 := \phi' r'_2/b,\\
r_3 :=& -(f'_{\overline{A}_\delta} + \tau f'_{i_\delta,j_\delta}) r'_3/a - (f'_{i_\delta,j_\delta} r'_1 + \rho' r'_2)/b - f'_{i_\delta,j_\delta} r'_3/ab
\end{aligned}$$

where r_1, r_2, r_3 are independent random values since $f'_{\overline{A}_\delta} + \tau f'_{i_\delta,j_\delta} \neq 0$ except with negligible probability. Next, it creates a function key by using the randomness as

$$\begin{aligned}
K_0 =& (\hat{g}^a)^{f'_{S_\delta}\phi' r'_1} \hat{g}^{f'_{S_\delta}\phi' r'_3} \hat{g}^{f'_{\overline{A}_\delta}\phi' r'_1} \left(\hat{g}^{\tau f'_{i_\delta,j_\delta}}\right)^{\phi' r'_1} \left(\hat{g}^{\tau f'_{i_\delta,j_\delta}}\right)^{\phi' r'_3/a} (\hat{g}^a)^{\omega'_1 \phi' r'_2}\\
& (\hat{g}^b)^{-\omega'_2 (f'_{\overline{A}_\delta} + \tau f'_{i_\delta,j_\delta}) r'_3} (\hat{g}^a)^{-\omega'_2 (f'_{i_\delta,j_\delta} r'_1 + \rho' r'_2)} \hat{g}^{-\omega'_2 f'_{i_\delta,j_\delta} r'_3},\\
K_1 =& (\hat{g}^a)^{\phi' r'_1} \hat{g}^{\phi' r'_3},\ K_2 = (\hat{g}^a)^{\phi' r'_2},\\
K_3 =& (\hat{g}^b)^{-(f'_{\overline{A}_\delta} r'_3 + \tau f'_{i_\delta,j_\delta} r'_3)} (\hat{g}^a)^{-(f'_{i_\delta,j_\delta} r'_1 + \rho' r'_2)} \hat{g}^{-f'_{i_\delta,j_\delta} r'_3}.
\end{aligned}$$

This function key is correctly distributed by the following equation

$$\begin{aligned}
K_0 =& (\hat{g}^{ab})^{f_Y r_1} \left(\hat{g}^{ab\omega'_1 + \rho'}\right)^{r_2} \left(\hat{g}^{ab\omega'_2 + \phi'}\right)^{r_3}\\
=& \left(\hat{g}^{ab f'_{S_\delta}} \hat{g}^{b f'_{\overline{A}_\delta}}\right)^{\phi' r'_1/b + \phi' r'_3/ab} \left(\hat{g}^{b\tau f'_{i_\delta,j_\delta}} \hat{g}^{f'_{i_\delta,j_\delta}}\right)^{\phi' r'_1/b + \phi' r'_3/ab}\\
& \left(\hat{g}^{ab\omega'_1 + \rho'}\right)^{\phi' r'_2/b} \left(\hat{g}^{ab\omega'_2 + \phi'}\right)^{r_3}\\
=& \left(\hat{g}^{a f'_{S_\delta}}\right)^{\phi' r'_1} \left(\hat{g}^{f'_{S_\delta}}\right)^{\phi' r'_3} \left(\hat{g}^{f'_{\overline{A}_\delta}}\right)^{\phi' r'_1} \left(\hat{g}^{f'_{\overline{A}_\delta}}\right)^{\phi' r'_3/a}\\
& \left(\hat{g}^{\tau f'_{i_\delta,j_\delta}}\right)^{\phi' r'_1} \left(\hat{g}^{\tau f'_{i_\delta,j_\delta}}\right)^{\phi' r'_3/a} \left(\hat{g}^{f'_{i_\delta,j_\delta}}\right)^{\phi' r'_1/b} \left(\hat{g}^{f'_{i_\delta,j_\delta}}\right)^{\phi' r'_3/ab}\\
& \left(\hat{g}^{a\omega'_1}\right)^{\phi' r'_2} \left(\hat{g}^{\rho'}\right)^{\phi' r'_2/b} \left(\hat{g}^{ab\omega'_2 + \phi'}\right)^{-(f'_{\overline{A}_\delta} + \tau f'_{i_\delta,j_\delta}) r'_3/a - (f'_{i_\delta,j_\delta} r'_1 + \rho' r'_2)/b - f'_{i_\delta,j_\delta} r'_3/ab}\\
=& (\hat{g}^a)^{f'_{S_\delta}\phi' r'_1} \hat{g}^{f'_{S_\delta}\phi' r'_3} \hat{g}^{f'_{\overline{A}_\delta}\phi' r'_1} \left(\hat{g}^{\tau f'_{i_\delta,j_\delta}}\right)^{\phi' r'_1} \left(\hat{g}^{\tau f'_{i_\delta,j_\delta}}\right)^{\phi' r'_3/a} (\hat{g}^a)^{\omega'_1 \phi' r'_2}\\
& (\hat{g}^b)^{-\omega'_2 (f'_{\overline{A}_\delta} + \tau f'_{i_\delta,j_\delta}) r'_3} (\hat{g}^a)^{-\omega'_2 (f'_{i_\delta,j_\delta} r'_1 + \rho' r'_2)} \hat{g}^{-\omega'_2 f'_{i_\delta,j_\delta} r'_3}.
\end{aligned}$$

If this is a ciphertext query for a client index i, a message vector $\vec{x}_i = (x_{i,1}, \ldots, x_{i,\ell})$, and a time period $T \neq T^*$, then $\mathcal{B}$ retrieves $(T, H(T) = (g^{ab})^{h'}, h')$ from the H-list and generates a ciphertext as follows: For each $j \in [\ell]$, it performs:

- **Case** $(i,j) \neq (i_\delta, j_\delta)$: If $x_{i,j} = x^*_{\mu,i,j}$, it creates $C_{i,j,1} = (g^{ab})^{h' f'_{i,j}}$ since $TRF_{i,j}(x_{i,j}) := f'_{i,j}$. Otherwise $(x_{i,j} \neq x^*_{\mu,i,j})$, it creates $C_{i,j,1} = (g^{ab} g^b)^{h' f'_{i,j}}$ since $TRF_{i,j}(x_{i,j}) := (1 + 1/a) f'_{i,j}$.
- **Case** $(i,j) = (i_\delta, j_\delta)$: If $x_{i,j} = x^*_{\mu,i,j}$, it creates $C_{i,j,1} = g^{h' f'_{i,j}}$ since $TRF_{i,j}(x_{i,j}) := (1/ab) f'_{i,j}$. Otherwise $(x_{i,j} \neq x^*_{\mu,i,j})$, it creates $C_{i,j,1} = (g^b g)^{h' f'_{i,j}}$ since $TRF_{i,j}(x_{i,j}) := (1/a + 1/ab) f'_{i,j}$.

Next, it generates a ciphertext as $\left(\left\{C_{i,j,1}\right\}_{j=1}^{\ell}, C_{i,2} = (g^{ab})^{h' \omega'_{i,1}} g^{h' \rho'_i}, C_{i,3} = (g^{ab})^{h' \omega'_{i,2}} g^{h' \phi'_i}\right)$.

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

We have shown that our MC-HVE scheme provides the selective single-challenge security when all clients are not corrupted. Now we show that our MC-HVE scheme still provides the selective single-challenge security when corrupted clients are statically fixed. In other words, if there is an attacker $\mathcal{A}$ that breaks the selective single-challenge security with static corruptions of the MC-HVE scheme, then another algorithm $\mathcal{B}$ that uses the attacker $\mathcal{A}$ as a sub-algorithm can break the selective single-challenge security with no corruptions of the MC-HVE scheme.

The basic idea of this proof is that a simulator directly selects encryption keys for corrupted clients. In this case, the generation of function keys can be problem since function key elements composed of secret components corresponding to corrupted clients and uncorrupted clients. Fortunately, in our MC-HVE scheme, it is possible to convert a function key SK for uncorrupted clients to another function key SK' associated with all clients by using the encryption keys of corrupted clients. The detailed proof of this is described in the following theorem.

Theorem 3.6. *The above MC-HVE scheme is selectively single-challenge IND-secure with static corruptions in the random oracle model if the MC-HVE scheme is selectively single-challenge IND-secure with no corruptions.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that breaks the selective single-challenge IND-security with static corruptions. By using $\mathcal{A}$, a simulator $\mathcal{B}$ try to break the selective single-challenge IND-security with no corruptions played by a challenger $\mathcal{C}$. The simulator $\mathcal{B}$ is described as follows:

Init: $\mathcal{A}$ submits the set of corrupted client indexes $\bar{I}$, two challenge message lists X_0^*, X_1^*, and a challenge time period T^*. Let $I = \{1,\ldots,n\} \setminus \bar{I}$ be the set of uncorrupted client indexes. $\mathcal{B}$ submits two challenge message lists X_0^*, X_1^*, the challenge time period T^* to $\mathcal{C}$. Note that $\mathcal{C}$ plays the selective single-challenge game with no corruptions for the set I.

Setup: $\mathcal{B}$ receives PP from $\mathcal{C}$. It chooses random PRF keys $\{z_i\}_{i\in\bar{I}}$ and random exponents $\{\omega_{i,1},\omega_{i,2}\}_{i\in\bar{I}}$. Next, it gives $\{EK_i = (z_i,\omega_{i,1},\omega_{i,2})\}_{i\in\bar{I}}$ and PP to $\mathcal{A}$. It derives $z_{i,j} = PRF(z_i, j)$ for all $i \in \bar{I}$ and $j \in [\ell]$ for later use.

Challenge: $\mathcal{B}$ receives challenge ciphertexts $\{CT_{i,T^*}\}_{i\in I}$ from $\mathcal{C}$ and gives $\{CT_{i,T^*}\}_{i\in I}$ to $\mathcal{A}$.

Query: $\mathcal{A}$ adaptively requests hash, function key, and ciphertext queries. $\mathcal{B}$ handles these queries as follows:

If this is a hash query for a time period T, then $\mathcal{B}$ relays this query to $\mathcal{C}$ and gives the response of $\mathcal{C}$ to $\mathcal{A}$.

If this is a function key query for a list of vectors $Y = (\vec{y}_1,\ldots,\vec{y}_n)$, then $\mathcal{B}$ proceeds as follows:

1. It first sets $Y' = \{y_i\}_{i\in I}$ derived from Y. It requests a function key for Y' to $\mathcal{C}$ and receives a function key $SK_{Y'} = (K'_0, K'_1, K'_2, K'_3)$ for the set I.
2. Let $\overline{S}$ be the set of index tuples (i,j) that are not wildcard positions in $\overline{Y} = \{\vec{y}_i\}_{i\in\bar{I}}$. It calculates $f_{\overline{Y}} = \sum_{(i,j)\in\overline{S}} PRF(z_{i,j}, y_{i,j})$ and computes $K_0 = K'_0 \cdot (K'_1)^{f_{\overline{Y}}} \cdot (K'_2)^{\sum_{i\in\bar{I}}\omega_{i,1}} \cdot (K'_3)^{\sum_{i\in\bar{I}}\omega_{i,2}}$.
3. It gives $SK_Y = (K_0, K_1 = K'_1, K_2 = K'_2, K_3 = K'_3)$ to $\mathcal{A}$.

If this is a ciphertext query for a client index $i \in I$, a message vector $\vec{x}_i$, and a time period $T \neq T^*$, then $\mathcal{B}$ relays this query to $\mathcal{C}$ and gives $CT_{i,T}$ from $\mathcal{C}$ to $\mathcal{A}$.

Guess: $\mathcal{A}$ outputs a guess μ'. $\mathcal{B}$ also outputs μ'. □

Table 1: Comparison of MC-HVE schemes in bilinear groups

Scheme	PP Size	SK Size	CT Size	GenKey	Encrypt	Decrypt
KPEJ [41]	$O(1)$	$2n\ell$	$2n\ell$	$3n\ell$E	3ℓE	$2n\ell$P
Ours	$O(1)$	4	$n(\ell+2)$	6E	$(\ell+2)$E	4P

Let n be the number of clients and ℓ be the number of attributes in each client. We count the number of group elements to measure the size. We use symbols E for exponentiation and P for pairing.

3.6 Discussions

Efficiency Analysis. We compare our MC-HVE scheme with other MC-HVE schemes in Table 1. In our MC-HVE scheme, a ciphertext consists of $\ell+2$ group elements per each client, and $n(\ell+2)$ group elements for n clients. A function key in our scheme is very compact because it consists of just 4 group elements, regardless of the number of clients. The encryption algorithm consists of ℓ PRF operations and $\ell+2$ exponentiation operations per each client. The function key generation algorithm consists of $n\ell$ PRF operations and 6 exponentiation operations. The decryption algorithm consists of $O(n\ell)$ multiplication operations and only 4 pairing operations, which are independent of the number of clients and the number of messages. In the MC-HVE scheme of Kamp et al. [41], a ciphertext consists of $2n\ell$ group elements and a function key consists of $2n\ell$ group elements. The decryption algorithm of their scheme requires $2n\ell$ pairing operations which are linearly dependent on the number of clients and the number of messages. Therefore, our MC-HVE scheme is more efficient in terms of ciphertext size, function key size, and the performance of the decryption algorithm compared to the previous MC-HVE scheme.

Supporting Comparison and Range Queries. Boneh and Waters [14] presented an HVE scheme which is a special kind of predicate encryption, and showed that it can be extended to support conjunctive comparison, range, and subset queries by carefully encoding the messages of ciphertexts and the attributes of function keys. Our MC-HVE scheme also can be extended to support conjunctive comparison, range, and subset queries in the multi-client setting by using the same extension method of them. However, this extension method only works for a small domain $\{1,\ldots,D\}$ since the ciphertext of this extension method consists of $O(D)$ group elements. In order to overcome this problem, we propose a multi-client range query encryption scheme which supports conjunctive range queries for a large domain in the next section.

Support for Dynamic Number of Clients. The number of clients supported by our MC-HVE scheme is fixed at n. In a real environment, the number of clients can start with a small value and grow to a large value over time. Fortunately, our MC-HVE scheme allows the number of clients to grow dynamically over time because it supports a wildcard in the attributes of a function key. Let n be the maximum number of clients and n_1 be the current number of clients. In this case, a trusted center generates a function key so that all vectors $\vec{y}_{n_1+1},\ldots,\vec{y}_n$ have a wildcard character. The encryption algorithm is the same as before, but the center additionally creates ciphertext elements $\{C_{i,2},C_{i,3}\}$ for $n_1+1 \leq i \leq n$ and publishes these elements. The decryption algorithm is the same as before. If the number of clients increases from n_1 to n_2 at a time period T', a new function key issued by the center can process ciphertexts associated with the same time period $T'' \geq T'$ which are generated by n_2 number of clients.

Support for Different Number of Attributes. In our MC-HVE scheme, the number of attributes supported by each client is fixed to the same value ℓ. With a slight modification, our MC-HVE scheme can handle a case where the number of attributes for each individual client is different by using the fact that a wildcard can

be included in the attributes of a function key. For example, suppose that a client with an index i supports ℓ_i number of attributes such that $\ell_i \leq \ell$ where ℓ is the maximum number of attributes in each client. In this case, the encryption algorithm is same as before except that it creates ciphertext elements up to ℓ_i. A center, which stores all number of attributes for individual clients, creates a function key which is set with a wildcard for all attributes $y_{i,j}$ in a client index i and a location j such that $1 \leq i \leq n$ and $\ell_i + 1 \leq j \leq \ell$.

Removing Random Oracles. To provide multi-client functionality, our MC-HVE scheme uses a hash function which is modeled as a random oracle in the security proof. That is, because each client generates a ciphertext by using the same hash $H(T)$ for the same time period T, all clients are synchronized to use the same encryption random even if they independently perform the encryption algorithm. However, it is a very strong assumption that the hash function works like the random oracle. Thus, it is necessary to design an efficient MC-HVE scheme that does not use the random oracle. One way to remove the random oracle is to provide all group elements associated with individual time periods to the public parameters. That is, if the maximum time period T_{max} is restricted to be a polynomial value, all group elements $h_1, \ldots, h_{T_{max}}$ are provided in the public parameters and the encryption algorithm uses h_T instead of $H(T)$.

4 Multi-Client Range Query Encryption

In this section, we propose a multi-client range query encryption scheme by using a binary tree and prove the selective security under static assumptions.

4.1 Design Principle

To support efficient range queries, we use a binary tree to represent values. The method to effectively represent values and ranges using a binary tree was used by Shi et al. [37] to design a multi-dimensional range query on encrypted data (MRQED) scheme. In order to design a multi-client range query encryption (MC-RQE) scheme, we try to combine a binary tree and a simplified MC-HVE scheme for each node in the tree. However, the method of simply applying the MC-HVE scheme to the binary tree can lead to collusion attacks such that different function keys can be mixed to derive new function keys. In order to prevent this collusion attacks, we apply a secret sharing scheme so that only the function key elements contained in the same function key can be combined for the decryption process.

4.2 Binary Tree

A perfect binary tree $\mathcal{BT}$ is a tree data structure in which all internal nodes have two child nodes and all leaf nodes have the same depth. Let $D = 2^\ell$ be the number of leaf nodes in $\mathcal{BT}$. The number of all nodes in $\mathcal{BT}$ is $2D-1$ and we denote v_i as a node in $\mathcal{BT}$. The depth d_i of a node v_i is the length of the path from a root node to the node. The root node of a tree has depth zero. The depth of $\mathcal{BT}$ is the length of the path from the root node to a leaf node. A level of $\mathcal{BT}$ is a set of all nodes at given depth. Each node $v_i \in \mathcal{BT}$ has an identifier $L_i \in \{0,1\}^*$ which is a fixed and unique string. A subtree $\mathcal{T}_i$ in $\mathcal{BT}$ is defined as a tree that is rooted at a node $v_i \in \mathcal{BT}$.

Let $\mathcal{BT}$ be a perfect binary tree with $D = 2^\ell$ leaf nodes. We define the following two functions in $\mathcal{BT}$.

Path($\mathcal{BT}, x$)**:** It takes as input a tree $\mathcal{BT}$ and a value $x \in [D]$. Let v_x be a leaf node in $\mathcal{BT}$ that is assigned to x. Let $(v_{x,0}, v_{x,1}, \ldots, v_{x,\ell})$ be the path from a root node $v_{x,0}$ to the leaf node $v_{x,\ell} = v_x$. It sets a path set $PV_x = (v_{x,0}, \ldots, v_{x,\ell})$ and outputs PV_x.

Cover($\mathcal{BT}, y = (y_L, y_R)$): It takes as input a tree $\mathcal{BT}$ and a range $(y_L, y_R) \in [D]^2$ such that $y_L \leq y_R$. Let v_L and v_R be leaf nodes assigned to y_L and y_R respectively. Let S be the set of leaf nodes from v_L to v_R, and R be the set of all leaf nodes excluding S. It computes the Steiner Tree $ST(R)$ which is the subtree of $\mathcal{BT}$ that connects all nodes in R with the root node. Let $\mathcal{T}_{y,1}, \ldots \mathcal{T}_{y,m}$ be all the subtrees of $\mathcal{BT}$ that hang off $ST(R)$, that is all subtrees whose roots $v_{y,1}, \ldots v_{y,m}$ are not in $ST(R)$ but adjacent to nodes of outdegree 1 in $ST(R)$. It outputs a cover set $CV_y = (v_{y,1}, \ldots, v_{y,m})$.

For the path set PV_x and the cover set CV_y defined in the above two algorithms, the following properties are established. 1) If $x \in [y_L, y_R]$, then there is only one node in $PV_x \cap CV_y$. 2) If $x \notin [y_L, y_R]$, then $PV_x \cap CV_y = \emptyset$. 3) The maximum size of CV_y is $2 \log D$.

4.3 Predicate for Conjunctive Range

Let $Y = (y_1, \ldots, y_n)$ be a list of ranges where $y_i = (y_{i,L}, y_{i,R})$ and $X = (x_1, \ldots, x_n)$ be a list of values where $x_i \in [D]$. We define a predicate f_Y over $\mathcal{D}^n$ for conjunctive range as

$$f_Y(X) = \begin{cases} 1 & \text{if } x_i \in [y_{i,L}, y_{i,R}] \text{ for all } i \in [n] \\ 0 & \text{otherwise.} \end{cases}$$

4.4 Construction

We propose an MC-RQE scheme that combines a simplified MC-HVE scheme with a binary tree to efficiently process range queries. The detailed description of our scheme is given as follows.

MC-RQE.Setup($1^\lambda, n$): Let n be the number of clients and ℓ be the depth of a perfect binary tree. It obtains a bilinear group $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ of prime order p by running $\mathcal{G}(1^\lambda)$. Let $g, \hat{g}$ be random generators of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. It chooses random PRF keys $\{z_i\}_{i=1}^n$, random exponents $\{\omega_{i,1}, \omega_{i,2}\}_{i=1}^n$, and a random element $\hat{v} \in \hat{\mathbb{G}}$. It selects a hash function H from the family of hash functions. It outputs a master key $MK = \big(\{z_i\}_{i=1}^n, \hat{v}, \{\hat{w}_{i,1} = \hat{v}^{\omega_{i,1}}, \hat{w}_{i,2} = \hat{v}^{\omega_{i,2}}\}_{i=1}^n\big)$, encryption keys $\{EK_i = (z_i, \omega_{i,1}, \omega_{i,2})\}_{i=1}^n$ for all clients, and public parameters

$$PP = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, n, \ell, H\big).$$

MC-RQE.GenKey(Y, MK, PP): Let $\mathcal{BT}$ be a perfect binary tree with depth ℓ and $Y = (y_1 = (y_{1,L}, y_{1,R}), \ldots, y_n = (y_{n,L}, y_{n,R}))$ where $y_{i,L} \leq y_{i,R}$.

1. It first selects random exponents $\gamma_1, \ldots, \gamma_{n-1} \in \mathbb{Z}_p$ and sets $\gamma_n = -\sum_{i=1}^{n-1} \gamma_i$ to satisfy $\sum_{i=1}^n \gamma_i = 0$.
2. For each client $i \in [n]$, it performs the following steps:
   (a) It obtains a cover set $CV_i = (v_{i,1}, \ldots, v_{i,m})$ by running **Cover**$(\mathcal{BT}, (y_{i,L}, y_{i,R}))$.
   (b) For each node $v_{i,j} \in CV_i$, it calculates $f_{i,j} = PRF(z_i, v_{i,j})$ and computes a node key by selecting random exponents $r_{i,j,1}, r_{i,j,2}, r_{i,j,3} \in \mathbb{Z}_p$ as

$$NK_{i,j} = \Big(K_{i,j,0} = \hat{g}^{\gamma_i} \hat{v}^{f_{i,j} \cdot r_{i,j,1}} \hat{w}_{i,1}^{r_{i,j,2}} \hat{w}_{i,2}^{r_{i,j,3}},\ K_{i,j,1} = \hat{v}^{r_{i,j,1}},\ K_{i,j,2} = \hat{v}^{r_{i,j,2}},\ K_{i,j,3} = \hat{v}^{r_{i,j,3}}\Big).$$

   (c) It creates a client function key $SK_i = \big(CV_i, NK_{i,1}, \ldots, NK_{i,m_i}\big)$ where m_i is the size of CV_i.
3. Finally, it outputs a function key $SK_Y = \big(SK_1, \ldots, SK_n\big)$.

MC-RQE.Encrypt(x_i, T, EK_i, PP)**:** Let $\mathcal{BT}$ be a perfect binary tree with depth ℓ and $EK_i = (z_i, \omega_{i,1}, \omega_{i,2})$. It first obtains a path set $PV_i = (v_{i,0}, v_{i,1}, \ldots, v_{i,\ell})$ by running **Path**$(\mathcal{BT}, x_i)$. For each node $v_{i,j} \in PV_i$, it calculates $f_{i,j} = PRF(z_i, v_{i,j})$ and computes $C_{i,j,1} = H(T)^{f_{i,j}}$. It outputs a ciphertext as

$$CT_{i,T} = \Big(\big\{ C_{i,j,1} \big\}_{j=0}^{\ell},\ C_{i,2} = H(T)^{\omega_{i,1}},\ C_{i,3} = H(T)^{\omega_{i,2}} \Big).$$

MC-RQE.Decrypt$(CT_{1,T}, \ldots, CT_{n,T}, SK_Y, PP)$**:** Let $CT_{i,T} = (\{C_{i,j,1}\}_{j=0}^{\ell}, C_{i,2}, C_{i,3})$ and $SK_Y = (SK_1, \ldots, SK_n)$ where $SK_i = (CV_i, NK_{i,1}, \ldots, NK_{i,m_i})$ and $NK_{i,j} = (K_{i,j,0}, K_{i,j,1}, K_{i,j,2}, K_{i,j,3})$.

1. For each combination $\Big\{ \big((C_{i,j_i,1}, C_{i,2}, C_{i,3}), (K_{i,j'_i,0}, K_{i,j'_i,1}, K_{i,j'_i,2}, K_{i,j'_i,3}) \big) \Big\}_{i=1}^{n}$ of CT_T and SK_Y where $(C_{i,j_i,1}, C_{i,2}, C_{i,3}) \in CT_{i,T}$ for some j_i and $(K_{i,j'_i,0}, K_{i,j'_i,1}, K_{i,j'_i,2}, K_{i,j'_i,3}) \in SK_i$ for some j'_i with the restriction that the node depth of j_i and the node depth of j'_i are the same, it checks the following equation

$$e(H(T), \prod_{i=1}^{n} K_{i,j'_i,0}) \stackrel{?}{=} \prod_{i=1}^{n} e(C_{i,j_i,1}, K_{i,j'_i,1}) \cdot \prod_{i=1}^{n} e(C_{i,2}, K_{i,j'_i,2}) \cdot \prod_{i=1}^{n} e(C_{i,3}, K_{i,j'_i,3}).$$

   If this check succeeds, it outputs 1 since it found a matching combination. Otherwise, it continues to the next combination.

2. Finally, it outputs 0 since it fails to find a matching combination.

4.5 Correctness

If $f_X(Y) = 1$ is satisfied for the message X of a ciphertext and the range Y of a function key, then one common node exists for each client by the property of the path set in a ciphertext and the cover set in a function key. Thus, we can derive the following equation since the PRF value $f_{i,j}$ of the ciphertext matches one of the PRF values $\{f_{i,j}\}$ of the function key.

$$\begin{aligned}
e(H(T), \prod_{i=1}^{n} K_{i,j_i,0}) &= e\big(H(T), \prod_{i=1}^{n} \hat{g}^{\gamma_i} \hat{v}^{f_{i,j_i} \cdot r_{i,j_i,1}} \hat{w}_{i,1}^{r_{i,j_i,2}} \hat{w}_{i,2}^{r_{i,j_i,3}}\big) \\
&= e\big(H(T), \hat{g}^{\sum_{i=1}^{n} \gamma_i} \prod_{i=1}^{n} \hat{v}^{f_{i,j_i} \cdot r_{i,j_i,1}} \prod_{i=1}^{n} \hat{v}^{\omega_{i,1} r_{i,j_i,2}} \prod_{i=1}^{n} \hat{v}^{\omega_{i,2} r_{i,j_i,3}}\big) \\
&= \prod_{i=1}^{n} e\big(H(T)^{f_{i,j_i}}, \hat{v}^{r_{i,j_i,1}}\big) \cdot \prod_{i=1}^{n} e\big(H(T)^{\omega_{i,1}}, \hat{v}^{r_{i,j_i,2}}\big) \cdot \prod_{i=1}^{n} e\big(H(T)^{\omega_{i,2}}, \hat{v}^{r_{i,j_i,3}}\big) \\
&= \prod_{i=1}^{n} e\big(C_{i,j_i,1}, K_{i,j_i,1}\big) \cdot \prod_{i=1}^{n} e\big(C_{i,2}, K_{i,j_i,2}\big) \cdot \prod_{i=1}^{n} e\big(C_{i,3}, K_{i,j_i,3}\big).
\end{aligned}$$

4.6 Security Analysis

In order to prove the selective single-challenge weak security of our MC-RQE scheme, we first prove this security in the absence of corrupted clients, and then prove this security in the case where the corrupted clients are statically fixed.

The basic idea of proving the selective single-challenge weak security of the MC-RQE scheme in the no corrupted clients setting is similar to the proof of the MC-HVE scheme. That is, all challenge ciphertext elements associated with different nodes among the path nodes which are associated with the challenge

messages X_0^*, X_1^* are changed to random elements. The overall structure of the security proof is similar to that of the MC-HVE scheme. However, there is a considerable difference from the proof of the MC-HVE scheme since the proof of our MC-RQE scheme must deal with nodes in the binary tree. For further details of this proof, please refer to the following theorem.

Theorem 4.1. *The above MC-RQE scheme is selectively single-challenge weak IND-secure with no corruptions in the random oracle model if the A3DH assumption holds and the PRF is secure.*

Proof. Suppose there exists an adversary that breaks the selective single-challenge weak IND-security game with no corruptions. The adversary initially submits two challenge messages $X_0^* = (x_{0,1}^*, \ldots, x_{0,n}^*)$, $X_1^* = (x_{1,1}^*, \ldots, x_{1,n}^*)$, and challenge time T^*. We obtain $PV_{\mu,i}^* = (v_{\mu,i,0}^*, v_{\mu,i,1}^*, \ldots, v_{\mu,i,\ell}^*)$ by running $\mathbf{Path}(\mathcal{BT}, x_{\mu,i}^*)$ for all $\mu \in \{0,1\}$ and $i \in [n]$. From this path set $PV_{\mu,i}^*$, we define two node sets $E_i = \{v_{\mu,i,j}^* : (v_{\mu,i,j}^* \in PV_{\mu,i}^*) \wedge (v_{0,i,j}^* = v_{1,i,j}^*)\}$ and $\overline{E}_i = \{v_{\mu,i,j}^* : (v_{\mu,i,j}^* \in PV_{\mu,i}^*) \wedge (v_{0,i,j}^* \neq v_{1,i,j}^*)\}$ which are partitions of all nodes in the path set. To argue that the adversary cannot win this game, we define a sequence of hybrid games as follows:

Game $\mathbf{G}_0$. The first game $\mathbf{G}_0$ denotes the selective single-challenge weak security game which is defined in Section 2.3 where $\overline{I} = \emptyset$ for no corruptions.

Game $\mathbf{G}_1$. In this game $\mathbf{G}_1$, we replace the pseudo-random functions $PRF(z_i, x)$ with the truly random function $TRF_i(x)$ for all i. This change can be easily done by the security of PRF.

Game $\mathbf{G}_2$. This game $\mathbf{G}_2$ is similar to the game $\mathbf{G}_1$ except the generation of the challenge ciphertext. In this game, the simulator slightly changes the generation of challenge ciphertext elements as $C_{i,2} = H(T^*)^{\omega_{i,1}} g^{\rho_i}$ and $C_{i,3} = H(T^*)^{\omega_{i,2}} g^{\phi_i}$ for all index $i \in [n]$ with random exponents ρ_i and ϕ_i.

Game $\mathbf{G}_3$. This final game $\mathbf{G}_3$ differs from the game $\mathbf{G}_2$ in that the challenge ciphertext elements $\{C_{i,j,1}\}$ associated with $\overline{E}_i$ are generated as random elements for all $i \in [n]$. Thus the challenge ciphertext gives no information about the challenge message X_μ^*. Therefore, the advantage of the adversary in this final game is zero.

Let $S_{\mathcal{A}}^{\mathbf{G}_i}$ 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

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{SE\text{-}1\text{-}IND}(\lambda) &\leq \sum_{i=1}^{3} \left| \Pr[S_{\mathcal{A}}^{\mathbf{G}_{i-1}}] - \Pr[S_{\mathcal{A}}^{\mathbf{G}_i}] \right| + \Pr[S_{\mathcal{A}}^{\mathbf{G}_3}] \\ &\leq n \cdot \mathbf{Adv}_{\mathcal{B}}^{PRF}(\lambda) + (1 + n\ell) \cdot \mathbf{Adv}_{\mathcal{B}}^{A3DH}(\lambda)\end{aligned}$$

where $n\ell$ is the size of the challenge message. This completes our proof. □

Lemma 4.2. *If the PRF is secure, then no polynomial-time adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage.*

Proof. This proof of this is relatively straightforward from the security of PRF. That is, we can use additional hybrid games that change a PRF to a truly random function. Note that there are at most n number of z_i in the security proof. □

Lemma 4.3. *If the A3DH assumption holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_1$ and $\mathbf{G}_2$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the A3DH assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e),g,g^a,g^b,g^{ab},g^{abc},\hat{g},\hat{g}^a,\hat{g}^b)$ and Z where $Z = g^c$ or $Z = R$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows.

Init: $\mathcal{A}$ submits challenge message lists $X_0^* = (x_{0,1}^*,\ldots,x_{0,n}^*)$, $X_1^* = (x_{1,1}^*,\ldots,x_{1,n}^*)$, and a challenge time period T^*. $\mathcal{B}$ flips a random coin $\mu \in \{0,1\}$ to fix X_μ^* as the target message list and obtains $PV_{\mu,i}^* = (v_{\mu,i,0}^*,\ldots,v_{\mu,i,\ell}^*)$ by running $\mathbf{Path}(\mathcal{BT},x_{\mu,i}^*)$ for all $i \in [n]$.
Setup: $\mathcal{B}$ proceeds the following steps:

1. It first selects random exponents $\{\omega_{i,1}',\omega_{i,2}',\rho_i',\phi_i'\}_{i=1}^n \in \mathbb{Z}_p$ and defines $\{\omega_{i,1} := \omega_{i,1}' + (1/ab)\rho_i', \omega_{i,2} := \omega_{i,2}' + (1/ab)\phi_i'\}_{i=1}^n$. Next, it implicitly sets $\hat{v} = \hat{g}^{ab}, \{\hat{w}_{i,1} = (\hat{g}^{ab})^{\omega_{i,1}'}\hat{g}^{\rho_i'}, \hat{w}_{i,2} = (\hat{g}^{ab})^{\omega_{i,2}'}\hat{g}^{\phi_i'}\}_{i=1}^n$.

2. It now defines the value of TRF by using lazy sampling. Let $v_{i,j}$ be a node for a client i and $PV_{\mu,i}^*$ be the challenge path set for the same client i. If $v_{i,j} \in PV_{\mu,i}^*$, then it defines $TRF_i(v_{i,j}) := f_{i,j}'$ by selecting a fixed random $f_{i,j}'$ for $v_{i,j}$. Otherwise ($v_{i,j} \notin PV_{\mu,i}^*$), it defines $TRF_i(v_{i,j}) := (1+1/a)f_{i,j}'$ by selecting a fixed random $f_{i,j}'$ for $v_{i,j}$. That is, it implicitly defines

$$\hat{v}^{TRF_i(v_{i,j})} := (\hat{g}^{ab})^{f_{i,j}'} \text{ if } v_{i,j} \in PV_{\mu,i}^*, \;\; \hat{v}^{TRF_i(v_{i,j})} := (\hat{g}^{ab}\hat{g}^{b})^{f_{i,j}'} \text{ if } v_{i,j} \notin PV_{\mu,i}^*.$$

3. For the target message list $X_\mu^* = (x_{\mu,1}^*,\ldots,x_{\mu,n}^*)$ with $PV_{\mu,i}^* = (v_{\mu,i,0}^*,\ldots,v_{\mu,i,\ell}^*)$ for all i, it selects a random $f_{i,j}'$ and defines $TRF_i(v_{\mu,i,j}^*) := f_{i,j}'$ for all $i \in [n], j \in [0,\ell]$.

4. It initializes a hash table H-list and publishes $PP = ((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e),g,\hat{g},n,\ell,H)$.

Challenge: $\mathcal{B}$ retrieves $\{f_{i,j}'\}$ since $TRF_i(v_{\mu,i,j}^*) := f_{i,j}'$ for the challenge messages X_μ^*. Next, it implicitly sets $H(T^*) = g^{abc}$ and creates challenge ciphertexts for the time period T^* as

$$\left(\left\{C_{i,j,1} = (g^{abc})^{f_{i,j}'}\right\}_{j=0}^{\ell}, C_{i,2} = (g^{abc})^{\omega_{i,1}'}Z^{\rho_i'}, C_{i,3} = (g^{abc})^{\omega_{i,2}'}Z^{\phi_i'}\right) \text{ for all } i \in [n].$$

If $Z = g^c$, then it plays $\mathbf{G}_2$ since ciphertext elements are well formed as

$$C_{i,2} = (g^{abc})^{\omega_{i,1}' + (1/ab)\rho_i'} = (g^{abc})^{\omega_{i,1}'}(g^c)^{\rho_i'}, \; C_{i,3} = (g^{abc})^{\omega_{i,2}' + (1/ab)\phi_i'} = (g^{abc})^{\omega_{i,2}'}(g^c)^{\phi_i'}.$$

Otherwise ($Z = R = g^d$), it plays $\mathbf{G}_3$ since ciphertext elements are created with setting $\rho_i = (-c+d)\rho_i'$, $\phi_i = (-c+d)\phi_i'$ as

$$C_{i,2} = H(T^*)^{\omega_{i,1}}g^{\rho_i} = (g^{abc})^{\omega_{i,1}' + (1/ab)\rho_i'}g^{(-c+d)\rho_i'} = (g^{abc})^{\omega_{i,1}'}(g^d)^{\rho_i'},$$
$$C_{i,3} = H(T^*)^{\omega_{i,2}}g^{\phi_i} = (g^{abc})^{\omega_{i,2}' + (1/ab)\phi_i'}g^{(-c+d)\phi_i'} = (g^{abc})^{\omega_{i,2}'}(g^d)^{\phi_i'}.$$

Query: $\mathcal{A}$ adaptively requests hash, function key, and ciphertext queries. $\mathcal{B}$ handles these queries as follows. If this is a hash query for a time period T, then $\mathcal{B}$ proceeds as follows: If T was queried before, then it retrieves $(T,h,-)$ from H-list and returns h. Otherwise, it performs:

- **Case** $T = T^*$: It sets $H(T^*) = g^{abc}$ and stores $(T^*,H(T^*),-)$ to H-list. It then returns g^{abc}.
- **Case** $T \neq T^*$: It selects a random exponent $h' \in \mathbb{Z}_p$ and stores $(T,H(T) = (g^{ab})^{h'},h')$ to H-list. It then returns $(g^{ab})^{h'}$.

If this is a function key query for a list of ranges $Y = (y_1, \ldots, y_n)$ where $y_i = (y_{i,L}, y_{i,R})$, then $\mathcal{B}$ generates a function key as follows:

1. It first derives cover sets $CV_1, \ldots, CV_n$ from $y_1, \ldots, y_n$. For each CV_i, it defines two node sets PV_i and $\overline{PV}_i$ as $PV_i = \{v_{i,j} : v_{i,j} \in CV_i \wedge v_{i,j} \in PV^*_{\mu,i}\}$ and $\overline{PV}_i = \{v_{i,j} : v_{i,j} \in CV_i \wedge v_{i,j} \notin PV^*_{\mu,i}\}$.

2. It also defines three index sets $I_1, I_2, I_3 \subseteq [n]$ which are partitions of client indexes. It defines $I_1 = \{i : \exists v_{i,j} \in CV_i$ such that $v_{i,j} \in PV_i\}$. It next selects a random index $\tilde{i} \in [n] \setminus I_1$ and defines $I_3 = \{\tilde{i}\}$. It then defines $I_2 = [n] \setminus (I_1 \cup I_3)$. By using these index sets, it will define γ_i differently depending on the index sets. For each client index $i \in I_2$, it chooses a random exponent γ'_i and implicitly defines $\gamma_i := \gamma'_i$.

3. For each client $i \in [n]$, it performs the following steps:

   (a) For each node $v_{i,j}$ such that $(i \in I_1) \wedge (v_{i,j} \in PV_i)$, it proceeds as follows: It defines $TRF_i(v_{i,j}) := f'_{i,j}$ by retrieving $f'_{i,j}$ chosen at the setup. It chooses random exponents $\gamma'_i, r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$, sets $\Delta_i = \rho'_i r'_{i,j,2} + \phi'_i r'_{i,j,3}$, and implicitly defines the randomness as

   $$\gamma_i := -\Delta_i/ab + \gamma'_i,\ r_{i,j,1} := r'_{i,j,1}/ab,\ r_{i,j,2} := r'_{i,j,2}/ab,\ r_{i,j,3} := r'_{i,j,3}/ab.$$

   Note that this γ_i is fixed for this index i since there is only one node $v_{i,j} \in PV_i$. Next, it creates a node key by using the defined randomness as

   $$K_{i,j,0} = \hat{g}^{\gamma'_i} \hat{g}^{f'_{i,j} \cdot r'_{i,j,1}} \hat{g}^{\omega'_{i,1} r'_{i,j,2}} \hat{g}^{\omega'_{i,2} r'_{i,j,3}},\ K_{i,j,1} = \hat{g}^{r'_{i,j,1}},\ K_{i,j,2} = \hat{g}^{r'_{i,j,2}},\ K_{i,j,3} = \hat{g}^{r'_{i,j,3}}.$$

   This node key is correctly distributed by the following equation

   $$\begin{aligned} K_{i,j,0} &= \hat{g}^{-(\rho'_i r'_{i,j,2} + \phi'_i r'_{i,j,3})/ab + \gamma'_i} \left(\hat{g}^{ab f'_{i,j}}\right)^{r'_{i,j,1}/ab} \left(\hat{g}^{ab\omega'_{i,1} + \rho'_i}\right)^{r'_{i,j,2}/ab} \left(\hat{g}^{ab\omega'_{i,2} + \phi'_i}\right)^{r'_{i,j,3}/ab} \\ &= \hat{g}^{\gamma'_i} \hat{g}^{f'_{i,j} \cdot r'_{i,j,1}} \hat{g}^{\omega'_{i,1} r'_{i,j,2}} \hat{g}^{\omega'_{i,2} r'_{i,j,3}}. \end{aligned}$$

   (b) For each node $v_{i,j}$ such that $(i \in I_1) \wedge (v_{i,j} \in \overline{PV}_i)$, it proceeds as follows: It defines $TRF_i(v_{i,j}) := (1 + 1/a) f'_{i,j}$ by retrieving $f'_{i,j}$ chosen at the setup. Recall that the random γ_i was already defined as $\gamma_i := -\Delta_i/ab + \gamma'_i$ for some node $v_{i,j'}$. It chooses random exponents $r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$ and implicitly defines the randomness as

   $$\begin{aligned} r_{i,j,1} &:= r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab,\ r_{i,j,2} := \phi'_i r'_{i,j,2}/b, \\ r_{i,j,3} &:= -f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b + \Delta_i/(\phi'_i ab). \end{aligned}$$

   Next, it creates a node key by using the defined randomness as

   $$\begin{aligned} K_{i,j,0} &= \hat{g}^{\gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}} \hat{g}^{\omega'_{i,2} \Delta_i/\phi'_i}, \\ K_{i,j,1} &= (\hat{g}^a)^{r'_{i,j,1}} \hat{g}^{\phi'_i r'_{i,j,3}},\ K_{i,j,2} = (\hat{g}^a)^{\phi'_i r'_{i,j,2}},\ K_{i,j,3} = (\hat{g}^b)^{-f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\rho'_i r'_{i,j,2}} \hat{g}^{\Delta_i/\phi'_i}. \end{aligned}$$

   This node key is correctly distributed by the following equation

   $$\begin{aligned} K_{i,j,0} =& \hat{g}^{-\Delta_i/ab + \gamma'_i} \left(\hat{g}^{ab f'_{i,j}} \hat{g}^{b f'_{i,j}}\right)^{r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab} \left(\hat{g}^{ab\omega'_{i,1} + \rho'_i}\right)^{\phi'_i r'_{i,j,2}/b} \\ & \left(\hat{g}^{ab\omega'_{i,2} + \phi'_i}\right)^{-f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b + \Delta_i/(\phi'_i ab)} \\ =& \hat{g}^{\gamma'_i} \left(\hat{g}^{ab f'_{i,j}}\right)^{r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab} \left(\hat{g}^{b f'_{i,j}}\right)^{r'_{i,j,1}/b} \left(\hat{g}^{ab\omega'_{i,1}}\right)^{\phi'_i r'_{i,j,2}/b} \\ & \left(\hat{g}^{ab\omega'_{i,2}}\right)^{-f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b + \Delta_i/(\phi'_i ab)} \\ =& \hat{g}^{\gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} \\ & (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}} \hat{g}^{\omega'_{i,2} \Delta_i/\phi'_i}. \end{aligned}$$

(c) For each node $v_{i,j}$ such that $(i \in I_2) \wedge (v_{i,j} \in \overline{PV}_i)$, it proceeds as follows: It defines $TRF_i(v_{i,j}) := (1+1/a)f'_{i,j}$ by using a fixed random $f'_{i,j}$ for $v_{i,j}$. Recall that γ_i was already defined as $\gamma_i := \gamma'_i$ for the set I_2. It chooses random exponents $r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$ and implicitly defines the randomness as

$$r_{i,j,1} := r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab,\ r_{i,j,2} := \phi'_i r'_{i,j,2}/b,\ r_{i,j,3} := -f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b.$$

Next, it creates a node key by using the defined randomness as

$$\begin{aligned}
K_{i,j,0} &= \hat{g}^{\gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}}, \\
K_{i,j,1} &= (\hat{g}^a)^{r'_{i,j,1}} \hat{g}^{\phi'_i r'_{i,j,3}},\ K_{i,j,2} = (\hat{g}^a)^{\phi'_i r'_{i,j,2}},\ K_{i,j,3} = (\hat{g}^b)^{-f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\rho'_i r'_{i,j,2}}.
\end{aligned}$$

This node key is correctly distributed by the following equation

$$\begin{aligned}
K_{i,j,0} =& \hat{g}^{\gamma'_i} \left(\hat{g}^{ab f'_{i,j}} \hat{g}^{b f'_{i,j}}\right)^{r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab} \left(\hat{g}^{ab\omega'_{i,1} + \rho'_i}\right)^{\phi'_i r'_{i,j,2}/b} \left(\hat{g}^{ab\omega'_{i,2} + \phi'_i}\right)^{-f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b} \\
=& \hat{g}^{\gamma'_i} \left(\hat{g}^{ab f'_{i,j}}\right)^{r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab} \left(\hat{g}^{b f'_{i,j}}\right)^{r'_{i,j,1}/b} \left(\hat{g}^{ab\omega'_{i,1}}\right)^{\phi'_i r'_{i,j,2}/b} \left(\hat{g}^{ab\omega'_{i,2}}\right)^{-f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b} \\
=& \hat{g}^{\gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}}.
\end{aligned}$$

(d) For each node $v_{i,j}$ such that $(i \in I_3) \wedge (v_{i,j} \in \overline{PV}_i)$, it proceeds as follows: It defines $TRF_i(v_{i,j}) = (1+1/a)f'_{i,j}$ by using a fixed random $f'_{i,j}$ for $v_{i,j}$. It chooses random exponents $r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$ and implicitly defines the randomness as

$$\begin{aligned}
&\gamma_i := \textstyle\sum_{i \in I_1} (\Delta_i/ab - \gamma'_i) - \sum_{i \in I_2} \gamma'_i,\ r_{i,j,1} := r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab, \\
&r_{i,j,2} := \phi'_i r'_{i,j,2}/b,\ r_{i,j,3} := -f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b - \textstyle\sum_{i \in I_1} \Delta_i/(\phi'_i ab).
\end{aligned}$$

Next, it creates a node key by using the defined randomness as

$$\begin{aligned}
K_{i,j,0} =& \hat{g}^{-\sum_{i \in I_1} \gamma'_i - \sum_{i \in I_2} \gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} \\
& (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}} \hat{g}^{-\omega'_{i,2} \sum_{i \in I_1} \Delta_i/\phi'_i}, \\
K_{i,j,1} =& (\hat{g}^a)^{r'_{1,j}} \hat{g}^{\phi'_i r'_{i,j,3}},\ K_{i,j,2} = (\hat{g}^a)^{\phi'_i r'_{i,j,2}},\ K_{i,j,3} = (\hat{g}^b)^{-f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\rho'_i r'_{i,j,2}} \hat{g}^{-\sum_{i \in I_1} \Delta_i/\phi'_i}.
\end{aligned}$$

This node key is correctly distributed by the following equation

$$\begin{aligned}
K_{i,j,0} =& \hat{g}^{\sum_{i \in I_1} (\Delta_i/ab - \gamma'_i) - \sum_{i \in I_2} \gamma'_i} \left(\hat{g}^{(ab+b) f'_{i,j}}\right)^{r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab} \left(\hat{g}^{ab\omega'_{i,1} + \rho'_i}\right)^{\phi'_i r'_{i,j,2}/b} \\
& \left(\hat{g}^{ab\omega'_{i,2} + \phi'_i}\right)^{-f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b - \sum_{i \in I_1} \Delta_i/(\phi'_i ab)} \\
=& \hat{g}^{-\sum_{i \in I_1} \gamma'_i - \sum_{i \in I_2} \gamma'_i} \left(\hat{g}^{ab f'_{i,j}}\right)^{r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab} \left(\hat{g}^{b f'_{i,j}}\right)^{r'_{i,j,1}/b} \left(\hat{g}^{ab\omega'_{i,1}}\right)^{\phi'_i r'_{i,j,2}/b} \\
& \left(\hat{g}^{ab\omega'_{i,2}}\right)^{-f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b - \sum_{i \in I_1} \Delta_i/(\phi'_i ab)} \\
=& \hat{g}^{-\sum_{i \in I_1} \gamma'_i - \sum_{i \in I_2} \gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} \\
& (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}} \hat{g}^{-\omega'_{i,2} \sum_{i \in I_1} \Delta_i/\phi'_i}.
\end{aligned}$$

(e) Next, it creates a client function key $SK_i = (CV_i, NK_{i,1}, \ldots, NK_{i,m_i})$.

4. Finally, it generates a function key $SK_Y = (SK_1, \ldots, SK_n)$.

If this is a ciphertext query for a client index i, a message x_i, and a time period $T \neq T^*$, then $\mathcal{B}$ retrieves $(T, H(T) = (g^{ab})^{h'}, h')$ from the H-list and generates a ciphertext as follows: It obtains $PV_i = (v_{i,0}, \ldots, v_{i,\ell})$ by running $\mathbf{Path}(\mathcal{BT}, x_i)$. For each $v_{i,j} \in PV_i$, it performs:

- If $v_{i,j} \in PV^*_{\mu,i}$, it creates $C_{i,j,1} = (g^{ab})^{h' f'_{i,j}}$ since $TRF_i(v_{i,j}) := f'_{i,j}$. Otherwise ($v_{i,j} \notin PV^*_{\mu,i}$), it creates $C_{i,j,1} = (g^{ab} g^b)^{h' f'_{i,j}}$ since $TRF_i(v_{i,j}) := (1 + 1/a) f'_{i,j}$.

Next, it generates a ciphertext as $\left(\{C_{i,j,1}\}_{j=0}^{\ell}, C_{i,2} = (g^{ab})^{h' \omega'_{i,1}} g^{h' \rho'_i}, C_{i,3} = (g^{ab})^{h' \omega'_{i,2}} g^{h' \phi'_i}\right)$.

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

Lemma 4.4. *If the A3DH assumption holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_2$ and $\mathbf{G}_3$ with a non-negligible advantage.*

Proof. We define a sequence of hybrid games $\mathbf{H}_{2,(1,0)}, \mathbf{H}_{2,(1,1)}, \ldots, \mathbf{H}_{2,(1,\ell)}, \mathbf{H}_{2,(2,1)}, \ldots, \mathbf{H}_{2,(i_\delta, j_\delta)}, \ldots, \mathbf{H}_{2,(n,\ell)}$ where $\mathbf{H}_{2,(1,0)} = \mathbf{G}_2$, $(i_\delta, j_\delta) \in [n, \ell]$. For notational simplicity, we also define $\mathbf{H}_{2,(i_\delta - 1, \ell)} = \mathbf{H}_{2,(i_\delta, 0)}$. Recall that two node sets E_i and $\overline{E}_i$ were defined for each client index $i \in [n]$ as $E_i = \{v^*_{\mu,i,j} : (v^*_{\mu,i,j} \in PV^*_{\mu,i}) \wedge (v^*_{0,i,j} = v^*_{1,i,j})\}$ and $\overline{E}_i = \{v^*_{\mu,i,j} : (v^*_{\mu,i,j} \in PV^*_{\mu,i}) \wedge (v^*_{0,i,j} \neq v^*_{1,i,j})\}$. The game $\mathbf{H}_{2,(i_\delta, j_\delta)}$ is defined as follows:

Game $\mathbf{H}_{2,(i_\delta, j_\delta)}$ In this game $\mathbf{H}_{2,(i_\delta, j_\delta)}$, we slightly change the generation of challenge ciphertext elements $\{C_{i,j,1}\}$. For each client i and node $v_{i,j} \in PV^*_{\mu,i}$, a simulator creates the element $C_{i,j,1}$ as follows:

- If $v_{i,j} \in E_i$, then it creates $C_{i,j,1}$ normally.
- If $v_{i,j} \in \overline{E}_i$ and $(i,j) \leq (i_\delta, j_\delta)$, then it creates $C_{i,j,1}$ as a random element in $\mathbb{G}$.
- If $v_{i,j} \in \overline{E}_i$ and $(i,j) > (i_\delta, j_\delta)$, then it creates $C_{i,j,1}$ normally.

The simulator creates challenge ciphertext elements $C_{i,2}, C_{i,3}$, as the same way as the game $\mathbf{G}_3$. It is obvious that $\mathbf{H}_{2,(n,\ell)} = \mathbf{G}_3$.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H}_{2,(i_\delta, j_\delta - 1)}$ and $\mathbf{H}_{2,(i_\delta, j_\delta)}$ with a non-negligible advantage. Without loss of generality, we assume that $v_{i_\delta, j_\delta} \in \overline{E}_{i_\delta}$ since $\mathbf{H}_{2,(i_\delta, j_\delta - 1)} = \mathbf{H}_{2,(i_\delta, j_\delta)}$ if $v_{i_\delta, j_\delta} \in E_{i_\delta}$. A simulator $\mathcal{B}$ that solves the A3DH assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, g^a, g^b, g^{ab}, g^{abc}, \hat{g}^a, \hat{g}^b)$ and Z where $Z = g^c$ or $Z = R$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows.

Init: $\mathcal{A}$ submits challenge message lists $X^*_0 = (x^*_{0,1}, \ldots, x^*_{0,n})$, $X^*_1 = (x^*_{1,1}, \ldots, x^*_{1,n})$, and a challenge time period T^*. $\mathcal{B}$ flips a random coin $\mu \in \{0,1\}$ to fix X^*_μ as the target message list and obtains $PV^*_{\mu,i} = (v^*_{\mu,i,0}, \ldots, v^*_{\mu,i,\ell})$ by running $\mathbf{Path}(\mathcal{BT}, x^*_{\mu,i})$ for all $i \in [n]$.

Setup: $\mathcal{B}$ proceeds as follows:

1. It first selects random exponents $\{\omega'_{i,1}, \omega'_{i,2}, \rho'_i, \phi'_i\}_{i=1}^n \in \mathbb{Z}_p$ and defines $\{\omega_{i,1} := \omega'_{i,1} + (1/ab)\rho'_i, \omega_{i,2} := \omega'_{i,2} + (1/ab)\phi'_i\}_{i=1}^n$. Next, it implicitly sets $\hat{v} = \hat{g}^{ab}, \{\hat{w}_{i,1} = (\hat{g}^{ab})^{\omega'_{i,1}} \hat{g}^{\rho'_i}, \hat{w}_{i,2} = (\hat{g}^{ab})^{\omega'_{i,2}} \hat{g}^{\phi'_i}\}_{i=1}^n$.
2. It now defines the value of TRF by using lazy sampling. Let $v_{i,j}$ be a node for a client i and $PV^*_{\mu,i}$ be the challenge path set for the same client i.
   - **Case** $i \neq i_\delta$ or $v_{i,j} \neq v_{i_\delta, j_\delta}$: If $v_{i,j} \in PV^*_{\mu,i}$, then it defines $TRF_i(v_{i,j}) := f'_{i,j}$ by selecting a fixed random $f'_{i,j}$ for $v_{i,j}$. Otherwise ($v_{i,j} \notin PV^*_{\mu,i}$), it defines $TRF_i(v_{i,j}) := (1 + 1/a) f'_{i,j}$ by selecting a fixed random $f'_{i,j}$ for $v_{i,j}$. That is, it implicitly defines

$$\hat{v}^{TRF_i(v_{i,j})} := (\hat{g}^{ab})^{f'_{i,j}} \text{ if } v_{i,j} \in PV^*_{\mu,i}, \quad \hat{v}^{TRF_i(v_{i,j})} := (\hat{g}^{ab} \hat{g}^b)^{f'_{i,j}} \text{ if } v_{i,j} \notin PV^*_{\mu,i}.$$

- **Case** $i = i_\delta$ and $v_{i,j} = v_{i_\delta,j_\delta}$: It defines $TRF_i(v_{i,j}) := (1/ab)f'_{i,j}$ since $v_{i,j} = v_{i_\delta,j_\delta} \in PV^*_{\mu,i}$ by selecting a fixed random $f'_{i,j}$ for $v_{i,j}$. That is, it implicitly defines

$$\hat{v}^{TRF_i(v_{i,j})} := (\hat{g})^{f'_{i,j}} \text{ since } v_{i,j} \in PV^*_{\mu,i}.$$

3. For the target message list $X^*_\mu = (x^*_{\mu,1}, \ldots, x^*_{\mu,n})$ with $PV^*_{\mu,i} = (v^*_{\mu,i,0}, \ldots, v^*_{\mu,i,\ell})$ for all i, it selects a random $f'_{i,j}$ and defines $TRF_i(v^*_{\mu,i,j}) := (1/ab)f'_{i,j}$ if $(i,j) = (i_\delta, j_\delta)$ and $TRF_i(v^*_{\mu,i,j}) := f'_{i,j}$ if $(i,j) \neq (i_\delta, j_\delta)$.

4. It initializes a hash table H-list and publishes $PP = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, n, \ell, H)$.

Challenge: $\mathcal{B}$ first sets $H(T^*) = g^{abc}$ and creates each challenge ciphertext element $C_{i,j,1}$ for each client $i \in [n]$ and node $v_{i,j} \in PV^*_{\mu,i}$ as follows:

- **Case** $(i,j) < (i_\delta, j_\delta)$: If $v_{i,j} \in E_i$, then it creates $C_{i,j,1} = (g^{abc})^{f'_{i,j}}$ since $TRF_i(v^*_{\mu,i,j}) := f'_{i,j}$. Otherwise ($v_{i,j} \in \overline{E}_i$), it chooses a random element $P_{i,j,1} \in \mathbb{G}$ and sets $C_{i,j,1} = P_{i,j,1}$.

- **Case** $(i,j) = (i_\delta, j_\delta)$: It creates $C_{i,j,1} = Z^{f'_{i,j}}$ since $v_{i_\delta,j_\delta} \in \overline{E}_i$ and $TRF_i(v^*_{\mu,i,j}) := (1/ab)f'_{i,j}$.

- **Case** $(i,j) > (i_\delta, j_\delta)$: It creates $C_{i,j,1} = (g^{abc})^{f'_{i,j}}$ since $TRF_i(v^*_{\mu,i,j}) := f'_{i,j}$.

Next, it chooses a random element $P \in \mathbb{G}$ and generates challenge ciphertexts for the time period T^* as

$$\left(\left\{ C_{i,j,1} \right\}_{j=0}^{\ell},\ C_{i,2} = (g^{abc})^{\omega'_{i,1}} P^{\rho'_i},\ C_{i,3} = (g^{abc})^{\omega'_{i,2}} P^{\phi'_i} \right) \text{ for all } i \in [n].$$

If $Z = g^c$, then $\mathcal{B}$ plays $\mathbf{H}_{3,(i_\delta, j_\delta - 1)}$. Otherwise, it plays $\mathbf{H}_{3,(i_\delta, j_\delta)}$.

Query: $\mathcal{A}$ adaptively requests hash, function key, and ciphertext queries. $\mathcal{B}$ handles these queries as follows. If this is a hash query for a time period T, then $\mathcal{B}$ proceeds as follows: If T was queried before, then it retrieves $(T, h, -)$ from H-list and returns h. Otherwise, it performs:

- **Case** $T = T^*$: It sets $H(T^*) = g^{abc}$ and stores $(T^*, H(T^*), -)$ to H-list. It then returns g^{abc}.

- **Case** $T \neq T^*$: It selects a random exponent $h' \in \mathbb{Z}_p$ and stores $(T, H(T) = (g^{ab})^{h'}, h')$ to H-list. It then returns $(g^{ab})^{h'}$.

If this is a function key query for a list of vectors $Y = (y_1, \ldots, y_n)$ where $y_i = (y_{i,L}, y_{i,R})$, then $\mathcal{B}$ generates a function key as follows:

1. It first derives cover sets $CV_1, \ldots, CV_n$ from $y_1, \ldots, y_n$. For each CV_i, it defines two node sets PV_i and $\overline{PV}_i$ as $PV_i = \{v_{i,j} : v_{i,j} \in CV_i \wedge v_{i,j} \in PV^*_{\mu,i}\}$ and $\overline{PV}_i = \{v_{i,j} : v_{i,j} \in CV_i \wedge v_{i,j} \notin PV^*_{\mu,i}\}$.

2. It also defines three index sets $I_1, I_2, I_3 \subseteq [n]$ which are partitions of client indexes. It defines $I_1 = \{i : \exists v_{i,j} \in CV_i \text{ such that } v_{i,j} \in PV_i\}$. It next selects a random index $\tilde{i} \in [n] \setminus I_1$ and defines $I_3 = \{\tilde{i}\}$. It then defines $I_2 = [n] \setminus (I_1 \cup I_3)$. By using these sets, it will define γ_i differently depending on the index sets. For each client index $i \in I_2$, it chooses a random exponent γ'_i and implicitly defines $\gamma_i := \gamma'_i$.

3. For each client $i \in [n]$, it performs the following steps:

   (a) For each node $v_{i,j}$ such that $(i \in I_1) \wedge (v_{i,j} \in PV_i)$, it proceeds as follows:

- Case $i \neq i_\delta$ or $v_{i,j} \neq v_{i_\delta,j_\delta}$: It defines $TRF_i(v_{i,j}) := f'_{i,j}$ by retrieving $f'_{i,j}$ chosen at the setup. It chooses random exponents $\gamma'_i, r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$, sets $\Delta_i = \rho'_i r'_{i,j,2} + \phi'_i r'_{i,j,3}$, and implicitly defines the randomness as

$$\gamma_i := -\Delta_i/ab + \gamma'_i,\ r_{i,j,1} := r'_{i,j,1}/ab,\ r_{i,j,2} := r'_{i,j,2}/ab,\ r_{i,j,3} := r'_{i,j,3}/ab.$$

  Note that this γ_i is fixed for this index i since there is only one tuple $(i, v_{i,j}) \in PV$ for the index i. Next, it creates a node key by using the defined randomness as

$$K_{i,j,0} = \hat{g}^{\gamma'_i} \hat{g}^{f'_{i,j} r'_{i,j,1}} \hat{g}^{\omega'_{i,1} r'_{i,j,2}} \hat{g}^{\omega'_{i,2} r'_{i,j,3}},\ K_{i,j,1} = \hat{g}^{r'_{i,j,1}},\ K_{i,j,2} = \hat{g}^{r'_{i,j,2}},\ K_{i,j,3} = \hat{g}^{r'_{i,j,3}}.$$

  This node key is correctly distributed by the following equation

$$\begin{aligned} K_{i,j,0} &= \hat{g}^{-(\rho'_i r'_{i,j,2} + \phi'_i r'_{i,j,3})/ab + \gamma'_i} \left(\hat{g}^{ab f'_{i,j}}\right)^{r'_{i,j,1}/ab} \left(\hat{g}^{ab\omega'_{i,1} + \rho'_i}\right)^{r'_{i,j,2}/ab} \left(\hat{g}^{ab\omega'_{i,2} + \phi'_i}\right)^{r'_{i,j,3}/ab} \\ &= \hat{g}^{\gamma'_i} \hat{g}^{f'_{i,j} r'_{i,j,1}} \hat{g}^{\omega'_{i,1} r'_{i,j,2}} \hat{g}^{\omega'_{i,2} r'_{i,j,3}}. \end{aligned}$$

- Case $i = i_\delta$ and $v_{i,j} = v_{i_\delta,j_\delta}$: It defines $TRF_i(v_{i,j}) := (1/ab) f'_{i,j}$ by using $f'_{i,j}$ chosen at the setup. It chooses random exponents $\gamma'_i, r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$, sets $\Delta_i = f'_{i,j} r'_{i,j,1} + \rho'_i r'_{i,j,2} + \phi'_i r'_{i,j,3}$, and implicitly defines the randomness as

$$\gamma_i := -\Delta_i/ab + \gamma'_i,\ r_{i,j,1} := r'_{i,j,1}/ab,\ r_{i,j,2} := r'_{i,j,2}/ab,\ r_{i,j,3} := r'_{i,j,3}/ab.$$

  Note that this γ_i is fixed for this index i since there is only one tuple $(i, v_{i,j}) \in PV$ for the index i. Next, it creates a node key by using the defined randomness as

$$K_{i,j,0} = \hat{g}^{\gamma'_i} \hat{g}^{\omega'_{i,1} r'_{i,j,2}} \hat{g}^{\omega'_{i,2} r'_{i,j,3}},\ K_{i,j,1} = \hat{g}^{r'_{i,j,1}},\ K_{i,j,2} = \hat{g}^{r'_{i,j,2}},\ K_{i,j,3} = \hat{g}^{r'_{i,j,3}}.$$

  This node key is correctly distributed by the following equation

$$\begin{aligned} K_{i,j,0} &= \hat{g}^{-(f'_{i,j} r'_{i,j,1} + \rho'_i r'_{i,j,2} + \phi'_i r'_{i,j,3})/ab + \gamma'_i} \left(\hat{g}^{f'_{i,j}}\right)^{r'_{i,j,1}/ab} \left(\hat{g}^{ab\omega'_{i,1} + \rho'_i}\right)^{r'_{i,j,2}/ab} \left(\hat{g}^{ab\omega'_{i,2} + \phi'_i}\right)^{r'_{i,j,3}/ab} \\ &= \hat{g}^{\gamma'_i} \hat{g}^{\omega'_{i,1} r'_{i,j,2}} \hat{g}^{\omega'_{i,2} r'_{i,j,3}}. \end{aligned}$$

(b) For each node $v_{i,j}$ such that $(i \in I_1) \wedge (v_{i,j} \in \overline{PV}_i)$, it proceeds as follows: It defines $TRF_i(v_{i,j}) := (1 + 1/a) f'_{i,j}$ by retrieving $f'_{i,j}$ chosen at the setup. Recall that the random γ_i was already defined as $\gamma_i := -\Delta_i/ab + \gamma'_i$ for some node $v_{i,j'}$. It chooses random exponents $r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$ and implicitly defines the randomness as

$$\begin{aligned} r_{i,j,1} &:= r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab,\ r_{i,j,2} := \phi'_i r'_{i,j,2}/b, \\ r_{i,j,3} &:= -f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b + \Delta_i/(\phi'_i ab). \end{aligned}$$

  Next, it creates a node key by using the defined randomness as

$$\begin{aligned} K_{i,j,0} &= \hat{g}^{\gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}} \hat{g}^{\omega'_{i,2} \Delta_i/\phi'_i}, \\ K_{i,j,1} &= (\hat{g}^a)^{r'_{i,j,1}} \hat{g}^{\phi'_i r'_{i,j,3}},\ K_{i,j,2} = (\hat{g}^a)^{\phi'_i r'_{i,j,2}},\ K_{i,j,3} = (\hat{g}^b)^{-f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\rho'_i r'_{i,j,2}} \hat{g}^{\Delta_i/\phi'_i}. \end{aligned}$$

  This node key is correctly distributed since it is almost the same as that of Lemma 4.3.

(c) For each node $v_{i,j}$ such that $(i \in I_2) \wedge (v_{i,j} \in \overline{PV}_i)$, it proceeds as follows: It defines $TRF_i(v_{i,j}) := (1+1/a)f'_{i,j}$ by using a fixed random $f'_{i,j}$ for $v_{i,j}$. Recall that γ_i was already defined as $\gamma_i := \gamma'_i$ for the set I_2. It chooses random exponents $r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$ and implicitly defines the randomness as

$$r_{i,j,1} := r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab,\ r_{i,j,2} := \phi'_i r'_{i,j,2}/b,\ r_{i,j,3} := -f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b.$$

Next, it creates a node key by using the defined randomness as

$$K_{i,j,0} = \hat{g}^{\gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}},$$
$$K_{i,j,1} = (\hat{g}^a)^{r'_{i,j,1}} \hat{g}^{\phi'_i r'_{i,j,3}},\ K_{i,j,2} = (\hat{g}^a)^{\phi'_i r'_{i,j,2}},\ K_{i,j,3} = (\hat{g}^b)^{-f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\rho'_i r'_{i,j,2}}.$$

This node key is correctly distributed since it is almost the same as that of Lemma 4.3.

(d) For each node $v_{i,j}$ such that $(i \in I_3) \wedge (v_{i,j} \in \overline{PV}_i)$, it proceeds as follows: It defines $TRF_i(v_{i,j}) = (1+1/a)f'_{i,j}$ by using a fixed random $f'_{i,j}$ for $v_{i,j}$. It chooses random exponents $r'_{i,j,1}, r'_{i,j,2}, r'_{i,j,3} \in \mathbb{Z}_p$ and implicitly defines the randomness as

$$\gamma_i := \textstyle\sum_{i \in I_1} (\Delta_i/ab - \gamma'_i) - \sum_{i \in I_2} \gamma'_i,\ r_{i,j,1} := r'_{i,j,1}/b + \phi'_i r'_{i,j,3}/ab,$$
$$r_{i,j,2} := \phi'_i r'_{i,j,2}/b,\ r_{i,j,3} := -f'_{i,j} r'_{i,j,3}/a - \rho'_i r'_{i,j,2}/b - \textstyle\sum_{i \in I_1} \Delta_i/(\phi'_i ab).$$

Next, it creates a node key by using the defined randomness as

$$\begin{aligned} K_{i,j,0} =& \hat{g}^{-\sum_{i \in I_1} \gamma'_i - \sum_{i \in I_2} \gamma'_i} (\hat{g}^a)^{f'_{i,j} r'_{i,j,1}} \hat{g}^{f'_{i,j} \phi'_i r'_{i,j,3}} \hat{g}^{f'_{i,j} r'_{i,j,1}} (\hat{g}^a)^{\omega'_{i,1} \phi'_i r'_{i,j,2}} \\ & (\hat{g}^b)^{-\omega'_{i,2} f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\omega'_{i,2} \rho'_i r'_{i,j,2}} \hat{g}^{-\omega'_{i,2} \sum_{i \in I_1} \Delta_i/\phi'_i}, \\ K_{i,j,1} =& (\hat{g}^a)^{r'_{1,j}} \hat{g}^{\phi'_i r'_{i,j,3}},\ K_{i,j,2} = (\hat{g}^a)^{\phi'_i r'_{i,j,2}},\ K_{i,j,3} = (\hat{g}^b)^{-f'_{i,j} r'_{i,j,3}} (\hat{g}^a)^{-\rho'_i r'_{i,j,2}} \hat{g}^{-\sum_{i \in I_1} \Delta_i/\phi'_i}. \end{aligned}$$

This node key is correctly distributed since it is the same as that of Lemma 4.3.

(e) Next, it creates a client function key $SK_i = (CV_i, NK_{i,1}, \ldots, NK_{i,m_i})$.

4. Finally, it generates a function key $SK_Y = (SK_1, \ldots, SK_n)$.

If this is a ciphertext query for a client index i, a message vector $\vec{x}_i = (x_{i,1}, \ldots, x_{i,\ell})$, and a time period $T \neq T^*$, then $\mathcal{B}$ retrieves $(T, H(T) = (g^{ab})^{h'}, h')$ from the H-list and generates a ciphertext as follows: It obtains $PV_i = (v_{i,0}, \ldots, v_{i,\ell})$ by running $\mathbf{Path}(\mathcal{BT}, x_i)$. For each $v_{i,j} \in PV_i$, it performs:

- **Case** $i \neq i_\delta$ or $v_{i,j} \neq v_{i_\delta,j_\delta}$: If $v_{i,j} \in PV^*_{\mu,i}$, it creates $C_{i,j,1} = (g^{ab})^{h' f'_{i,j}}$ since $TRF_i(v_{i,j}) := f'_{i,j}$. Otherwise ($v_{i,j} \notin PV^*_{\mu,i}$), it creates $C_{i,j,1} = (g^{ab} g^b)^{h' f'_{i,j}}$ since $TRF_i(v_{i,j}) := (1+1/a)f'_{i,j}$.
- **Case** $i = i_\delta$ and $v_{i,j} = v_{i_\delta,j_\delta}$: It creates $C_{i,j,1} = g^{h' f'_{i,j}}$ since $TRF_i(v_{i,j}) := (1/ab)f'_{i,j}$.

Next, it generates a ciphertext as $\left(\left\{C_{i,j,1}\right\}_{j=0}^{\ell}, C_{i,2} = (g^{ab})^{h' \omega'_{i,1}} g^{h' \rho'_i}, C_{i,3} = (g^{ab})^{h' \omega'_{i,2}} g^{h' \phi'_i}\right)$.

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

Theorem 4.5. *The above MC-RQE scheme is selectively single-challenge weak IND-secure with static corruptions in the random oracle model if the MC-RQE scheme is selectively single-challenge weak IND-secure with no corruptions.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that breaks the selective single-challenge weak IND-security with static corruptions. By using $\mathcal{A}$, a simulator $\mathcal{B}$ try to break the selective single-challenge weak IND-security with no corruptions played by a challenger $\mathcal{C}$. The simulator $\mathcal{B}$ is described as follows:

Init: $\mathcal{A}$ submits the set of corrupted client indexes $\bar{I}$, two challenge message lists X_0^*, X_1^*, and a challenge time period T^*. Let $I = \{1,\ldots,n\} \setminus \bar{I}$ be the set of uncorrupted client indexes. $\mathcal{B}$ submits two challenge message lists X_0^*, X_1^*, the challenge time period T^* to $\mathcal{C}$. Note that $\mathcal{C}$ plays the selective security game with no corruptions for the set I.

Setup: $\mathcal{B}$ receives PP from $\mathcal{C}$. It chooses random PRF keys $\{z_i\}_{i\in\bar{I}}$ and random exponents $\{\omega_{i,1}, \omega_{i,2}\}_{i\in\bar{I}}$. Next, it gives $\{EK_i = (z_i, \omega_{i,1}, \omega_{i,2})\}_{i\in\bar{I}}$ and PP to $\mathcal{A}$.

Challenge: $\mathcal{B}$ receives challenge ciphertexts $\{CT_{i,T^*}\}_{i\in I}$ from $\mathcal{C}$ and gives $\{CT_{i,T^*}\}_{i\in I}$ to $\mathcal{A}$.

Query: $\mathcal{A}$ adaptively requests hash, function key, and ciphertext queries. $\mathcal{B}$ handles these queries as follows: If this is a hash query for a time period T, then $\mathcal{B}$ relays this query to $\mathcal{C}$ and gives the response of $\mathcal{C}$ to $\mathcal{A}$. If this is a function key query for a list of vectors $Y = (y_1,\ldots,y_n)$, then $\mathcal{B}$ proceeds as follows:

1. It first sets $Y' = \{y_i\}_{i\in I}$ derived from Y. It requests a function key for Y' to $\mathcal{C}$ and receives a function key $SK_{Y'} = \{SK'_i = (NK'_{i,1},\ldots,NK'_{i,2\ell})\}_{i\in I}$ where $NK'_{i,j} = (K'_{i,j,0},\ldots,K'_{i,j,3})$. It selects a random exponent $r \in \mathbb{Z}_p$ and sets $\tilde{v} = \hat{g}^r$, which is formed as $\tilde{v} = \hat{v}^{r'}$ for some unknown random r'.
2. It selects random exponents $\gamma_1,\ldots,\gamma_{n-1} \in \mathbb{Z}_p$ and sets $\gamma_n = -\sum_{i=1}^{n-1}\gamma_i$ to satisfy $\sum_{i=1}^{n}\gamma_i = 0$.
3. For each uncorrupted client $i \in I$, it performs the following steps:
   (a) Let $SK'_i = (NK'_{i,1},\ldots,NK'_{i,2\ell})\}_{i\in I}$ where $NK'_{i,j} = (K'_{i,j,0},\ldots,K'_{i,j,3})$.
   (b) For each $j \in [2\ell]$, it modifies a node key as $NK_{i,j} = (K_{i,j,0} = K'_{i,j,0}\cdot\hat{g}^{\gamma_i}, K_{i,j,1} = K'_{i,j,1}, K_{i,j,2} = K'_{i,j,2}, K_{i,j,3} = K'_{i,j,3})$ by using the random γ_i.
   (c) It sets $SK_i = \{NK_{i,1},\ldots,NK_{i,2\ell}\}$.
4. For each corrupted client $i \in \bar{I}$, it performs the following steps:
   (a) It obtains a cover set $CV_i = (v_{i,1},\ldots,v_{i,m_i})$ by running the **Cover** algorithm for the range $y_i = (y_{i,L}, y_{i,R})$.
   (b) For each node $v_{i,j} \in CV_i$, it calculates $f_{i,j} = PRF(z_i, v_{i,j})$ and compute a node key by selecting random exponents $r_{i,j,1}, r_{i,j,2}, r_{i,j,3}$ as
   $$NK_{i,j} = \left(K_{i,j,0} = \hat{g}^{\gamma_i}\tilde{v}^{f_{i,j}\cdot r_{i,j,1}}\tilde{v}^{\omega_{i,1}r_{i,j,2}}\tilde{v}^{\omega_{i,2}r_{i,j,3}}, K_{i,j,1} = \tilde{v}^{r_{i,j,1}}, K_{i,j,2} = \tilde{v}^{r_{i,j,2}}, K_{i,j,3} = \tilde{v}^{r_{i,j,3}}\right)$$
   (c) Next, it creates a client function key $SK_i = (CV_i, NK_{i,1},\ldots,NK_{i,m_i})$.
5. It gives a function key $SK_Y = (\{SK_i\}_{i\in I}, \{SK_i\}_{i\in\bar{I}})$ to $\mathcal{A}$.

If this is a ciphertext query for a client index $i \in I$, a message x_i, and a time period $T \neq T^*$, then $\mathcal{B}$ relays this query to $\mathcal{C}$ and gives $CT_{i,T}$ from $\mathcal{C}$ to $\mathcal{A}$.

Guess: $\mathcal{A}$ outputs a guess μ'. $\mathcal{B}$ also outputs μ'. □

Table 2: Comparison of MC-RQE and MRQED schemes

Scheme	PP Size	SK Size	CT Size	Encrypt	Decrypt	Math
MRQED [37]	$O(n\ell)$	$O(n\ell)$	$O(n\ell)$	$O(n\ell)$	$O(\ell^n)$	BLM
MRQED [20]	$O(n\ell)$	$O(n\ell)$	$O(n\ell)$	$O(n\ell)$	$O(\ell^n)$	Lattice
Ours	$O(1)$	$O(n\ell)$	$O(n\ell)$	$O(\ell)$	$O(n2^n)$	BLM

Let n be the number of clients (or dimensions in MRQED) and ℓ be the depth of a binary tree. We use BLM for bilinear maps.

4.7 Discussions

Efficiency Analysis. In our MC-RQE scheme, a ciphertext consists of $\ell+2$ group elements per each client and $n(\ell+2)$ group elements for n clients where ℓ is the depth of a binary tree. Since a function key is associated with 2ℓ tree nodes per each client and each node requires 4 group elements, a function key for n clients consists of $8n\ell$ group elements. The encryption algorithm requires $O(n\ell)$ exponentiation operations, and the function key generation algorithm also requires $O(n\ell)$ exponentiation operations. The decryption algorithm is the slowest algorithm, and it checks whether the pairing expression is satisfied for each combination of function key elements and ciphertext elements. In this case, the number of possible combinations is at most 2^n since the node depth of function keys is known and $3n+1$ pairing operations are required for each combination. Thus, the decryption algorithm requires $O(n2^n)$ pairing operations. The detailed performance is given in Table 2.

Allowing Matching Function Keys. We proved that our MC-RQE scheme is secure in a weak security model where matching function keys ($f(X_0^*) = f(X_1^*) = 1$) are not allowed. In fact, we can prove that our MC-RQE scheme is secure even in a security model where one matching function key is allowed by using the fact that the underlying MC-HVE scheme allows many matching function keys. However, it is not easy to prove the security of our MC-RQE scheme in a strong security model in which two or more matching function keys are allowed. The fundamental reason of the difficulty is that the matching node of a binary tree that succeeds in decryption is additionally exposed during the decryption process of the MC-RQE scheme. That is, for two challenge messages X_0^* and X_1^*, an attacker first prepares two matching function keys in which the tree node of successful decryption is the same for X_0^*, and the tree node for X_1^* is changed. Then the attacker decrypts a challenge ciphertext by using these matching function keys. At this time, the attacker can distinguish the challenge ciphertext by checking whether the tree node that is successfully decrypted is the same or changed.

Comparison with MRQED Scheme. Our MC-RQE scheme can be compared with the MRQED scheme of Shi et al. [37] in terms of effectively supporting range queries on encrypted data. As described above, our MC-RQE scheme uses the path set and cover set of a binary tree to handle range queries in the same way as the MRQED scheme. However, the MRQED scheme is a public-key cryptosystem that combines an anonymous IBE scheme with a binary tree, and it handles a single ciphertext in the decryption process. In contrast, our MC-HVE scheme is a private-key cryptosystem that combines our MC-HVE scheme and a binary tree, and it can handle multiple ciphertexts in the decryption process. The detailed comparison is given in Table 2.

5 Conclusion

In this paper, we presented efficient MC-FE schemes that support conjunctive equality or range queries on encrypted data in bilinear groups. Our MC-HVE scheme is a construction that efficiently supports conjunctive equality queries with wildcards for ciphertexts generated by multiple clients at the same time period. Our MC-RQE scheme uses binary trees to efficiently support conjunctive range queries while the size of a ciphertext and a function key is very compact. We also proved that our MC-HVE and MC-RQE schemes are selectively secure with static corruptions under static assumptions.

This work leaves some interesting problems: The first problem is to devise an efficient MC-FE scheme that supports more expressive predicate queries than conjunctive equality or range queries. As an example, extending an IPE scheme to the multi-client setting can be interesting. The second problem is to construct an MC-HVE scheme that supports not only the message hiding security but also the function hiding security. The third problem is to devise an efficient MC-RQE scheme that provides the strong security.

References

[1] Michel Abdalla, Fabrice Benhamouda, and Romain Gay. From single-input to multi-client inner-product functional encryption. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019*, volume 11923 of *Lecture Notes in Computer Science*, pages 552–582. Springer, 2019.

[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, 2017.

[4] Shweta Agrawal, Rishab Goyal, and Junichi Tomida. Multi-input quadratic functional encryption from pairings. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021*, volume 12828 of *Lecture Notes in Computer Science*, pages 208–238. Springer, 2021.

[5] 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.

[6] Shweta Agrawal, Monosij Maitra, Narasimha Sai Vempati, and Shota Yamada. Functional encryption for Turing machines with dynamic bounded collusion from LWE. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021*, volume 12828 of *Lecture Notes in Computer Science*, pages 239–269. Springer, 2021.

[7] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017*, volume 10401 of *Lecture Notes in Computer Science*, pages 67–98. Springer, 2017.

[8] 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.

[9] Carlo Blundo, Vincenzo Iovino, and Giuseppe Persiano. Private-key hidden vector encryption with key confidentiality. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *Cryptology and Network Security - CANS 2009*, volume 5888 of *Lecture Notes in Computer Science*, pages 259–277. Springer, 2009.

[10] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[11] 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.

[12] 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.

[13] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: A new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.

[14] 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.

[15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM Conference on Computer and Communications Security*, pages 668–679. ACM, 2015.

[16] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 353–373. Springer, 2013.

[17] 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.

[18] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security - CCS 2006*, pages 79–88. ACM, 2006.

[19] 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.

[20] Romain Gay, Pierrick Méaux, and Hoeteck Wee. Predicate encryption for multi-dimensional range queries from lattices. In Jonathan Katz, editor, *Public-Key Cryptography - PKC 2015*, volume 9020 of *Lecture Notes in Computer Science*, pages 752–776. Springer, 2015.

[21] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public-Key Cryptography - PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2006.

[22] 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.

[23] Philippe Golle, Jessica Staddon, and Brent R. Waters. Secure conjunctive keyword search over encrypted data. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *Applied Cryptography and Network Security - ACNS 2004*, volume 3089 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004.

[24] 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.

[25] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, 2015.

[26] 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.

[27] 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.

[28] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, and Cong Zuo. Result pattern hiding searchable encryption for conjunctive queries. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM Conference on Computer and Communications Security - CCS 2018*, pages 745–762. ACM, 2018.

[29] Kwangsu Lee and Dong Hoon Lee. Improved hidden vector encryption with short ciphertexts and tokens. *Des. Codes Cryptogr.*, 58(3):297–319, 2011.

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

[31] Yanbin Lu. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *Network and Distributed System Security Symposium - NDSS 2012*. The Internet Society, 2012.

[32] Phillip A. Porras and Vitaly Shmatikov. Large-scale collection and sanitization of network security data: risks and challenges. In Christian Hempelmann and Victor Raskin, editors, *New Security Paradigms Workshop 2006*, pages 57–64. ACM, 2006.

[33] Michael Rushanan, Aviel D. Rubin, Denis Foo Kune, and Colleen M. Swanson. SoK: Security and privacy in implantable medical devices and body area networks. In *IEEE Symposium on Security and Privacy, SP 2014*, pages 524–539. IEEE Computer Society, 2014.

[34] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security - CCS 2010*, pages 463–472. ACM, 2010.

[35] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

[36] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In Omer Reingold, editor, *Theory of Cryptography - TCC 2009*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2009.

[37] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, pages 350–364. IEEE Computer Society, 2007.

[38] Elaine Shi, T-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Network and Distributed System Security - NDSS 2011*. The Internet Society, 2011.

[39] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 560–578. Springer, 2008.

[40] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[41] Tim van de Kamp, Andreas Peter, Maarten H. Everts, and Willem Jonker. Multi-client predicate-only encryption for conjunctive equality tests. In Srdjan Capkun and Sherman S. M. Chow, editors, *Cryptology and Network Security - CANS 2017*, volume 11261 of *Lecture Notes in Computer Science*, pages 135–157. Springer, 2018.