

Multi-Client Order-Revealing Encryption

JIEUN EOM¹, DONG HOON LEE¹, (Member, IEEE), AND KWANGSU LEE²

¹Graduate School of Information Security, Korea University, Seoul 02841, South Korea

²Department of Computer and Information Security, Sejong University, Seoul 05000, South Korea

Corresponding author: Kwangsu Lee (kwangsu@sejong.ac.kr)

This work was supported by the Institute for Information & communications Technology Promotion (IITP) through the Korea Government (MSIT) under Grant 2016-6-00600-A Study on Functional Encryption: Construction, Security Analysis, and Implementation.

ABSTRACT Order-revealing encryption is a useful cryptographic primitive that provides range queries on encrypted data since anyone can compare the order of plaintexts by running a public comparison algorithm. Most studies on order-revealing encryption focus only on comparing ciphertexts generated by a single client, and there is no study on comparing ciphertexts generated by multiple clients. In this paper, we propose the concept of multi-client order-revealing encryption that supports comparisons not only on ciphertexts generated by one client but also on ciphertexts generated by multiple clients. We also define a simulation-based security model for the multi-client order-revealing encryption. The security model is defined with respect to the leakage function which quantifies how much information is leaked from the scheme. Next, we present two specific multi-client order-revealing encryption schemes with different leakage functions in bilinear maps and prove their security in the random oracle model. Finally, we give the implementation of the proposed schemes and suggest methods to improve the performance of ciphertext comparisons.

INDEX TERMS Bilinear maps, multi-client order-revealing encryption, order-revealing encryption, symmetric-key encryption.

I. INTRODUCTION

Today, a large amount of the users' data is collected and stored in cloud servers to provide various services utilizing this personal data. Recently, as the concern of privacy issues in personal data has increased, it has been an important issue to safely store personal data in a cloud server and to prevent it from being leaked. The simplest way to solve this problem is to perform data encryption. However, it is difficult for the cloud server to provide ordinary services such as keyword searches, range queries, and numeric operations on encrypted data since plaintexts are transformed to random ciphertexts. In order to overcome this problem, advanced encryption schemes that support computation on encrypted data such as homomorphic encryption and functional encryption have been actively studied [1], [2]. However, it is difficult to provide efficient services using them since these schemes are somewhat inefficient.

One way to allow efficient computation on encrypted data while providing privacy of user data is to consider an efficient encryption scheme that allows only a limited operation such as a search or range query. Searchable symmetric encryption (SSE) is a kind of symmetric-key encryption that supports keyword searching on encrypted data [3].

Order-preserving encryption (OPE) and order-revealing encryption (ORE) are special kinds of symmetric-key encryption that can be used for efficient range queries over encrypted data by comparing ciphertexts without decrypting these ciphertexts. An OPE scheme is a deterministic encryption scheme, which encrypts plaintexts in numeric values to generate ciphertexts in numerical values by maintaining the order of plaintexts, so that the order of plaintexts can be compared by simply comparing the order of ciphertexts [4]–[6]. An ORE scheme is a probabilistic encryption scheme having ciphertexts of arbitrary values, and the order of plaintexts can be compared by running a public comparison algorithm on ciphertexts [7]–[10]. The first ORE scheme of Boneh *et al.* [7] provides the best possible security, but it is inefficient since it uses heavy cryptographic tools such as multi-linear maps. Recently, several practical ORE schemes have been proposed but these schemes inevitably leak some information on plaintexts in addition to the comparison result [8]–[10].

All of the previous ORE studies only considered to compare ciphertexts generated by a single client. However, in a real environment, it is necessary to compare ciphertexts generated by multiple clients if these clients handle related

plaintexts. For example, we consider a scenario where students are divided into multiple classes to take lectures taught by different instructors. In this case, the grades of each class are encrypted by the encryption key of each instructor, but if necessary, the grades of these different classes should be comparable without decryption. As another example, we can consider a scenario in which patients are treated by different physicians in a hospital and their medical data are encrypted and stored with the secret keys of physicians. In this case, a physician may want to compare the medical data of patients that he or she has treated with the medical data of other patients that have been treated by other physicians for medical research purposes. To support these scenarios, a comparison key must be provided that can compare the encrypted data generated by multiple clients and this comparison key should be provided only to an authorized user. We call the ORE scheme that supports comparison operations not only on ciphertexts generated by one client but also on ciphertexts generated by different clients, as the multi-client order-revealing encryption (MC-ORE) scheme.

We note that an MC-ORE scheme can be easily derived from a multi-input functional encryption (MI-FE) scheme [11]. That is, if each ciphertext slot of an MI-FE scheme is related to the client index of an MC-ORE scheme and an MI-FE private key for the comparison function on two ciphertexts is provided as an MC-ORE comparison key, then we can build an MC-ORE scheme from an MI-FE scheme. However, this approach is not practical because an MI-FE scheme for general functions requires heavy cryptographic tools such as multi-linear maps or indistinguishable obfuscation.

A. OUR RESULTS

We summarize the contributions of this paper which include the notion of MC-ORE and two practical MC-ORE schemes with limited leakage in bilinear maps.

1) DEFINITION

We first introduce the notion of MC-ORE by extending the concept of ORE [7] to additionally support the comparison operation on ciphertexts which are generated by multiple clients. In an MC-ORE scheme, each client creates ciphertexts by encrypting plaintexts with his/her secret key and anyone can publicly compare the order of two ciphertexts generated by a single client similar to the functionality of ORE. In addition to this basic functionality, it supports the comparison operation of two ciphertexts created by different clients if an additional comparison key for two clients is given. Note that the comparison of two ciphertexts generated from different clients is not a public operation since a comparison key given from a trusted center is needed to prevent the leakage resulting from these comparisons. To define the security model of MC-ORE, we follow the security model of ORE that allows the leakage [9]. In this work, we give a simulation-based security model for MC-ORE with a leakage function \mathcal{L} . Informally, this definition states that if an

adversary can obtain information from ciphertexts of clients' plaintexts $(j_1, m_1), \dots, (j_q, m_q)$ where j_k is the index of a client, then it can be inferred from $\mathcal{L}((j_1, m_1), \dots, (j_q, m_q))$. One difference between our security model and that of ORE with the leakage is that the adversary can query many comparison keys for different clients. To handle this comparison key query, we define the static security model which requires that the adversary should first specify a set of corrupted client indices.

2) BASIC CONSTRUCTION

Next we propose two realizable MC-ORE schemes with different leakage functions. Our first MC-ORE scheme conceptually follows the design principle of the ORE scheme of Chenette et al. [9] that encrypts each bit of a plaintext by using a pseudo-random function (PRF) that takes a prefix of the plaintext as an input. However, it is not easy to extend an ORE scheme that uses a PRF to an MC-ORE scheme that supports the comparison operation for different clients since the outputs of PRF with different client's keys are random values. To solve this difficulty, we use an algebraic PRF in bilinear groups which is defined as $PRF_{s,x} = H(x)^s$ where H is a hash function and s is a PRF key [12]. Suppose there is a single client and the client creates ciphertexts $C = (H(x)^s, H(x+1)^s)$ and $C' = (H(x')^s, H(x'+1)^s)$ for plaintexts x and x' in binary values by using a secret key s . A user can publicly check whether $x+1 = x'$ or not by comparing $H(x+1)^s = H(x')^s$ from two ciphertexts. Now suppose there are two clients with different secret keys s and s' and clients create ciphertexts $C = (H(x)^s, H(x+1)^s)$ and $C' = (H(x')^{s'}, H(x'+1)^{s'})$ for plaintexts x and x' in binary values respectively. To compare two ciphertexts generated by different clients, a user first receives a comparison key $CK = (\hat{g}^{rs}, \hat{g}^{r's'})$ from a trusted center and checks whether $x+1 = x'$ or not by comparing $e(H(x+1)^s, \hat{g}^{r's'}) = e(H(x')^{s'}, \hat{g}^{rs})$. To extend the comparison of binary values to large values, we modify the encoding method of Chenette et al. [9] that uses the prefixes of a plaintext. Let $m = x_1 x_2 \dots x_n \in \{0, 1\}^n$ be a plaintext. For each $i \in [n]$, the encryption algorithm encodes two strings $E_{i,0} = x_1 x_2 \dots x_{i-1} \| 0 x_i$ and $E_{i,1} = x_1 x_2 \dots x_{i-1} \| (0 x_i + 1)$ and evaluates $C_{i,0} = H(E_{i,0})^s$ and $C_{i,1} = H(E_{i,1})^s$. For example, the third bit of $m = 101$ is encoded as $E_{3,0} = 10 \| 01 = 1001$ and $E_{3,1} = 10 \| (01 + 1) = 10 \| 10 = 1010$. The ciphertext is formed as $CT = (\{C_{i,0}, C_{i,1}\}_{i \in [n]})$. Note that we have $m < m'$ if there is the smallest index i^* such that the prefixes of two plaintexts with $i^* - 1$ length are equal and $x_{i^*} + 1 = x'_{i^*}$. We prove the security of our first MC-ORE scheme in the simulation-based (SIM) security model with a leakage function that reveals the comparison result as well as the most significant differing bit.

3) ENHANCED CONSTRUCTION

Our second MC-ORE scheme is the enhanced version of the first MC-ORE scheme that reduces the leakage due to the comparison of ciphertexts generated by a single client. In our first scheme, a ciphertext was simultaneously used for

two purposes: ciphertext comparisons in a single client and ciphertext comparisons between different clients. In the second scheme, we divide the ciphertext into two parts and treat each ciphertext part differently. That is, the first ciphertext part is only used for ciphertext comparisons in a single client, and the second ciphertext part is only used for ciphertext comparisons between different clients. For the first ciphertext part, we can use any ORE scheme that has the reduced leakage [8], [10]. For the second ciphertext part, we construct an encrypted ORE (EORE) scheme by modifying our first MC-ORE scheme. In the EORE scheme, an (encrypted) ciphertext is created by first generating a ciphertext of the first MC-ORE scheme and then encrypting it with a public-key encryption scheme. Unlike the first MC-ORE scheme, this EORE scheme does not allow ciphertext comparisons in a single client since ciphertexts are securely encrypted. However, it allows ciphertext comparisons between different clients since it can derive the original ciphertexts of the first MC-ORE scheme if a comparison key is provided by the trusted center. Therefore, there is *no leakage* from the second ciphertext part and the leakage only depends on the first ciphertext part if comparison keys are not exposed. We prove the SIM security of our second MC-ORE scheme under the external Diffie-Hellman assumption.

4) IMPLEMENTATION

Finally, we implement our MC-ORE schemes and evaluate the performance of each algorithm. The proposed MC-ORE scheme provides single-client comparison and multi-client comparison algorithms. In the MC-ORE scheme, the most computationally expensive algorithm is the multi-client comparison algorithm since it requires two pairing operations per each bit comparison until the most significant differing bit (MSDB) is found. To improve this multi-client comparison, we present other comparison methods and compare the performance of these suggested methods. The first method is a simple method that performs the comparison sequentially from the ciphertext element of the most significant bit to that of the least significant bit. It is efficient when the MSDB exists in the higher bits, but it is inefficient when the MSDB exists in the lower bits. The second method is a binary search method that uses a binary search instead of a sequential search to find the MSDB location. This method performs approximately $\log n$ computations to find the MSDB location where n is the length of a plaintext. The third method is a hybrid method that combines multi-client comparisons and single-client comparisons. This method can improve the performance of ciphertext comparisons between multiple ciphertexts by performing one multi-client comparison and many single-client comparisons.

B. RELATED WORK

1) ORDER-PRESERVING ENCRYPTION

The concept of OPE was introduced by Agrawal *et al.* [4] in the database community, and this is a symmetric-key

encryption scheme that supports efficient comparison operations on ciphertexts since the order of plaintexts is maintained in ciphertexts. The security model of OPE was presented by Boldyreva *et al.* [5], and it is called indistinguishability under ordered chosen plaintext attack (IND-OCPA). The security notion of IND-OCPA says that an adversary can not obtain any information from ciphertexts except the order of underlying plaintexts. However, the ciphertext space of OPE is required to be extremely large to satisfy this IND-OCPA security. To achieve this IND-OCPA security, several variants of OPE such as mutable OPE have been proposed, but most of them are inefficient since they require stateful encryption and an interactive protocol [13]–[15].

2) ORDER-REVEALING ENCRYPTION

Boneh *et al.* [7] introduced the notion of ORE which is a generalization of OPE where the order of plaintexts can be publicly compared by running a comparison algorithm on ciphertexts. They also proposed a specific ORE scheme that achieves the IND-OCPA security by using multi-linear maps, but this scheme is quite impractical. Chenette *et al.* [9] constructed the first practical ORE scheme by encrypting each bit of messages using pseudo-random functions. They showed that their scheme achieve a weaker security model of ORE that reveals additional information of underlying plaintexts in addition to the order of plaintexts. After the work of Chenette *et al.*, many ORE schemes were proposed to reduce the additional leakage. Lewi and Wu [10] constructed an IND-OCPA secure ORE scheme only for small plaintext spaces by decomposing the encryption algorithm into two separate functions, left encryption and right encryption where the right encryption achieves the IND-OCPA security. Cash *et al.* [8] constructed an ORE scheme with reduced leakage by using property-preserving hash functions in bilinear maps. Although this ORE scheme achieves to reduce the leakage, it is inefficient due to the larger size of ciphertexts and the pairing operation.

3) ATTACKS ON ORE

Naveed *et al.* [16] explored inference attacks on encrypted database columns to recover messages against ORE-encrypted databases. These attacks usually use the order and frequency of plaintexts and auxiliary information such as plaintext distribution. Durak *et al.* [17] and Grubbs *et al.* [18] proposed improved inference attacks of Naveed *et al.* in several ways and additionally presented leakage-abuse attacks against ORE schemes with the specified leakage. Both attacks showed that the leakage of ORE can be effectively used to recover more accurate plaintexts than that was theoretically analyzed.

II. MULTI-CLIENT ORDER-REVEALING ENCRYPTION

In this section, we define the syntax and the security model of multi-client order-revealing encryption by extending those of order-revealing encryption.

A. NOTATION

Let $[n]$ be the set of $\{1, \dots, n\}$ and $[k, n]$ be the set of $\{k, \dots, n\}$. Let $\mathbf{cmp}(m, m')$ be a comparison function that returns 1 if $m < m'$ and returns 0, otherwise. Let $\mathbf{ind}(m, m')$ be an index function that returns the index of the most significant differing bit between plaintexts m and m' of n -bits and returns $n + 1$ if $m = m'$. Let $\mathbf{prefix}(m, i)$ be a prefix function that takes as input a plaintext $m = x_1 x_2 \dots x_n \in \{0, 1\}^n$ and an index i and returns $x_1 x_2 \dots x_{i-1}$ as the prefix of x_i .

B. ORDER-REVEALING ENCRYPTION

Order-revealing encryption (ORE) is a special kind of symmetric-key encryption that supports a comparison operation on encrypted data by using a public procedure [7]. In ORE, a client creates ciphertexts of plaintexts by using his/her secret key SK and uploads these ciphertexts to a remote database. After that, anyone can compare the order of two ciphertexts CT and CT' by using a public comparison algorithm. The following is the syntax of ORE given by Chenette et al. [9].

Definition 1 (ORE): An ORE scheme consists of three algorithms, **Setup**, **Encrypt**, **Compare** which are defined over a well-ordered domain \mathcal{D} as follows:

- **Setup**(1^λ). The setup algorithm takes as input a security parameter λ and outputs a secret key SK .
- **Encrypt**(m, SK). The encryption algorithm takes as input a plaintext $m \in \mathcal{D}$ and the secret key SK and outputs a ciphertext CT .
- **Compare**(CT, CT'). The comparison algorithm takes as input two ciphertexts CT and CT' and outputs a comparison bit $b \in \{0, 1\}$.

The correctness of ORE is defined as follows: For all SK generated by **Setup** and any CT, CT' generated by **Encrypt** on plaintexts m, m' , it is required that $\mathbf{Compare}(CT, CT') = \mathbf{cmp}(m, m')$.

The best possible security of ORE, which is IND-OCPA, was defined by Boneh et al. [7]. The simulation-based security of ORE with additional leakage \mathcal{L} was defined by Chenette et al. [9].

C. MULTI-CLIENT ORDER-REVEALING ENCRYPTION

Multi-client order-revealing encryption (MC-ORE) is an extension of ORE that supports comparison operations not only between ciphertexts generated by a single client but also between ciphertexts generated by different clients. In MC-ORE, each client of an index j creates ciphertexts of plaintexts by using his/her secret key SK_j which is given by a trusted center. Anyone can compare two ciphertexts CT_j and CT'_j generated by the single client by using a public comparison algorithm as the same as in ORE. In addition, a client can compare two ciphertexts CT_j and CT'_k generated by different clients with different indices j and k if the client obtains a comparison key $CK_{j,k}$ from the trusted center. The syntax of MC-ORE is given as follows.

Definition 2 (MC-ORE): An MC-ORE scheme consists of six algorithms, **Setup**, **GenKey**, **Encrypt**, **Compare**, **GenCmpKey**, and **CompareMC**, which are defined as follows:

- **Setup**($1^\lambda, N$). The setup algorithm takes as input a security parameter λ and the number of clients $N \in \mathbb{N}$ and outputs a master key MK and public parameters PP .
- **GenKey**(j, MK, PP). The key generation algorithm takes as input a client index $j \in [N]$, the master key MK , and the public parameters PP . It outputs a secret key SK_j for the client index j .
- **Encrypt**(m, SK_j, PP). The encryption algorithm takes as input a plaintext $m \in \mathcal{D}$, the secret key SK_j , and the public parameters PP . It outputs a ciphertext CT_j .
- **Compare**(CT_j, CT'_j, PP). The comparison algorithm takes as input two ciphertexts CT_j, CT'_j of the same client index j and the public parameters PP . It outputs a comparison bit $b \in \{0, 1\}$.
- **GenCmpKey**(j, k, MK, PP). The comparison key generation algorithm takes as input two client indices j, k , the master key MK , and the public parameters PP . It outputs a comparison key $CK_{j,k}$ for two clients.
- **CompareMC**($CT_j, CT'_k, CK_{j,k}, PP$). The multi-client comparison algorithm takes as input two ciphertexts CT_j, CT'_k of two client indices j, k , the comparison key $CK_{j,k}$, and the public parameters PP . It outputs a comparison bit $b \in \{0, 1\}$.

The correctness of MC-ORE is defined as follows: For all $PP, MK, \{SK_j\}_{j \in [N]}$ generated by **Setup** and **GenKey**, any $CK_{j,k}$ generated by **GenCmpKey**, and any CT_j, CT'_j, CT'_k generated by **Encrypt** on plaintexts m, m', m'' , it is required that:

$$\mathbf{Compare}(CT_j, CT'_j, PP) = \mathbf{cmp}(m, m') \text{ and}$$

$$\mathbf{CompareMC}(CT_j, CT'_k, CK_{j,k}, PP) = \mathbf{cmp}(m, m'').$$

The simulation-based security (SIM-security) model of MC-ORE is defined with a leakage function which enables quantifying any information inevitably leaked from the scheme. Since the leakage is affected by whether comparison keys are exposed, the leakage function \mathcal{L}_S is defined with respect to a set S of the revealed comparison keys. In the real experiment, an adversary can access a comparison key generation oracle to obtain any comparison key as well as an encryption oracle to obtain any ciphertext of its choice (j_i, m_i) where j_i is the client index corresponding to the i -th message m_i . Eventually, the adversary outputs the deducing result from the given information. In the ideal experiment, the adversary also can obtain any comparison key and any ciphertext, but all values are generated by the simulator which has only the information derived from the leakage function $\mathcal{L}_S((j_1, m_1), \dots, (j_q, m_q))$. The security is proved by showing the outputs of two distributions are indistinguishable.

However, the leakage function is influenced by the order of ciphertext queries and comparison key queries. When

(j_1, m_1) and (j_2, m_2) are queried to the encryption oracle, the simulator generates ciphertexts CT_{j_1} and CT_{j_2} with no leakage if the comparison key CK_{j_1, j_2} was not exposed. After that, if the adversary requests CK_{j_1, j_2} causing the leakage $\mathcal{L}_S((j_1, m_1), (j_2, m_2))$, it can identify that there is something wrong in the simulation of CT_{j_1} and CT_{j_2} . That is, the simulator should have generated the ciphertexts by predicting the leakage but it is difficult to simulate with such a flexible leakage function. In addition, when CK_{j_1, j_3} and CK_{j_2, j_3} are exposed, the simulator generates CT_{j_1} and CT_{j_2} with no leakage since CK_{j_1, j_2} is not exposed. After that, if (j_3, m_3) is queried to the encryption oracle, the simulator generates CT_{j_3} with the leakage $\mathcal{L}_S((j_1, m_3), (j_2, m_3))$. Again, the adversary can notice that the simulation of CT_{j_1} and CT_{j_2} is wrong. Thus, we define the static version of the SIM-security model in which a set S of revealed comparison keys and the ciphertext queries are initially fixed. The static SIM-security model of MC-ORE with the leakage function \mathcal{L}_S is defined as follows.

Definition 3 (Static SIM-Security With Leakage): For a security parameter λ , let \mathcal{A} be an adversary and \mathcal{B} be a simulator. Let $S = \{(j, k)\}_{j, k \in [N]}$ be a set of index tuples where $CK_{j, k}$ is revealed and let $\mathcal{L}_S(\cdot)$ be a leakage function. The experiments of $\text{REAL}_{\mathcal{A}}^{\text{MC-ORE}}(\lambda)$ and $\text{SIM}_{\mathcal{A}, \mathcal{B}, \mathcal{L}}^{\text{MC-ORE}}(\lambda)$ are defined as follows:

$\text{REAL}_{\mathcal{A}}^{\text{MC-ORE}}(\lambda)$

1. $(st_{\mathcal{A}}, S, ((j_1, m_1), \dots, (j_q, m_q))) \leftarrow \mathcal{A}(1^\lambda)$
2. $(PP, MK) \leftarrow \text{Setup}(1^\lambda, N)$
3. $CK_{j, k} \leftarrow \text{GenCmpKey}(j, k, MK, PP), \forall (j, k) \in S$
4. for $1 \leq i \leq q$,
 $CT_{j_i} \leftarrow \text{Encrypt}(m_i, SK_{j_i}, PP)$
5. Output $(CT_{j_1}, \dots, CT_{j_q})$ and $st_{\mathcal{A}}$

$\text{SIM}_{\mathcal{A}, \mathcal{B}, \mathcal{L}}^{\text{MC-ORE}}(\lambda)$

1. $(st_{\mathcal{A}}, S, ((j_1, m_1), \dots, (j_q, m_q))) \leftarrow \mathcal{A}(1^\lambda)$
2. $(st_{\mathcal{B}}, PP) \leftarrow \mathcal{B}(1^\lambda, N)$
3. $CK_{j, k} \leftarrow \mathcal{B}(st_{\mathcal{B}}), \forall (j, k) \in S$
4. for $1 \leq i \leq q$,
 $(st_{\mathcal{B}}, CT_{j_i}) \leftarrow \mathcal{B}(st_{\mathcal{B}}, \mathcal{L}_S((j_1, m_1), \dots, (j_i, m_i)))$
5. Output $(CT_{j_1}, \dots, CT_{j_q})$ and $st_{\mathcal{A}}$

We say that an MC-ORE scheme is ST-SIM secure if for all polynomial-size adversaries \mathcal{A} , there exists a polynomial-size simulator \mathcal{B} such that the outputs of the two distributions $\text{REAL}_{\mathcal{A}}^{\text{MC-ORE}}(\lambda)$ and $\text{SIM}_{\mathcal{A}, \mathcal{B}, \mathcal{L}}^{\text{MC-ORE}}(\lambda)$ are indistinguishable.

Remark 4: For $S = \{(j, k)\}_{j, k \in [N]}$ of index tuples where the comparison key $CK_{j, k}$ is revealed, let \mathcal{L}_S be the following leakage function:

$$\begin{aligned} \mathcal{L}_S((j_1, m_1), \dots, (j_q, m_q)) \\ = \{\text{cmp}(m_{i'}, m_i) : 1 \leq i' < i \leq q, j_{i'} = j_i \text{ or } (j_{i'}, j_i) \in S\}. \end{aligned}$$

If an MC-ORE scheme is secure with leakage \mathcal{L}_S , then it is IND-OCPA secure.

III. BASIC MC-ORE CONSTRUCTION

In this section, we propose our first construction of MC-ORE with leakage and prove the ST-SIM security of our scheme.

A. ASYMMETRIC BILINEAR GROUPS

Let \mathcal{G}_{as} be a group generator algorithm that 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 $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$ 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 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 $\hat{\mathbb{G}}$.

Assumption 5 (External Diffie-Hellman, XDH): Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be a tuple randomly generated by $\mathcal{G}_{as}(1^\lambda)$ where p is a prime order of the groups. Let g, \hat{g} be random generators of groups $\mathbb{G}, \hat{\mathbb{G}}$, respectively. The XDH assumption is that the decisional Diffie-Hellman(DDH) assumption holds in \mathbb{G} . That is, if the challenge tuple

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

are given, no PPT algorithm \mathcal{A} can distinguish $T = T_0 = g^{ab}$ from $T = T_1 = g^c$ with more than a negligible advantage. The advantage of \mathcal{A} is defined as $\text{Adv}_{\mathcal{A}}^{\text{XDH}}(\lambda) = |\Pr[\mathcal{A}(D, T_0) = 0] - \Pr[\mathcal{A}(D, T_1) = 0]|$ where the probability is taken over random choices of $a, b, c \in \mathbb{Z}_p$.

B. CONSTRUCTION

Before we present our basic MC-ORE scheme, we first define a leakage function for our scheme. Let $N \in \mathbb{N}$ be the maximum number of clients and $S = \{(j, k)\}_{j, k \in [N]}$ be a set of client index tuples where a comparison key $CK_{j, k}$ is revealed. A leakage function \mathcal{L}_S is defined as follows:

$$\begin{aligned} \mathcal{L}_S((j_1, m_1), \dots, (j_q, m_q)) = \{\text{cmp}(m_{i'}, m_i), \text{ind}(m_{i'}, m_i) : \\ 1 \leq i' < i \leq q, j_{i'} = j_i \text{ or } (j_{i'}, j_i) \in S\}. \end{aligned}$$

If $S = \emptyset$, then \mathcal{L}_S becomes equal to the leakage function \mathcal{L} defined by Chenette et al. [9]. Otherwise, i.e. if $S \neq \emptyset$, some comparison keys are revealed and it causes increased leakage. Our basic MC-ORE scheme is described as follows:

MC-ORE.Setup $(1^\lambda, N)$. This algorithm first generates bilinear groups $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ of prime order p with group generators $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$. It chooses a random exponent $s_j \in \mathbb{Z}_p$ for all $j \in [N]$ and outputs a master key $MK = \{s_j\}_{j \in [N]}$ and public parameters $PP = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, H)$ where $H : \{0, 1\}^* \rightarrow \mathbb{G}$ is a full-domain hash function.

MC-ORE.GenKey (j, MK, PP) . Let $MK = \{s_1, \dots, s_N\}$. It outputs a secret key $SK_j = s_j$.

MC-ORE.Encrypt(m, SK_j, PP). Let $m = x_1 x_2 \cdots x_n \in \{0, 1\}^n$ and $SK_j = s_j$. For each $i \in [n]$, it computes $C_{i,0} = H(\mathbf{prefix}(m, i) \| 0 x_i)^{s_j}$ and $C_{i,1} = H(\mathbf{prefix}(m, i) \| (0 x_i + 1))^{s_j}$ where $\|$ is the concatenation of two bit strings. It outputs a ciphertext $CT_j = (\{C_{i,0}, C_{i,1}\}_{i \in [n]})$.

MC-ORE.Compare(CT_j, CT'_j, PP). For the same client index j , let $CT_j = (\{C_{i,0}, C_{i,1}\}_{i \in [n]})$ and $CT'_j = (\{C'_{i,0}, C'_{i,1}\}_{i \in [n]})$. It first finds the smallest index i^* such that $C_{i^*,0} \neq C'_{i^*,0}$ by sequentially comparing $C_{i,0}$ and $C'_{i,0}$. If such index i^* exists and $C_{i^*,1} = C'_{i^*,1}$ holds, then it outputs 1. If such index i^* exists and $C_{i^*,0} = C'_{i^*,0}$, then it outputs 0. If no such index i^* exists, then it outputs 0.

MC-ORE.GenCmpKey(j, k, MK, PP). Let s_j and s_k be the secret keys of client indices j and k . It chooses a random exponent $r \in \mathbb{Z}_p$ and computes $K_0 = \hat{g}^{rs_j}, K_1 = \hat{g}^{rs_k}$. It outputs a comparison key $CK_{j,k} = (K_0, K_1)$.

MC-ORE.CompareMC($CT_j, CT'_k, CK_{j,k}, PP$). Let $CT_j = (\{C_{i,0}, C_{i,1}\}_{i \in [n]})$ and $CT'_k = (\{C'_{i,0}, C'_{i,1}\}_{i \in [n]})$. Let $CK_{j,k} = (K_0, K_1)$. It first finds the smallest index i^* such that $e(C_{i^*,0}, K_1) \neq e(C'_{i^*,0}, K_0)$ by sequentially comparing $e(C_{i,0}, K_1)$ and $e(C'_{i,0}, K_0)$. If such index i^* exists and $e(C_{i^*,1}, K_1) = e(C'_{i^*,0}, K_0)$ holds, then it outputs 1. If such index i^* exists and $e(C_{i^*,0}, K_1) = e(C'_{i^*,1}, K_0)$, then it outputs 0. If no such index i^* exists, then it outputs 0.

C. CORRECTNESS

To show the correctness of the above scheme, we define encoding functions E_0, E_1 that take (i, m) as input and output the encoded i -th bit of $m = x_1 \cdots x_n \in \{0, 1\}^n$ as follows:

$$E_0(i, m) = \mathbf{prefix}(m, i) \| 0 x_i, E_1(i, m) = \mathbf{prefix}(m, i) \| (0 x_i + 1).$$

The encoding functions satisfy the following conditions. If $m = m'$, $E_0(i, m) = E_0(i, m')$ holds for all $i \in [n]$. If $m < m'$ and i^* is the smallest index such that $x_{i^*} \neq x'_{i^*}$, then $E_0(i, m) = E_0(i, m')$ holds for all $i < i^*$ and $E_0(i, m) \neq E_0(i, m')$ holds for all $i \geq i^*$, and especially, $E_1(i^*, m) = E_0(i^*, m')$ holds.

Let $SK_j = s_j$ be the secret key of a client index j and $CT_j = (\{C_{i,0}, C_{i,1}\}_{i \in [n]})$ and $CT'_j = (\{C'_{i,0}, C'_{i,1}\}_{i \in [n]})$ be ciphertexts on messages $m = x_1 x_2 \cdots x_n \in \{0, 1\}^n$ and $m' = x'_1 x'_2 \cdots x'_n \in \{0, 1\}^n$. If $m < m'$, there must be the smallest index i^* such that $x_i = x'_i$ for all $i < i^*$ and $x_{i^*} \neq x'_{i^*}$. Thus, we have that

$$C_{i,0} = H(E_0(i, m))^{s_j} = H(E_0(i, m'))^{s_j} = C'_{i,0} \quad \forall i < i^*$$

and

$$C_{i^*,1} = H(E_1(i^*, m))^{s_j} = H(E_0(i^*, m'))^{s_j} = C'_{i^*,0}.$$

Let $SK_j = s_j$ and $SK_k = s_k$ be the secret keys of two client indices j and k , and $CK_{j,k} = (K_0, K_1) = (\hat{g}^{rs_j}, \hat{g}^{rs_k})$ be the comparison key. Let $CT_j = (\{C_{i,0}, C_{i,1}\}_{i \in [n]})$ and

$CT'_k = (\{C'_{i,0}, C'_{i,1}\}_{i \in [n]})$ be ciphertexts on messages m and m' . If $m < m'$, there must be the smallest index i^* such that $x_i = x'_i$ for all $i < i^*$ and $x_{i^*} \neq x'_{i^*}$. Thus, we have that

$$e(C_{i,0}, K_1) = e(H(E_0(i, m))^{s_j}, \hat{g}^{rs_k}) = e(H(E_0(i, m)), \hat{g})^{rs_j s_k} = e(H(E_0(i, m'))^{s_k}, \hat{g}^{rs_j}) = e(C'_{i,0}, K_0) \quad \forall i < i^*$$

and

$$e(C_{i^*,1}, K_1) = e(H(E_1(i^*, m))^{s_j}, \hat{g}^{rs_k}) = e(H(E_1(i^*, m)), \hat{g})^{rs_j s_k} = e(H(E_0(i^*, m'))^{s_k}, \hat{g}^{rs_j}) = e(C'_{i^*,0}, K_0).$$

D. SECURITY ANALYSIS

We prove the security of the basic MC-ORE scheme with the leakage function \mathcal{L}_S in the ST-SIM security model. We define a sequence of experiments from \mathbf{H}_0 corresponding to the real experiment to \mathbf{H}_3 corresponding to the ideal experiment and show that the outputs of two experiments are indistinguishable. At first, the ciphertexts of clients whose comparison keys are not exposed are randomly generated. In the next experiment, the ciphertexts of clients whose comparison keys are exposed are generated with random values. Finally, in the last experiment \mathbf{H}_3 , the ciphertexts are simulated with respect to the leakage function \mathcal{L}_S , and consequently \mathbf{H}_3 corresponds to the ideal experiment. The details are given as follows.

Theorem 6: The basic MC-ORE scheme is ST-SIM secure with the leakage function \mathcal{L}_S in the random oracle model if the XDH assumption holds.

Proof: We prove the security of the basic MC-ORE scheme through a sequence of hybrid experiments. The first experiment is defined as the real MC-ORE security experiment and the last one is defined as the ideal experiment with the leakage function \mathcal{L}_S in which the adversary has no advantage. The hybrid experiments $\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2$, and \mathbf{H}_3 are defined as follows:

H₀: This experiment corresponds to the real world experiment.

H₁: This experiment is similar to \mathbf{H}_0 except that the ciphertext CT_j such that $(j, j') \notin S$ for any client index j' is generated by using random elements.

H₂: This experiment is similar to \mathbf{H}_1 except that the ciphertext CT_j such that $(j, j') \in S$ for some client index j' is generated by using random elements.

H₃: In this experiment, the ciphertexts are generated with the leakage function \mathcal{L}_S and the rest are same to \mathbf{H}_2 . This experiment corresponds to the ideal world experiment.

From the following Lemmas 8, 9, and 10 that claim the indistinguishability of the above experiments, we have that \mathbf{H}_0 and \mathbf{H}_3 are computationally indistinguishable. ■

Before we present additional Lemmas for the proof of the above theorem, we define the encoded messages $E_0(k, m) = \mathbf{prefix}(m, k) \| 0 x_k$ and $E_1(k, m) = \mathbf{prefix}(m, k) \| (0 x_k + 1)$ where $m = x_1 \cdots x_n \in \{0, 1\}^n$. In addition, we introduce the multi-external Diffie-Hellman assumption.

Assumption 7 (Multi-External Diffie-Hellman, mXDH): Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be a tuple randomly generated by

$\mathcal{G}_{as}(1^\lambda)$ where p is a prime order of the groups. Let g, \hat{g} be random generators of groups $\mathbb{G}, \hat{\mathbb{G}}$, respectively. The mXDH assumption is that if the challenge tuple

$$D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, g^a, \{g^{b_{i,1}}, \dots, g^{b_{i,n}}\}_{i \in [t]}) \text{ and } T$$

are given, no PPT algorithm \mathcal{A} can distinguish $T = T_0 = (\{g^{ab_{i,1}}, \dots, g^{ab_{i,n}}\}_{i \in [t]})$ from $T = T_1 = (\{g^{c_{i,1}}, \dots, g^{c_{i,n}}\}_{i \in [t]})$ with more than a negligible advantage. The advantage of \mathcal{A} is defined as $\text{Adv}_{\mathcal{A}}^{\text{mXDH}}(\lambda) = |\Pr[\mathcal{A}(D, T_0) = 0] - \Pr[\mathcal{A}(D, T_1) = 0]|$ where the probability is taken over random choices of $a, (b_{i,1}, \dots, b_{i,n}), (c_{i,1}, \dots, c_{i,n}) \in \mathbb{Z}_p$ for all $i \in [t]$.

This mXDH assumption is equivalent to the XDH assumption since the challenge tuple of mXDH assumption can be obtained from the XDH assumption by using the random self-reducibility property [19].

Lemma 8: The hybrid experiments \mathbf{H}_0 and \mathbf{H}_1 are computationally indistinguishable to the polynomial-time adversary assuming that the mXDH assumption holds.

Proof: To prove this lemma, we additionally define a sequence of hybrid experiments $\mathbf{H}_0 = \mathbf{H}_{0,0}, \mathbf{H}_{0,1}, \dots, \mathbf{H}_{0,\tilde{q}} = \mathbf{H}_1$ for some \tilde{q} as follows.

$\mathbf{H}_{0,\mu}$: Let $I = (j_1, \dots, j_q)$ be a tuple of challenge client index.

For all $j_i \in I$ such that $(j_i, *) \notin S$, let $j_1^*, \dots, j_{\tilde{q}}^* \in I$ be distinct client indices where $\tilde{q} \leq q$. Let $\mathbf{SI}_\mu = \{i \in [q] : j_i = j_\mu^*\}$ be an index set of same client indices where $\mu \in [\tilde{q}]$. In this experiment, we change the generation of the μ -th ciphertext set with the index set \mathbf{SI}_μ . If $\ell \leq \mu$, the ciphertexts in the ℓ -th ciphertext set with \mathbf{SI}_ℓ are changed to be random elements. Otherwise, the ciphertexts in the ℓ -th ciphertext set with \mathbf{SI}_ℓ are generated by running the normal encryption algorithm. Note that the ciphertexts with the client index j_i such that $(j_i, *) \in S$ in $\mathbf{H}_{0,\mu-1}$ and $\mathbf{H}_{0,\mu}$ are equally generated by running the normal encryption algorithm.

Without loss of generality, we assume that $(j_\mu^*, *) \notin S$. Suppose there exists an adversary \mathcal{A} that distinguishes $\mathbf{H}_{0,\mu-1}$ from $\mathbf{H}_{0,\mu}$ with non-negligible advantage. A simulator \mathcal{B} that solves the mXDH assumption using \mathcal{A} is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, g^a, \{g^{b_{i,1}}, \dots, g^{b_{i,2n}}\}_{i \in [t]})$ and $T = (\{X_{i,1}, \dots, X_{i,2n}\}_{i \in [t]})$. \mathcal{B} interacts with \mathcal{A} as follows.

Let $(st_{\mathcal{A}}, S, ((j_1, m_1), \dots, (j_q, m_q)))$ be the output of \mathcal{A} and \mathbf{SI}_μ be the target index set of j_μ^* . The simulator \mathcal{B} first sets the secret keys of all clients except the target client. For each $j \neq j_\mu^*$, it chooses a random exponent s_j and sets $SK_j = s_j$. For the target client index j_μ^* , it implicitly sets $SK_{j_\mu^*} = a$. Now, \mathcal{B} can generate any comparison key $CK_{j,k}$ for all tuple $(j, k) \in S$ since it knows secret keys s_j and s_k if $(j, k) \in S$.

To handle hash queries, \mathcal{B} maintains a random oracle table T_H for the consistency of a simulation. Initially, \mathcal{B} fixes some hash queries for the simulation of the ciphertext with the challenge tuple (j_i, m_i) such that $i \in \mathbf{SI}_\mu$, which is output of \mathcal{A} . For the first message m_1 , \mathcal{B} sets $h_{k,0} = g^{b_{i,2k-1}}, h_{k,1} = g^{b_{i,2k}}$ and adds the tuples $(E_0(k, m_1), h_{k,0})$ and $(E_1(k, m_1), h_{k,1})$ to

the table T_H for all $k \in [n]$. For each message m_i , \mathcal{B} first finds the biggest index $d = \mathbf{ind}(m_i, m_{i'})$ for any $i' < i$ and finds tuples $(E_0(k, m_{i'}), h'_{k,0}), (E_1(k, m_{i'}), h'_{k,1})$ from T_H for all $k \in [d]$. It sets $h_{k,0} = h'_{k,0}, h_{k,1} = h'_{k,1}$ for all $k \in [d-1]$ since $E_0(k, m_{i'}) = E_0(k, m_i)$ and $E_1(k, m_{i'}) = E_1(k, m_i)$. If $\mathbf{cmp}(m_i, m_{i'}) = 1$, then \mathcal{B} sets $h_{d,0} = g^{b_{i,2d-1}}, h_{d,1} = h'_{d,0}$ and otherwise, it sets $h_{d,0} = h'_{d,1}, h_{d,1} = g^{b_{i,2d}}$. Next, it sets $h_{k,0} = g^{b_{i,2k-1}}, h_{k,1} = g^{b_{i,2k}}$ for all $k \in [d+1, n]$. It adds the tuples $(E_0(k, m_i), h_{k,0})$ and $(E_1(k, m_i), h_{k,1})$ to the table T_H for all $k \in [n]$. After that, if a random oracle query for an encoded message $E_\beta(k, m)$ is requested for each $\beta \in \{0, 1\}$, \mathcal{B} first finds a tuple $(E_\beta(k, m), h)$ on the table T_H . If the tuple does not exist, then it chooses a random element $h \in \mathbb{G}$ and adds the tuple $(E_\beta(k, m), h)$ to T_H . Finally it gives h to \mathcal{A} as a response.

To handle the creation of ciphertexts, \mathcal{B} carefully uses the hash table and the challenge elements in the assumption. Let $((j_1, m_1), \dots, (j_q, m_q))$ be the challenge tuples. If $(j_i, *) \in S$, then \mathcal{B} simply creates a ciphertext by running the **MC-ORE.Encrypt** algorithm with hash queries since it knows the secret key s_{j_i} . If $(j_i, *) \notin S$, then it means that $i \in \mathbf{SI}_\ell$ for some $\ell \in [\tilde{q}]$. \mathcal{B} creates a set of ciphertexts with the index set \mathbf{SI}_ℓ for each $\ell \in [\tilde{q}]$ as follows:

- **Case $\ell < \mu$:** \mathcal{B} creates the ciphertext for the index $i \in \mathbf{SI}_\ell$ sequentially. For the smallest index $i \in \mathbf{SI}_\ell$, it chooses random elements $R_{k,0}, R_{k,1} \in \mathbb{G}$ for all $k \in [n]$ and creates $CT_{j_i} = (\{R_{k,0}, R_{k,1}\}_{k \in [n]})$. For the next index i , it first finds the biggest index $d = \mathbf{ind}(m_i, m_{i'})$ for any $i' < i$. It sets $C_{k,0} = C'_{k,0}, C_{k,1} = C'_{k,1}$ for all $k \in [d-1]$ where $CT_{j_{i'}} = (\{C'_{k,0}, C'_{k,1}\}_{k \in [n]})$. If $\mathbf{cmp}(m_i, m_{i'}) = 1$, then it chooses a random element $R_{d,0} \in \mathbb{G}$ and sets $C_{d,0} = R_{d,0}, C_{d,1} = C'_{d,0}$. Otherwise, it chooses a random element $R_{d,1} \in \mathbb{G}$ and sets $C_{d,0} = C'_{d,1}, C_{d,1} = R_{d,1}$. Next, it chooses random elements $R_{k,0}, R_{k,1} \in \mathbb{G}$ and sets $C_{k,0} = R_{k,0}, C_{k,1} = R_{k,1}$ for all $k \in [d+1, n]$. It creates the ciphertext $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. At last, it creates the ℓ -th ciphertext set $\mathbf{CT}_{\mathbf{SI}_\ell} = (\{CT_{j_i}\}_{i \in \mathbf{SI}_\ell})$.
- **Case $\ell = \mu$:** \mathcal{B} creates the ciphertext for the index $i \in \mathbf{SI}_\mu$ sequentially. For the smallest index $i \in \mathbf{SI}_\mu$, \mathcal{B} sets $C_{k,0} = X_{i,2k-1}, C_{k,1} = X_{i,2k}$ for all $k \in [n]$ and creates the ciphertext $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. For the next index i , \mathcal{B} first finds the biggest index $d = \mathbf{ind}(m_i, m_{i'})$ for any $i' < i$. It sets $C_{k,0} = C'_{k,0}, C_{k,1} = C'_{k,1}$ for all $k \in [d-1]$ where $CT_{j_{i'}} = (\{C'_{k,0}, C'_{k,1}\}_{k \in [n]})$. If $\mathbf{cmp}(m_i, m_{i'}) = 1$, then it sets $C_{d,0} = X_{i,2d-1}, C_{d,1} = C'_{d,0}$ and otherwise, it sets $C_{d,0} = C'_{d,1}, C_{d,1} = X_{i,2d}$. Next, it sets $C_{k,0} = X_{i,2k-1}, C_{k,1} = X_{i,2k}$ for all $k \in [d+1, n]$. It creates the ciphertext $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. At last, it creates the μ -th ciphertext set $\mathbf{CT}_{\mathbf{SI}_\mu} = (\{CT_{j_i}\}_{i \in \mathbf{SI}_\mu})$. Note that it does not know the secret key a .
- **Case $\ell > \mu$:** It creates the ciphertext set $\mathbf{CT}_{\mathbf{SI}_\ell}$ by running the **MC-ORE.Encrypt** algorithm with hash queries.

If $T = ((g^{ab_{1,1}}, \dots, g^{ab_{1,2n}}), \dots, (g^{ab_{t,1}}, \dots, g^{ab_{t,2n}}))$, then $\mathbf{CT}_{S_{l\mu}}$ are ciphertexts in $\mathbf{H}_{0,\mu-1}$. Otherwise, $\mathbf{CT}_{S_{l\mu}}$ are ciphertexts in $\mathbf{H}_{0,\mu}$. By the mXDH assumption, two experiments $\mathbf{H}_{0,\mu-1}$ and $\mathbf{H}_{0,\mu}$ are computationally indistinguishable. ■

Lemma 9: The hybrid experiments \mathbf{H}_1 and \mathbf{H}_2 are computationally indistinguishable to the polynomial-time adversary assuming that the mXDH assumption holds.

Proof: We additionally define a sequence of hybrid experiments $\mathbf{H}_1 = \mathbf{H}_{1,0}, \mathbf{H}_{1,1}, \dots, \mathbf{H}_{1,\tilde{q}} = \mathbf{H}_2$ for some \tilde{q} as follows.

$\mathbf{H}_{1,\mu}$: Let $I = (j_1, \dots, j_q)$ be a tuple of challenge client index and let $j, j' \in I$ be co-related indices if $(j, j') \in S$ or there exist $\{k_i\}_{i \in [n]} \subseteq I$ such that $(j, k_1), (k_1, k_2), \dots, (k_{n-1}, k_n), (k_n, j') \in S$ for any $n \in [q-2]$. Let $\mathbf{RI}_\mu = \{i \in [q] : j_i S \text{ are co-related indices}\}$ be an index set of co-related client indices where $\mu \in [\tilde{q}]$. In this experiment, we change the generation of the μ -th ciphertext set with the index set \mathbf{RI}_μ . If $\ell \leq \mu$, the ciphertexts in the ℓ -th ciphertext set with \mathbf{RI}_ℓ are changed to be random elements. Otherwise, the ciphertexts in the ℓ -th ciphertext set with \mathbf{RI}_ℓ are generated by running the normal encryption algorithm. Note that the ciphertexts with the client index j_i such that $(j_i, *) \notin S$ in $\mathbf{H}_{1,\mu-1}$ and $\mathbf{H}_{1,\mu}$ are equally generated by using random elements.

Suppose there exists an adversary \mathcal{A} that distinguishes $\mathbf{H}_{1,\mu-1}$ from $\mathbf{H}_{1,\mu}$ with non-negligible advantage. A simulator \mathcal{B} that solves the mXDH assumption using \mathcal{A} is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, g^a, \{g^{b_{i,1}}, \dots, g^{b_{i,2n}}\}_{i \in [t]})$ and $T = (\{X_{i,1}, \dots, X_{i,2n}\}_{i \in [t]})$. \mathcal{B} interacts with \mathcal{A} as follows.

Let $(st_{\mathcal{A}}, S, ((j_1, m_1), \dots, (j_q, m_q)))$ be the output of \mathcal{A} and \mathbf{RI}_μ be the target index set. The simulator \mathcal{B} first sets the secret keys of clients as follows. For each $j = j_i$, if $i \notin \mathbf{RI}_\mu$, it chooses a random exponent $s_j \in \mathbb{Z}_p$ and sets $SK_j = s_j$. Otherwise, it chooses a random exponent $s_j \in \mathbb{Z}_p$ and implicitly sets $SK_j = as_j$. Then, \mathcal{B} can generate a comparison key $CK_{j,k} = (\hat{g}^{rs_j}, \hat{g}^{rs_k})$ for each tuple $(j, k) \in S$ with the help of a random exponent $r \in \mathbb{Z}_p$, though it does not know a .

To handle hash queries, \mathcal{B} maintains a random oracle table T_H for the consistency of a simulation. This simulation is same to the proof of the Theorem 8 except that \mathcal{B} fixes some hash queries for the simulation of the ciphertext with the challenge tuple (j_i, m_i) such that $i \in \mathbf{RI}_\mu$.

To handle the creation of ciphertexts, \mathcal{B} carefully uses the hash table and the challenge elements in the assumption. Let $((j_1, m_1), \dots, (j_q, m_q))$ be the challenge tuples. If $(j_i, *) \notin S$, then \mathcal{B} creates a ciphertext by using random elements as in $\mathbf{H}_{1,\mu-1}$. If $(j_i, *) \in S$, then it means that $i \in \mathbf{RI}_\ell$ for some $\ell \in [\tilde{q}]$. \mathcal{B} creates a set of ciphertexts with the index set \mathbf{RI}_ℓ for each $\ell \in [\tilde{q}]$ as follows:

- **Case $\ell < \mu$:** \mathcal{B} creates the ciphertext for the index $i \in \mathbf{RI}_\ell$ sequentially. For the smallest index $i \in \mathbf{RI}_\ell$, it chooses random elements $R_{k,0}, R_{k,1} \in \mathbb{G}$ and com-

putes $C_{k,0} = R_{k,0}^{s_{j_i}}, C_{k,1} = R_{k,1}^{s_{j_i}}$ for all $k \in [n]$. It creates $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. For the next index i , \mathcal{B} first finds the biggest index $d = \mathbf{ind}(m_i, m_{i'})$ for any $i' < i$ and computes $s = s_{j_i}/s_{j_{i'}}$. It computes $C_{k,0} = C_{k,0}^{s'}, C_{k,1} = C_{k,1}^{s'}$ for all $k \in [d-1]$ where $CT_{j_{i'}} = (\{C'_{k,0}, C'_{k,1}\}_{k \in [n]})$. If $\mathbf{cmp}(m_i, m_{i'}) = 1$, then it chooses a random element $R_{d,0} \in \mathbb{G}$ and computes $C_{d,0} = R_{d,0}^{s_{j_i}}, C_{d,1} = C_{d,0}^{s'}$. Otherwise, it chooses a random element $R_{d,1} \in \mathbb{G}$ and computes $C_{d,0} = C_{d,1}^{s'}, C_{d,1} = R_{d,1}^{s_{j_i}}$. Next, it chooses random elements $R_{k,0}, R_{k,1} \in \mathbb{G}$ and computes $C_{k,0} = R_{k,0}^{s_{j_i}}, C_{k,1} = R_{k,1}^{s_{j_i}}$ for all $k \in [d+1, n]$. It creates the ciphertext $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. At last, it creates the ℓ -th ciphertext set $\mathbf{CT}_{\mathbf{RI}_\ell} = \{CT_{j_i}\}_{i \in \mathbf{RI}_\ell}$.

- **Case $\ell = \mu$:** \mathcal{B} creates the ciphertext for the index $i \in \mathbf{RI}_\mu$ sequentially. For the smallest index $i \in \mathbf{RI}_\mu$, \mathcal{B} computes $C_{k,0} = X_{1,2k-1}^{s_{j_i}}, C_{k,1} = X_{1,2k}^{s_{j_i}}$ for all $k \in [n]$ and creates the ciphertext $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. For the next index i , \mathcal{B} first finds the biggest index $d = \mathbf{ind}(m_i, m_{i'})$ for any $i' < i$ and it computes $s = s_{j_i}/s_{j_{i'}}$. It computes $C_{k,0} = C_{k,0}^{s'}, C_{k,1} = C_{k,1}^{s'}$ for all $k \in [d-1]$ where $CT_{j_{i'}} = (\{C'_{k,0}, C'_{k,1}\}_{k \in [n]})$. If $\mathbf{cmp}(m_i, m_{i'}) = 1$, then \mathcal{B} computes $C_{d,0} = X_{i,2d-1}^{s_{j_i}}, C_{d,1} = C_{d,0}^{s'}$ and otherwise, it computes $C_{d,0} = C_{d,1}^{s'}, C_{d,1} = X_{i,2d}^{s_{j_i}}$. Next, it computes $C_{k,0} = X_{i,2k-1}^{s_{j_i}}, C_{k,1} = X_{i,2k}^{s_{j_i}}$ for all $k \in [d+1, n]$. Then, it creates the ciphertext $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. At last, it creates the μ -th ciphertext set $\mathbf{CT}_{\mathbf{RI}_\mu} = \{CT_{j_i}\}_{i \in \mathbf{RI}_\mu}$. Note that it does not know the secret key a .
- **Case $\ell > \mu$:** It creates the ciphertext set $\mathbf{CT}_{\mathbf{RI}_\ell}$ by running the **MC-ORE.Encrypt** algorithm with hash queries.

If $T = ((g^{ab_{1,1}}, \dots, g^{ab_{1,2n}}), \dots, (g^{ab_{t,1}}, \dots, g^{ab_{t,2n}}))$, then $\mathbf{CT}_{\mathbf{RI}_\mu}$ are ciphertexts in $\mathbf{H}_{1,\mu-1}$. Otherwise, $\mathbf{CT}_{\mathbf{RI}_\mu}$ are ciphertexts in $\mathbf{H}_{1,\mu}$. By the mXDH assumption, two experiments $\mathbf{H}_{1,\mu-1}$ and $\mathbf{H}_{1,\mu}$ are computationally indistinguishable. ■

Lemma 10: The hybrid experiments \mathbf{H}_2 and \mathbf{H}_3 are indistinguishable to the polynomial-time adversary with the leakage function \mathcal{L}_S in the random oracle model.

Proof: Suppose there exists an adversary \mathcal{A} that distinguishes \mathbf{H}_2 from \mathbf{H}_3 with non-negligible advantage. We construct an efficient simulator \mathcal{B} for which the two distributions \mathbf{H}_2 and \mathbf{H}_3 are statistically indistinguishable.

Let $(st_{\mathcal{A}}, S, ((j_1, m_1), \dots, (j_q, m_q)))$ be the output of \mathcal{A} . \mathcal{B} first outputs random public parameters PP with the initial state $st_{\mathcal{B}}$. It selects a random secret key $SK_j = s_j \in \mathbb{Z}_p$ for each client index $j \in [N]$ and it can generate any comparison key $CK_{j,k}$ for $(j, k) \in S$ since it knows all secret keys.

To handle hash queries, \mathcal{B} maintains a random oracle table T_H for consistency of the simulation. If a random oracle query for $E_\beta(k, m)$ is requested for each $\beta \in \{0, 1\}$, \mathcal{B} first finds the tuple $(E_\beta(k, m), h)$ from the table T_H . If the tuple does not exist, then it chooses a random element $h \in \mathbb{G}$ and adds the tuple $(E_\beta(k, m), h)$ to T_H . Finally it gives h to \mathcal{A} as a response.

To handle the creation of ciphertexts, \mathcal{B} also maintains a ciphertext table T_{CT} for consistency of the simulation. Let $I = (j_1, \dots, j_q)$ be a tuple of challenge client index. For all $j_i \in I$ such that $(j_i, *) \notin S$, let $j_1^*, \dots, j_{q_1}^* \in I$ be distinct client indices and $\mathbf{SI}_\mu = \{i \in [q] : j_i = j_\mu^*\}$ be an index set of same client indices where $\mu \in [\tilde{q}_1]$. For all $j_i \in I$ such that $(j_i, *) \in S$, let $\mathbf{RI}_\mu = \{i \in [q] : j_i \text{ is co-related indices}\}$ be an index set of co-related client indices where $\mu \in [\tilde{q}_2]$. \mathcal{B} simulates the creation of a set of ciphertexts with a client index set \mathbf{SI}_μ or \mathbf{RI}_μ by using $st_{\mathcal{B}}$ and $\mathcal{L}_S((j_1, m_1), \dots, (j_q, m_q))$ as follows:

- For the creation of the ciphertexts with each set \mathbf{SI}_μ , \mathcal{B} initiates the ciphertext table T_{CT} . For the smallest index $i \in \mathbf{SI}_\mu$, \mathcal{B} chooses random elements $(c_{k,0}, c_{k,1}) \in \mathbb{G} \times \mathbb{G}$ and sets $(C_{k,0}, C_{k,1}) = (c_{k,0}, c_{k,1})$ for all $k \in [n]$. It adds the tuple $(i, (c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1}))$ to T_{CT} and creates $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. For the next index $i \in \mathbf{SI}_\ell$, it creates the ciphertext sequentially as follows. It first finds the biggest index $b = \mathbf{ind}(m_i, m_{i'})$ for any $i' < i$ and then finds a tuple $(i', (c'_{1,0}, c'_{1,1}), \dots, (c'_{n,0}, c'_{n,1}))$ from the table T_{CT} . If $b = n + 1$, it sets $(c_{k,0}, c_{k,1}) = (c'_{k,0}, c'_{k,1})$ for all $k \in [n]$. If not, it proceeds the following steps 1) – 3):
 - 1) It sets $(c_{k,0}, c_{k,1}) = (c'_{k,0}, c'_{k,1})$ for all $k \in [b - 1]$.
 - 2) It chooses random elements $(c_{k,0}, c_{k,1}) \in \mathbb{G} \times \mathbb{G}$ for all $k \in [b + 1, n]$.
 - 3) If $\mathbf{cmp}(m_i, m_{i'}) = 1$, it sets $c_{b,1} = c'_{b,0}$ and chooses a random element $c_{b,0} \in \mathbb{G}$. Otherwise, it sets $c_{b,0} = c'_{b,1}$ and chooses a random element $c_{b,1} \in \mathbb{G}$.

Then, \mathcal{B} creates $CT_{j_i} = (\{c_{k,0}, c_{k,1}\}_{k \in [n]})$ and adds the tuple $(i, (c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1}))$ to T_{CT} . At last, it creates the ciphertext set $\mathbf{CT}_{SI_\mu} = (\{CT_{j_i}\}_{i \in SI_\mu})$.

- For the creation of the ciphertext with each set \mathbf{RI}_μ , \mathcal{B} initiates the ciphertext table T_{CT} . For the smallest index $i \in \mathbf{RI}_\mu$, \mathcal{B} chooses random elements $(c_{k,0}, c_{k,1}) \in \mathbb{G} \times \mathbb{G}$ and computes $(C_{k,0}, C_{k,1}) = (c_{k,0}^{s_{j_i}}, c_{k,1}^{s_{j_i}})$ for all $k \in [n]$. It adds the tuple $(i, (c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1}))$ to T_{CT} and creates $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. For the next index $i \in \mathbf{RI}_\mu$, it creates the ciphertext sequentially as follows. It first finds the biggest index $b = \mathbf{ind}(m_i, m_{i'})$ for any $i' < i$ and then finds a tuple $(i', (c'_{1,0}, c'_{1,1}), \dots, (c'_{n,0}, c'_{n,1}))$ from the table T_{CT} . If $b = n + 1$, it sets $(c_{k,0}, c_{k,1}) = (c'_{k,0}, c'_{k,1})$ for all $k \in [n]$. If not, it proceeds the steps 1) – 3) described in the creation of the \mathbf{CT}_{SI_μ} . Then, \mathcal{B} computes $(C_{k,0}, C_{k,1}) = (c_{k,0}^{s_{j_i}}, c_{k,1}^{s_{j_i}})$ for all $k \in [n]$. It creates $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$ and adds the tuple $(i, (c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1}))$ to T_{CT} . At last, it creates the ciphertext set $\mathbf{CT}_{RI_\mu} = (\{CT_{j_i}\}_{i \in RI_\mu})$.

CORRECTNESS OF THE SIMULATION

To show the correctness of the simulation, we prove that the distributions $((\mathbf{CT}_{SI_1}, \dots, \mathbf{CT}_{SI_{\tilde{q}_1}}), (\mathbf{CT}_{RI_1}, \dots, \mathbf{CT}_{RI_{\tilde{q}_2}}))$ and $((\overline{\mathbf{CT}}_{SI_1}, \dots, \overline{\mathbf{CT}}_{SI_{\tilde{q}_1}}), (\overline{\mathbf{CT}}_{RI_1}, \dots, \overline{\mathbf{CT}}_{RI_{\tilde{q}_2}}))$ of the

ciphertexts output in H_2 and H_3 are statistically indistinguishable and the outputs of random oracle are properly simulated. We have to show that the following conditions hold.

- $\forall \ell \in [\tilde{q}_1], \forall \ell' \in [\tilde{q}_2], \mathbf{CT}_{SI_\ell}$ and $\mathbf{CT}_{RI_{\ell'}}$ are distributed independently.
- $\forall \ell \in [\tilde{q}_1], \forall \ell' \in [\tilde{q}_2], \mathbf{CT}_{SI_\ell} \equiv \overline{\mathbf{CT}}_{SI_\ell}$ and $\mathbf{CT}_{RI_{\ell'}} \equiv \overline{\mathbf{CT}}_{RI_{\ell'}}$.

The first condition is simply proved since each ciphertext for SI_ℓ and $RI_{\ell'}$ are simulated independently. Next, we use induction to prove that the second condition holds. For each $\ell \in [\tilde{q}_1]$ (or $\ell \in [\tilde{q}_2]$), let $\mathbf{CT}_{SI_\ell} = (CT_1, \dots, CT_t)$ and $\overline{\mathbf{CT}}_{SI_\ell} = (\overline{CT}_1, \dots, \overline{CT}_t)$ (or $\mathbf{CT}_{RI_\ell} = (CT_1, \dots, CT_t)$ and $\overline{\mathbf{CT}}_{RI_\ell} = (\overline{CT}_1, \dots, \overline{CT}_t)$). Obviously, the statement is true for $i = 1$. Assume that it is true for $i - 1$ and we prove that $(CT_1, \dots, CT_i) \equiv (\overline{CT}_1, \dots, \overline{CT}_i)$. The detailed description is provided in Appendix.

In addition, suppose that the tuple $(E_0(k, m), h)$ is in T_H and $(i, (c_{1,0}, c_{1,1}), \dots, (c_{n,0}, c_{n,1}))$ is in T_{CT} for some i such that $m_i = m$. By the Lemmas 8 and 9, \mathcal{A} can not find out that h and $c_{k,0}$ are different. This completes the correctness of simulation. ■

E. EXTENSIONS

We present several extensions of our basic MC-ORE scheme to overcome their shortcomings.

1) REDUCING TRUST ON THE CENTER

The basic MC-ORE scheme has the problem that a center should be fully trusted because it generates the secret keys of individual clients and comparison keys of different clients. The existence of a trusted center is very strong constraint and it is costly to ensure the security of such a center in reality. One way to reduce trust on the center is that each client himself selects a secret key and securely transfers the corresponding information to the center instead of having the center owns the secret keys. That is, each client chooses its secret key s_j and securely sends \hat{g}^{s_j} to the center, and then the center can generate a comparison key $CK = ((\hat{g}^{s_j})^r, (\hat{g}^{s_k})^r)$ by using $\hat{g}^{s_j}, \hat{g}^{s_k}$ received from clients and a random exponent r . In this case, the center only can generate comparison keys, but it can not generate client's ciphertexts since it does not have the secret keys of individual clients.

2) REMOVING THE TRUSTED CENTER

Unlike the previous ORE schemes, our basic MC-ORE scheme requires a center to generate secret keys of individual clients and comparison keys between different clients. Although we suggested a method to reduce trust on the center, we cannot remove the ability of the center to generate comparison keys. Note that if a comparison key is exposed, a malicious client can compare any ciphertexts between two clients by using the exposed comparison key. One idea to securely generate a comparison key even after the center is completely removed is that two clients perform

a cryptographic protocol to share the same random value \hat{g}^r which is used to create \hat{g}^{rs_j} and \hat{g}^{rs_k} . The simplest way to non-interactively share the random value is to use a hash function. That is, two clients with indices j and k generate $H(j||k)^{s_j}$ and $H(j||k)^{s_k}$ respectively, and transmit these values to a third client. Note that these values are a valid comparison key since $H(j||k)$ corresponds to \hat{g}^r for some random exponent r .

IV. ENHANCED MC-ORE CONSTRUCTION

In this section, we propose our second construction of MC-ORE with reduced leakage and prove the ST-SIM security of our scheme.

A. CONSTRUCTION

In the basic MC-ORE scheme, both ciphertext comparisons in a single client and between different clients leak the most significant differing bit as well as the result of the comparison. Although there are some ORE schemes with reduced leakage [8], [10], it is difficult to extend those schemes to support comparisons on ciphertexts generated by different clients. To build an MC-ORE scheme with reduced leakage, we divide the ciphertext into independent two parts such that the first part only supports ciphertext comparisons in a single client, and the second part only supports ciphertext comparisons between different clients. For the first part, we use any ORE scheme with reduced leakage. For the second part, we construct an encrypted ORE (EORE) scheme by modifying our basic MC-ORE scheme so that it can not be used for ciphertext comparisons in a single client. If the second part has no leakage until a comparison key is provided, only the reduced leakage of the ORE scheme affects the overall leakage.

1) ENCRYPTED ORE

We first construct an EORE scheme by modifying our basic MC-ORE scheme. The syntax of EORE is very similar to that of MC-ORE defined in Definition 2 except that the comparison algorithm is excluded. The ciphertext of the EORE scheme is created by first generating a ciphertext of the basic MC-ORE scheme and then encrypting it with a public-key encryption scheme. The comparison key of the EORE scheme includes additional elements that decrypt the encrypted ciphertext to obtain the comparison form of the basic MC-ORE scheme. The ciphertext comparison is performed in a similar manner to the basic MC-ORE scheme.

Let $S = \{(j, k)\}_{j,k \in [N]}$ be a set of index tuples where the comparison key $CK_{j,k}$ is revealed. A leakage function \mathcal{L}_S^{EORE} is defined as follows:

$$\mathcal{L}_S^{EORE}((j_1, m_1), \dots, (j_q, m_q)) = \{\mathbf{cmp}(m_{j_{i'}}, m_i), \mathbf{ind}(m_{j_{i'}}, m_i) : 1 \leq i' < i \leq q, (j_{i'}, j_i) \in S\}.$$

Our EORE scheme with leakage \mathcal{L}_S^{EORE} is given as follows:

EORE.Setup($1^\lambda, N$). This algorithm first generates bilinear groups $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ of prime order p with group generators $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$. It chooses ran-

dom exponents $s_j, a_j \in \mathbb{Z}_p$ and computes $h_j = g^{a_j}$ and $\hat{h}_j = \hat{g}^{a_j}$ for all $j \in [N]$. It outputs a master key $MK = (\{s_j, \hat{h}_j\}_{j \in [N]})$ and public parameters $PP = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, \{h_j\}_{j \in [N]}, H)$ where $H : \{0, 1\}^* \rightarrow \mathbb{G}$ is a full-domain hash function.

EORE.GenKey(j, MK, PP). Let $MK = (\{s_1, \dots, s_N\}, \{\hat{h}_1, \dots, \hat{h}_N\})$. It outputs a secret key $SK_j = s_j$.

EORE.Encrypt(m, SK_j, PP). Let $m = x_1 x_2 \dots x_n \in \{0, 1\}^n$ and $SK_j = s_j$. For each $i \in [n]$, it computes $F_{i,0} = H(\mathbf{prefix}(m, i) || 0x_i)^{s_j}$ and $F_{i,1} = H(\mathbf{prefix}(m, i) || (0 x_i + 1))^{s_j}$. For each $F_{i,b}$, it selects a random exponent $t \in \mathbb{Z}_p$ and computes $C_{i,b,0} = F_{i,b} h_j^t$ and $C_{i,b,1} = g^t$. It outputs a ciphertext $CT_j = (\{C_{i,b,0}, C_{i,b,1}\}_{i \in [n], b \in \{0,1\}})$.

EORE.GenCmpKey(j, k, MK, PP). Let s_j and s_k be the secret keys of client indices j and k . It chooses a random exponent $r \in \mathbb{Z}_p$ and computes $K_{0,0} = \hat{g}^{rs_j}$, $K_{0,1} = \hat{h}_k^{rs_j}$ and $K_{1,0} = \hat{g}^{rs_k}$, $K_{1,1} = \hat{h}_j^{rs_k}$. It outputs the comparison key $CK_{j,k} = (\{K_{b,0}, K_{b,1}\}_{b \in \{0,1\}})$.

EORE.CompareMC($CT_j, CT'_k, CK_{j,k}, PP$). Let $CT_j = (\{C_{i,b,0}, C_{i,b,1}\})$ and $CT'_k = (\{C'_{i,b,0}, C'_{i,b,1}\})$ for $i \in [n]$ and $b \in \{0, 1\}$. Let $CK_{j,k} = (\{K_{b,0}, K_{b,1}\}_{b \in \{0,1\}})$. It first finds the smallest index i^* such that $e(C_{i^*,0,0}, K_{1,0}) / e(C_{i^*,0,1}, K_{1,1}) \neq e(C'_{i^*,0,0}, K_{0,0}) / e(C'_{i^*,0,1}, K_{0,1})$ by sequentially comparing these values from an index 0 to n . If such index i^* exists and $e(C_{i^*,1,0}, K_{1,0}) / e(C_{i^*,1,1}, K_{1,1}) = e(C'_{i^*,0,0}, K_{0,0}) / e(C'_{i^*,0,1}, K_{0,1})$ holds, it outputs 1. If such index i^* exists and $e(C_{i^*,0,0}, K_{1,0}) / e(C_{i^*,0,1}, K_{1,1}) = e(C'_{i^*,1,0}, K_{0,0}) / e(C'_{i^*,1,1}, K_{0,1})$, then it outputs 0. If no such index i^* exists, then it outputs 0.

Remark 11: The leakage function \mathcal{L}_S^{EORE} is same to the leakage function \mathcal{L}_S of the basic MC-ORE scheme except that it excludes the condition $j_{i'} = j_i$. It means that the basic MC-ORE scheme leaks the comparison result between ciphertexts of a single client, but the EORE scheme does not leak any information before the comparison key is revealed.

2) MULTI-CLIENT ORE

Now we construct an enhanced MC-ORE scheme by composing any ORE scheme with reduced leakage and the above EORE scheme. As mentioned before, the ciphertext of the enhanced MC-ORE scheme consists of two parts such that the first part is created from the ORE scheme and the second part is created from the EORE scheme.

Let \mathcal{L}_j^{ORE} be the leakage function of the underlying ORE scheme corresponding to the client index j and \mathcal{L}_S^{EORE} be the leakage function of our EORE scheme. A leakage function \mathcal{L}_S^{MC-ORE} is defined as follows:

$$\mathcal{L}_S^{MC-ORE}((j_1, m_1), \dots, (j_q, m_q)) = \{\mathcal{L}_j^{ORE}(m_{i_1}, \dots, m_{i_\rho}) \cup \mathcal{L}_S^{EORE} : j = j_{i_1} = \dots = j_{i_\rho}\}.$$

where the sequence sets $M_j = \{m_{i_1}, \dots, m_{i_\rho}\}$ satisfy $\bigcap M_j = \emptyset$ and $\bigcup M_j = \{m_1, \dots, m_q\}$. Here, if $S = \emptyset$, meaning that

any comparison key is not revealed, then \mathcal{L}_S^{MC-ORE} becomes equal to the reduced leakage functions $\{\mathcal{L}_j^{ORE}\}$ for each j . Otherwise, if $S \neq \emptyset$, to achieve reducing the leakage, the ORE scheme is restricted from having no leakage beyond the leakage of the EORE scheme for the same client. That is, \mathcal{L}_S^{MC-ORE} will be at most \mathcal{L}_S^{EORE} . Our MC-ORE scheme with leakage \mathcal{L}_S^{MC-ORE} that combines an ORE scheme and our EORE scheme is described as follows:

- MC-ORE.Setup**($1^\lambda, N$). It obtains MK_{EORE} and PP_{EORE} by running **EORE.Setup**($1^\lambda, N$) and outputs $MK = MK_{EORE}$ and $PP = PP_{EORE}$.
- MC-ORE.GenKey**(j, MK, PP). It runs **ORE.Setup**(1^λ) and **EORE.GenKey**(j, MK, PP) to obtain $SK_{ORE,j}$ and $SK_{EORE,j}$, respectively. It outputs a secret key $SK_j = (SK_{ORE,j}, SK_{EORE,j})$.
- MC-ORE.Encrypt**(m, SK_j, PP). Let $SK_j = (SK_{ORE,j}, SK_{EORE,j})$. It first obtains OC_j and EC_j by running **ORE.Encrypt**($m, SK_{ORE,j}$) and **EORE.Encrypt**($m, SK_{EORE,j}, PP$) respectively. It outputs a ciphertext $CT_j = (OC_j, EC_j)$.
- MC-ORE.Compare**(CT_j, CT'_j, PP). Let $CT_j = (OC_j, EC_j)$ and $CT'_j = (OC'_j, EC'_j)$ for the same client index j . It returns **ORE.Compare**(OC_j, OC'_j).
- MC-ORE.GenCmpKey**(j, k, MK, PP). Let SK_j and SK_k be the secret keys for the client indices j and k . It outputs the comparison key $CK_{j,k}$ by running **EORE.GenCmpKey**(j, k, MK, PP).
- MC-ORE.CompareMC**($CT_j, CT'_k, CK_{j,k}, PP$). Let $CT_j = (OC_j, EC_j)$ and $CT'_k = (OC'_k, EC'_k)$. It returns the result of **EORE.CompareMC**($EC_j, EC'_k, CK_{j,k}, PP$).

B. CORRECTNESS

For the ciphertext comparisons in a single client, the correctness follows from that of the underlying ORE scheme. For the ciphertext comparisons between different clients, the correctness is shown as follows. Let $SK_j = s_j$ and $SK_k = s_k$ be the secret keys of client indices j and k , and $CK_{j,k} = (K_{0,0}, K_{0,1}, K_{1,0}, K_{1,1}) = (\hat{g}^{rs_j}, \hat{h}_k^{rs_j}, \hat{g}^{rs_k}, \hat{h}_j^{rs_k})$ be the comparison key of (j, k) . Let $EC_j = (\{C_{i,b,0}, C_{i,b,1}\}_{i \in [n], b \in \{0,1\}})$ and $EC'_k = (\{C'_{i,b,0}, C'_{i,b,1}\}_{i \in [n], b \in \{0,1\}})$ be ciphertexts on messages m and m' . If $m < m'$, there must be a smallest index i^* such that $x_i = x'_i$ for all $i < i^*$ and $x_{i^*} \neq x'_{i^*}$. Then we have that

$$\begin{aligned}
 & e(C_{i,0,0}, K_{1,0})/e(C_{i,0,1}, K_{1,1}) \\
 &= e(H(E_0(i, m))^{s_j} \hat{h}_j^t, \hat{g}^{rs_k})/e(g^t, \hat{h}_j^{rs_k}) \\
 &= e(H(E_0(i, m'))^{s_k} \hat{h}_k^t, \hat{g}^{rs_j})/e(g^t, \hat{h}_k^{rs_j}) \\
 &= e(C'_{i,0,0}, K_{0,0})/e(C'_{i,0,1}, K_{0,1}) \quad \forall i < i^*, \\
 &\quad \times e(C_{i^*,1,0}, K_{1,0})/e(C_{i^*,1,1}, K_{1,1}) \\
 &= e(H(E_1(i^*, m))^{s_j} \hat{h}_j^t, \hat{g}^{rs_k})/e(g^t, \hat{h}_j^{rs_k}) \\
 &= e(H(E_0(i^*, m'))^{s_k} \hat{h}_k^t, \hat{g}^{rs_j})/e(g^t, \hat{h}_k^{rs_j}) \\
 &= e(C'_{i^*,0,0}, K_{0,0})/e(C'_{i^*,0,1}, K_{0,1}).
 \end{aligned}$$

C. SECURITY ANALYSIS

We now prove the security of the enhanced MC-ORE scheme with the leakage function \mathcal{L}_S^{MC-ORE} in the ST-SIM security model. We begin by giving a high-level overview of the security proof. We define a sequence of experiments from \mathbf{H}_0 corresponding to the real experiment to \mathbf{H}_4 corresponding to the ideal experiment and show that the outputs of two experiments are indistinguishable. Since the ciphertext is divided into two parts: the ORE ciphertext OC , and the EORE ciphertext EC , the hybrid experiments are also defined separately. At first, the ORE ciphertexts are simulated only with the leakage functions \mathcal{L}_j^{ORE} . In the next experiment, the EORE ciphertexts of clients whose comparison keys are not exposed are randomly generated. Then, in the next experiment, the EORE ciphertexts of clients whose comparison keys are exposed are generated with random values. Finally, in the last experiment \mathbf{H}_4 , the EORE ciphertexts of clients whose comparison keys are exposed are simulated with respect to the leakage function \mathcal{L}_S^{EORE} , and consequently \mathbf{H}_4 corresponds to the ideal experiment. The details are given as follows.

Theorem 12: The enhanced MC-ORE scheme is ST-SIM secure with the leakage function \mathcal{L}_S^{MC-ORE} in the random oracle model if the ORE scheme is SIM secure with the leakage function \mathcal{L}^{ORE} , the basic MC-ORE scheme is ST-SIM secure with the leakage function \mathcal{L}_S , and the XDH assumption holds.

Proof: We prove the security of our enhanced MC-ORE scheme through a sequence of hybrid experiments. The first experiment is defined as the real MC-ORE security experiment and the last one is defined as the ideal experiment with the leakage function \mathcal{L}_S^{MC-ORE} in which the adversary has no advantage. The hybrid experiments $\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3$, and \mathbf{H}_4 are defined as follows:

- \mathbf{H}_0 : This experiment corresponds to the real world experiment.
- \mathbf{H}_1 : In this experiment, the ORE ciphertexts OC_j are generated with the leakage function \mathcal{L}_j^{ORE} and the rest are same to \mathbf{H}_0 . We have that \mathbf{H}_0 and \mathbf{H}_1 are indistinguishable if the underlying ORE scheme is secure with respect to the leakage function \mathcal{L}^{ORE} .
- \mathbf{H}_2 : This experiment is similar to \mathbf{H}_1 except that the EORE ciphertext EC_j such that $(j, j') \notin S$ for any client index j' is generated by using random elements.
- \mathbf{H}_3 : This experiment is similar to \mathbf{H}_2 except that the EORE ciphertext EC_j such that $(j, j') \in S$ for some client indices j' is generated by using random elements.
- \mathbf{H}_4 : In this experiment, the EORE ciphertext EC_j such that $(j, j) \in S$ for some client indices j is generated with the leakage function \mathcal{L}_S^{EORE} and the rest are same to \mathbf{H}_3 . This experiment corresponds to the ideal world experiment.

From the following Lemmas 13, 14, 15, and 16 that claim the indistinguishability of the experiments, we have that \mathbf{H}_0 and \mathbf{H}_4 are computationally indistinguishable. ■

Lemma 13: The hybrid experiments \mathbf{H}_0 and \mathbf{H}_1 are computationally indistinguishable to the polynomial-time adversary if the underlying ORE scheme is SIM secure with the leakage function \mathcal{L}^{ORE} .

Proof: The proof of this lemma is simple since a ciphertext CT_j consists of two independent part OC_j and EC_j . A simulator can use the simulator of the ORE scheme for the generation of OC_j and it can generate other elements in EC_j by the randomly chosen master key of an EORE scheme. ■

Lemma 14: The hybrid experiments \mathbf{H}_1 and \mathbf{H}_2 are computationally indistinguishable to the polynomial-time adversary assuming that the mXDH assumption holds.

Proof: To prove this lemma, we define a sequence of hybrid experiments $\mathbf{H}_1 = \mathbf{H}_{1,0}, \mathbf{H}_{1,1}, \dots, \mathbf{H}_{1,q} = \mathbf{H}_2$ as follows.

$\mathbf{H}_{1,\mu}$: In this experiment, we change the generation of the μ -th ciphertext if $(j_\mu, *) \notin S$. If $i \leq \mu$ and $(j_i, *) \notin S$, the i -th EORE ciphertext EC_{j_i} is generated by using random elements. Otherwise, the i -th EORE ciphertext EC_{j_i} is generated by running the normal encryption algorithm. Note that $\mathbf{H}_{1,\mu-1}$ and $\mathbf{H}_{1,\mu}$ are trivially equal if $(j_i, *) \in S$.

Without loss of generality, we assume that $(j_\mu, *) \notin S$. Suppose there exists an adversary \mathcal{A} that distinguishes $\mathbf{H}_{1,\mu-1}$ from $\mathbf{H}_{1,\mu}$ with non-negligible advantage. A simulator \mathcal{B} that solves the mXDH assumption using \mathcal{A} is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, g^a, g^{b_1}, \dots, g^{b_{2n}})$ and $T = (X_1, \dots, X_{2n})$. \mathcal{B} interacts with \mathcal{A} as follows.

Let $(st_{\mathcal{A}}, S, ((j_1, m_1), \dots, (j_q, m_q)))$ be the output of \mathcal{A} . The simulator \mathcal{B} first sets the public parameters corresponding to the client index. For each $j \in [N]$, if $j \neq j_\mu$, \mathcal{S} chooses a random exponent $\alpha_j \in \mathbb{Z}_p$ and computes $h_j = g^{\alpha_j}$. For the target client j_μ , it sets $h_{j_\mu} = g^a$. Next, for each $j \in [N]$, \mathcal{B} chooses a random exponent $s_j \in \mathbb{Z}_p$ and sets the secret key $SK_j = s_j$. It can generate the comparison key $CK_{j,k}$ for any tuple $(j, k) \in S$ since it knows secret keys s_j and s_k .

Let (j_i, m_i) be the i -th ciphertext query for a client index j_i . Let $E_0(k, m) = \text{prefix}(m, k) \parallel 0x_k$ and $E_1(k, m) = \text{prefix}(m, k) \parallel (0x_k + 1)$ be encoded messages where $m = x_1 \dots x_n \in \{0, 1\}^n$. If $(j_i, *) \in S$, then \mathcal{B} simply creates a ciphertext by running the **EORE.Encrypt** algorithm since it know the secret key s_{j_i} . If $(j_i, *) \notin S$, then \mathcal{B} creates the i -th EORE ciphertext EC_{j_i} as follows:

- **Case $i < \mu$:** It chooses random elements $R_{k,0} = (R_{k,0,0}, R_{k,1,0}), R_{k,1} = (R_{k,0,1}, R_{k,1,1}) \in \mathbb{G} \times \mathbb{G}$ for all $k \in [n]$ and creates $EC_{j_i} = (\{R_{k,0}, R_{k,1}\}_{k \in [n]})$.
- **Case $i = \mu$:** For each $\beta \in \{0, 1\}$, it computes $F_{k,\beta} = H(E_\beta(k, m_i))^{s_{j_i}}$ for all $k \in [n]$. For each $F_{k,0}$ and $F_{k,1}$, it sets $C_{k,0} = (F_{k,0} \cdot X_{2k-1}, g^{b_{2k-1}})$ and $C_{k,1} = (F_{k,1} \cdot X_{2k}, g^{b_{2k}})$ and creates $EC_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$.
- **Case $i > \mu$:** It creates the EORE ciphertext EC_{j_i} by running the **EORE.Encrypt** algorithm.

If $T = (g^{ab_1}, \dots, g^{ab_{2n}})$, then EC_{j_μ} is a ciphertext in $\mathbf{H}_{1,\mu-1}$. Otherwise, EC_{j_μ} is a ciphertext in $\mathbf{H}_{1,\mu}$. By the mXDH

assumption, two experiments $\mathbf{H}_{1,\mu-1}$ and $\mathbf{H}_{1,\mu}$ are computationally indistinguishable. ■

Lemma 15: The hybrid experiments \mathbf{H}_2 and \mathbf{H}_3 are computationally indistinguishable to the polynomial-time adversary assuming that the mXDH assumption holds.

Proof: We additionally define a sequence of hybrid experiments $\mathbf{H}_2 = \mathbf{H}_{2,0}, \mathbf{H}_{2,1}, \dots, \mathbf{H}_{2,\tilde{q}} = \mathbf{H}_3$ for some \tilde{q} as follows.

$\mathbf{H}_{2,\mu}$: Let $I = (j_1, \dots, j_q)$ be a tuple of challenge client index and $\mathbf{RI}_\mu = \{i \in [q] : j_i \text{ is co-related indices}\}$ be an index set of co-related client indices where $\mu \in [\tilde{q}]$. In this experiment, we change the generation of the μ -th EORE ciphertext set with the index set \mathbf{RI}_μ . If $\ell \leq \mu$, the EORE ciphertexts in the ℓ -th ciphertext set with \mathbf{RI}_ℓ are changed to be random elements. Otherwise, the ciphertexts in the ℓ -th ciphertext set with \mathbf{RI}_ℓ are generated by running the normal encryption algorithm. Note that the ciphertexts with the client index j_i such that $(j_i, *) \notin S$ in $\mathbf{H}_{2,\mu-1}$ and $\mathbf{H}_{2,\mu}$ are equally generated by using random elements.

Suppose there exists an adversary \mathcal{A} that distinguishes $\mathbf{H}_{2,\mu-1}$ from $\mathbf{H}_{2,\mu}$ with non-negligible advantage. A simulator \mathcal{B} that solves the mXDH assumption using \mathcal{A} is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, g^a, \{g^{b_{i,1}}, \dots, g^{b_{i,2n}}\}_{i \in [t]})$ and $T = (\{X_{i,1}, \dots, X_{i,2n}\}_{i \in [t]})$. \mathcal{B} runs the simulator $\mathcal{B}_{bMC-ORE}$ of the Lemma 9 as a subsimulator by submitting the challenge tuple of the mXDH assumption. Then \mathcal{B} that interacts with \mathcal{A} is described as follows.

Let $(st_{\mathcal{A}}, S, ((j_1, m_1), \dots, (j_q, m_q)))$ be the output of \mathcal{A} . The simulator \mathcal{B} first sets the public parameters corresponding to the client index. For each $j \in [N]$, \mathcal{S} chooses a random exponent $\alpha_j \in \mathbb{Z}_p$ and computes $h_j = g^{\alpha_j}$. For each tuple $(j, k) \in S$, \mathcal{B} obtains $CK'_{j,k} = (K_0, K_1)$ by running $\mathcal{B}_{bMC-ORE}$ and computes the comparison key $CK_{j,k} = (K_0, K_0^{\alpha_k}, K_1, K_1^{\alpha_j})$.

For the creation of the EORE ciphertexts with the client index j_i for $i \in \mathbf{RI}_\ell$, \mathcal{B} first runs $\mathcal{B}_{bMC-ORE}$ and obtains $CT'_{j_i} = (\{F_{k,0}, F_{k,1}\}_{k \in [n]})$. For each $F_{k,b}$, it chooses a random exponent $t \in \mathbb{Z}_p$ and computes $C_{k,b} = (F_{k,b} \cdot h_{j_i}^t, g^t)$ where $b \in \{0, 1\}$. It creates $EC_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$ and hence creates the EORE ciphertext set $\mathbf{EC}_{\mathbf{RI}_\ell} = (\{EC_{j_i}\}_{i \in \mathbf{RI}_\ell})$

By the Lemma 9, two experiments $\mathbf{H}_{2,\mu-1}$ and $\mathbf{H}_{2,\mu}$ are computationally indistinguishable. ■

Lemma 16: The hybrid experiments \mathbf{H}_3 and \mathbf{H}_4 are indistinguishable to the polynomial-time adversary with the leakage function \mathcal{L}_S^{EORE} in the random oracle model.

Proof: Suppose there exists an adversary \mathcal{A} that distinguishes \mathbf{H}_3 from \mathbf{H}_4 with non-negligible advantage. We construct an efficient simulator \mathcal{B} for which the two distributions \mathbf{H}_3 and \mathbf{H}_4 are statistically indistinguishable. \mathcal{B} runs the simulator $\mathcal{B}_{bMC-ORE}$ of the Lemma 10 as a subsimulator.

Let $(st_{\mathcal{A}}, S, ((j_1, m_1), \dots, (j_q, m_q)))$ be the output of \mathcal{A} . The simulator \mathcal{B} first sets the public parameters corresponding to the client index. For each $j \in [N]$, \mathcal{B} chooses a

TABLE 1. Performance comparison between our MC-ORE schemes.

Scheme	Encrypt (ms)	Compare (μs)	CompareMC (ms)	$ CT $	$ CK $
Basic MC-ORE	45.8	1.65	35.6	$2n \mathbb{G} $	$2 \hat{\mathbb{G}} $
Encrypted ORE	107.4	-	58.2	$> 4n \mathbb{G} $	$4 \hat{\mathbb{G}} $

random exponent $\alpha_j \in \mathbb{Z}_p$ and computes $h_j = g^{\alpha_j}$. For each tuple $(j, k) \in S$, \mathcal{B} obtains $CK'_{j,k} = (K_0, K_1)$ by running $\mathcal{B}_{bMC-ORE}$ and computes the comparison key $CK_{j,k} = (K_0, K_0^{\alpha_k}, K_1, K_1^{\alpha_j})$.

For the generation of the i -th EORE ciphertext EC_{j_i} with the client index j_i , \mathcal{B} first runs $\mathcal{B}_{bMC-ORE}$ and obtains $CT'_{j_i} = (\{F_{k,0}, F_{k,1}\}_{k \in [n]})$. For each $F_{k,b}$, it chooses a random exponent $t \in \mathbb{Z}_p$ and computes $C_{k,b} = (F_{k,b} \cdot h_{j_i}^t, g^t)$ where $b \in \{0, 1\}$. It creates $EC_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$.

By the Lemma 10, the distributions $(CT'_{j_1}, \dots, CT'_{j_q})$ and $(\overline{CT'_{j_1}}, \dots, \overline{CT'_{j_q}})$ of the basic MC-ORE ciphertexts output in \mathbf{H}_2 and \mathbf{H}_3 of the Theorem 6 are indistinguishable. Thus, it can be easily derived that the distributions $(EC_{j_1}, \dots, EC_{j_q})$ and $(\overline{EC_{j_1}}, \dots, \overline{EC_{j_q}})$ in \mathbf{H}_3 and \mathbf{H}_4 are also indistinguishable. Since OC_j and EC_j have been generated with respect to \mathcal{L}_j^{ORE} and \mathcal{L}_S^{EORE} , \mathbf{H}_4 corresponds to the ideal experiment. ■

V. IMPLEMENTATION

In this section, we measure the performance of our MC-ORE schemes and compare various ciphertext comparison methods. Our implementation is entirely written in C and employs a 224-bit MNT curves from the PBC library for pairing operations. We run our implementation on a laptop with 4GHz Intel Core i7-6700K CPU and 16GB RAM.

A. PERFORMANCE OF MC-ORE

We evaluate the runtime of **Encrypt**, **Compare**, and **CompareMC** algorithms for 32-bit integers and the benchmarks averaged over 100 iterations are given in Table 1. Compared to the basic MC-ORE scheme, the encrypted ORE scheme takes more time to run each algorithm and the size of the ciphertext and the comparison key is about twice as large. The reason why the encrypted ORE scheme is less efficient is that its **Encrypt** algorithm runs the **Encrypt** algorithm of the basic MC-ORE scheme as a subalgorithm and the **CompareMC** algorithm requires twice as many pairing operations as the basic MC-ORE scheme. This shows that although the security of the MC-ORE scheme is improved by reducing the leakage, at the same time, the efficiency is decreased. We note that the runtime of the **Compare** algorithm and the accurate size of the ciphertext of the enhanced MC-ORE scheme depend on the underlying ORE scheme.

B. RANGE QUERY METHODS

One possible application of MC-ORE is a range query for an encrypted database, in which case a database server must perform the multi-client comparison algorithm many times

to find a subset of database that satisfies the range query. Suppose that a database server keeps each client database $D_j \in [R]^M$ that store ciphertexts generated by a client with index j where the database consists of maximum M values each in the range $[R]$. A client with an index k may request a range query by giving CT' on a plaintext m' encrypted with SK_k to find a subset of ciphertexts in D_j less than m' . If the server has a comparison key $CK_{j,k}$, then it can answer the query by simply running the multi-client comparison algorithm M times. However, this naive method is very slow since M multi-client comparison operations are required. Therefore, we need better methods to handle range queries by using comparison operations more efficiently.

We present two additional methods and compare these methods with the simple method described before. The detailed explanation of each method is given as follows.

- **Simple Method.** The simple method simply runs the **CompareMC** algorithm M times. Recall that the **CompareMC** algorithm tries to find the MSDB from the higher bit to the lower bit sequentially. If the MSDB is located in higher bits, then the comparison operation is considerably efficient. On the other hand (if the MSDB is located in lower bits), the comparison operation is relatively slow.
- **BinSearch Method.** The binary search method uses a modification of the **CompareMC** algorithm that finds the MSDB more efficiently by using binary searching instead of sequential searching. Let $CT = \{C_{i,0}, C_{i,1}\}_{i \in [n]}$ be one ciphertext in a database D_j and $CT' = \{C'_{i,0}, C'_{i,1}\}_{i \in [n]}$ be a ciphertext created by a client with k . A server with a comparison key $CK_{j,k} = (K_0, K_1)$ first checks whether $e(C_{n/2,0}, K_1)$ and $e(C'_{n/2,0}, K_0)$ are equal or not. If the values are equal, it checks again whether $e(C_{3n/4,0}, K_1)$ and $e(C'_{3n/4,0}, K_0)$ are equal since the MSDB is in the range $[n/2 + 1, n]$. On the other hand, if the values are not equal, it checks whether $e(C_{n/4,0}, K_1)$ and $e(C'_{n/4,0}, K_0)$ are equal since the MSDB is in the range $[1, n/2]$. By repeating this process $\log n$ times, the server can find the MSDB. Since the database contains at most M entries, it runs this modified comparison algorithm M times.
- **Hybrid Method.** The hybrid method uses the **CompareMC** algorithm and the **Compare** algorithm together since the **Compare** algorithm is fast and it can compare the order of ciphertexts in a database which are created by a single client. Let CT_i be a ciphertext on a message m_i in a database D_j and CT' be a ciphertext

TABLE 2. Performance comparison between range query methods.

R	Simple Method (s)	BinSearch Method (s)	Hybrid Method (s)
2^8	11.50	4.79	2.46
2^{16}	8.19	4.84	1.73
2^{24}	5.90	4.89	1.27
2^{28}	4.55	4.89	1.10
2^{32}	3.48	4.82	0.87

on a message m' given by a client in a range query. To answer the range query of the client, a server should find a subset S of ciphertexts in D_j such that $m_i < m'$. The server first compares CT' with one specific $CT_{i^*} \in D_j$ by running the **CompareMC** algorithm, and then it divides all other $CT_i \in D_j$ into two groups L and R by running the **Compare** algorithm on input CT_i and CT_{i^*} where L contains ciphertexts of $m_i < m_{i^*}$ and R contains ciphertexts of $m_{i^*} \leq m_i$. If $m_{i^*} < m'$, then the server adds L to the subset S and repeats the above process by setting $D_j = R$. If $m' < m_{i^*}$, the server repeats the above process by setting $D_j = L$.

We compared the performance of each method only for the basic MC-ORE scheme. For the comparison, we set $M = 100$ and $R \in \{2^8, 2^{16}, 2^{24}, 2^{28}, 2^{32}\}$. We randomly selected 32-bit integers m_1, \dots, m_{100} and m' within a specific range $[0, R]$, and then we encrypted m_1, \dots, m_{100} with SK and m' with SK' . The running time of the above three range query methods is given in Table 2. The binary search method executes 12 pairing operations per a single ciphertext comparison whereas the simple method performs at least 4 up to 66 pairing operations depending on the location of the MSDB. In other words, the binary search method is better than the simple method if the data are within a small range and the high-order bits are equal, but it is less efficient if the data are within a large range and the probability that the high-order bits are equal is lower. The hybrid method is always more efficient than the simple method and the binary search method, since some comparisons are performed by using the **Compare** algorithm instead of using the **CompareMC** algorithm. That is, the performance of the hybrid method is far superior because the **CompareMC** algorithm is performed for comparisons on the specific ciphertexts and the **Compare** algorithm is executed for the remaining ciphertext comparisons.

It is important to improve the performance of the algorithm, but our results show that efficiency can be improved in an appropriate way depending on the application environment. If our MC-ORE scheme is applied to an environment other than a database range query, we can consider another way to improve its performance or its security.

VI. CONCLUSION

We introduced the concept of multi-client order-revealing encryption (MC-ORE) that supports comparisons on

ciphertexts generated by multiple clients as well as generated by one client. We also defined the simulation-based security model for MC-ORE with respect to a leakage function. We then proposed two practical MC-ORE schemes with different leakage functions and proved their security in the defined security model. The first scheme leaks more information, namely the most significant differing bit, and the second scheme is the enhanced scheme with reduced leakage. We implemented our schemes to measure the performance of each algorithm and provided additional range query methods to improve the performance in a database range query.

APPENDIX

PROOF OF THE LEMMA III.6

CORRECTNESS OF THE SIMULATION

To complete the proof of the Lemma 10, we have to show that the following condition holds.

- $\forall \ell \in [\tilde{q}_1], \forall \ell' \in [\tilde{q}_2], \mathbf{CT}_{Sl_\ell} \equiv \overline{\mathbf{CT}}_{Sl_\ell}$ and $\mathbf{CT}_{Rl_{\ell'}} \equiv \overline{\mathbf{CT}}_{Rl_{\ell'}}$.

Proof: We use induction as follows.

- For each $\ell \in [\tilde{q}_1]$, let $\mathbf{CT}_{Sl_\ell} = (CT_1, \dots, CT_t)$ and $\overline{\mathbf{CT}}_{Sl_\ell} = (\overline{CT}_1, \dots, \overline{CT}_t)$. Obviously, the statement is true for $i = 1$. Assume that it is true for $i - 1$ and we must prove that $(CT_1, \dots, CT_i) \equiv (\overline{CT}_1, \dots, \overline{CT}_i)$. Suppose that $CT_i, CT_{i'}$ are the ciphertexts of m, m' where $i' < i$. For the biggest index $b = \mathbf{ind}(m, m')$, if $b = n + 1$, then CT_i and $CT_{i'}$ are the ciphertexts of the same message. In the simulation, \mathcal{B} finds the tuple $(-, (c'_{1,0}, c'_{1,1}), \dots, (c'_{n,0}, c'_{n,1}))$ from the table T_{CT} and uses it to simulate the ciphertext CT_i by setting $(C_{k,0}, C_{k,1}) = (c'_{k,0}, c'_{k,1})$ for all $k \in [n]$. Then, we have

$$C_{k,0} = c_{k,0} = c'_{k,0} = C'_{k,0} \quad \forall k \in [n].$$

Otherwise, m and m' may have the same prefix of the length $b - 1$. For $k \in [b - 1]$, $(C_{k,0}, C_{k,1})$ has been simulated as the previous case and for $k \in [b + 1, n]$, $(C_{k,0}, C_{k,1})$ has been simulated by using random elements. For the remain part $(C_{b,0}, C_{b,1})$, \mathcal{B} simulates $c_{b,1} = c'_{b,0}$ if $\mathbf{cmp}(m, m') = 1$. Then we have

$$C'_{b,0} = c'_{b,0} = c_{b,1} = C_{b,1}.$$

Since we assumed that $CT_{i'}$ and $\overline{CT}_{i'}$ are identically distributed, by induction, CT_i and \overline{CT}_i are identically distributed.

- For each $\ell \in [\tilde{q}_2]$, let $\mathbf{CT}_{RI_\ell} = (CT_1, \dots, CT_i)$ and $\overline{\mathbf{CT}}_{RI_\ell} = (\overline{CT}_1, \dots, \overline{CT}_i)$. Obviously, the statement is true for $i = 1$. Assume that it is true for $i - 1$ and we must prove that $(CT_1, \dots, CT_i) \equiv (\overline{CT}_1, \dots, \overline{CT}_i)$. Suppose that $CT_i, CT_{i'}$ are the ciphertexts of $(j, m), (j', m')$ where $i' < i$. For the biggest index $b = \mathbf{ind}(m, m')$, if $b = n + 1$, then CT_j and $CT_{j'}$ are the ciphertexts of the same message. In the simulation, \mathcal{B} finds the tuple $(-, (c'_{1,0}, c'_{1,1}), \dots, (c'_{n,0}, c'_{n,1}))$ from the table T_{CT} and uses it to simulate the ciphertext CT_i by computing $(C_{k,0}, C_{k,1}) = (c'_{k,0}^{S_j}, c'_{k,1}^{S_j})$ for all $k \in [n]$. Let $CK_{j,j'} = (K_0, K_1)$ and we have

$$\begin{aligned} e(C_{k,0}, K_0) &= e(c_{k,0}^{S_j}, K_0) = e(c'_{k,0}^{S_j}, K_0) \\ &= e(c'_{k,0}^{S_j}, K_1) = e(C'_{k,0}, K_1) \quad \forall k \in [n]. \end{aligned}$$

Otherwise, m and m' may have the same prefix of the length $b - 1$. For $k \in [b - 1]$, $(C_{k,0}, C_{k,1})$ has been simulated as the previous case and for $k \in [b + 1, n]$, $(C_{k,0}, C_{k,1})$ has been simulated by using random elements. For the remain part $(C_{b,0}, C_{b,1})$, \mathcal{B} simulates $c_{b,1} = c'_{b,0}$ if $\mathbf{cmp}(m, m') = 1$. Then we have

$$\begin{aligned} e(C_{b,1}, K_0) &= e(c_{b,1}^{S_j}, K_0) = e(c'_{b,0}^{S_j}, K_0) \\ &= e(c'_{b,0}^{S_j}, K_1) = e(C'_{b,0}, K_1). \end{aligned}$$

Since we assumed that $CT_{i'}$ and $\overline{CT}_{i'}$ are identically distributed, by induction, CT_i and \overline{CT}_i are identically distributed. ■

REFERENCES

[1] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," in *Proc. IEEE Symp. Found. Comput. Sci.*, Oct. 2013, pp. 40–49.

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st ACM Symp. Theory Comput.*, 2009, pp. 169–178.

[3] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 44–55.

[4] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order-preserving encryption for numeric data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 563–574.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2009, pp. 224–241.

[6] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2011, pp. 578–595.

[7] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, "Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2015, pp. 563–594.

[8] D. Cash, F. H. Liu, A. O'Neill, and C. Zhang, "Reducing the leakage in practical order-revealing encryption," *Cryptol. ePrint Arch.*, Carson City, Nevada, USA, Tech. Rep. 2016/661, 2016. [Online]. Available: <http://eprint.iacr.org/2016/661>

[9] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, "Practical order-revealing encryption with limited leakage," in *Proc. 23rd Int. Conf. Fast Softw. Encryption*. Berlin, Germany: Springer, 2016, pp. 474–493.

[10] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proc. 23rd ACM Conf. Comput. Commun. Secur.*, 2016, pp. 1167–1178.

[11] S. Goldwasser et al., "Multi-input functional encryption," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2014, pp. 578–602.

[12] M. Naor, B. Pinkas, and O. Reingold, "Distributed pseudo-random functions and KDCs," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 1999, pp. 327–346.

[13] F. Kerschbaum and A. Schröpfer, "Optimal average-complexity ideal-security order-preserving encryption," in *Proc. 21st ACM Conf. Comput. Commun. Secur.*, 2014, pp. 275–286.

[14] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 463–477.

[15] D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich, "POPE: Partial order preserving encoding," in *Proc. 23rd ACM Conf. Comput. Commun. Secur.*, 2016, pp. 1131–1142.

[16] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. 22nd ACM Conf. Comput. Commun. Secur.*, 2015, pp. 644–655.

[17] F. B. Durak, T. M. DuBuisson, and D. Cash, "What else is revealed by order-revealing encryption?" in *Proc. 23rd ACM Conf. Comput. Commun. Secur.*, 2016, pp. 1155–1166.

[18] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, "Leakage-abuse attacks against order-revealing encryption," in *Proc. IEEE Symp. Secur. Privacy*, May 2017, pp. 655–672.

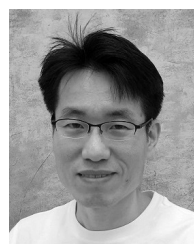
[19] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *J. ACM*, vol. 51, no. 2, pp. 231–262, 2004.



JIEUN EOM received the B.S. degree in mathematics and the M.S. degree in information security from Korea University, Seoul, in 2010 and 2012, respectively, where she is currently pursuing the Ph.D. degree in information security. Her research interests include cryptography and computer security.



DONG HOON LEE (M'06) received the B.S. degree in economics from Korea University in 1984, and the M.S. and Ph.D. degrees in computer science from the University of Oklahoma in 1988 and 1992, respectively. He is currently a Full Professor with Korea University. His research areas include cryptography and information security.



KWANGSU LEE received the B.S. degree in computer science from Yonsei University, South Korea, the M.S. degree in computer science from KAIST, South Korea, in 1998 and 2000, respectively, and the Ph.D. degree in information security from Korea University in 2011. He is currently an Assistant Professor with Sejong University. His research interests include cryptography, provable security, and pairing-based cryptography.

...