Multi-Client Order-Revealing Encryption

Jieun Eom\textsuperscript{*} \quad Dong Hoon Lee\textsuperscript{†} \quad Kwangsu Lee\textsuperscript{‡}

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 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.

Keywords: Symmetric-key encryption, Order-revealing encryption, Multi-client order-revealing encryption, Bilinear maps.

\textsuperscript{*}Korea University, Seoul, Korea. Email: jieunn.eom@gmail.com.
\textsuperscript{†}Korea University, Seoul, Korea. Email: donghlee@korea.ac.kr.
\textsuperscript{‡}Sejong University, Seoul, Korea. Email: kwangsu@sejong.ac.kr.
1 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 issue 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 [8, 9]. 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 [19]. 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 ciphertexts can be compared by simply comparing the order of ciphertexts [1, 4]. 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 [4, 6, 13]. The first ORE scheme of Boneh et al. [4] 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 [5, 6, 13].

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 [10]. 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.
1.1 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.

**Definition.** We first introduce the notion of MC-ORE by extending the concept of ORE 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 compare 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. In this work, we give a simulation-based security model for MC-ORE with a leakage function. Informally, this definition states that if an adversary can obtain information from ciphertexts of clients’ plaintexts \((j_1, m_1), \ldots, (j_q, m_q)\) where \(j_k\) is the index of a client, then it can be inferred from \(L((j_1, m_1), \ldots, (j_q, m_q))\). 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.

**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. 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 \(PRF_i(x) = H(x)^i\) where \(H\) is a hash function and \(s\) is a PRF key. Suppose there is a single client and the client creates ciphertexts \(C = (H(x)^i, H(x + 1)^i)\) and \(C' = (H(x')^i, H(x' + 1)^i)\) 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 \(H(x + 1)^i = H(x')^i\) from two ciphertexts. Now suppose there are two clients with different secret keys \(s\) and \(s'\) and clients create ciphertexts \(C = (H(x)^i, H(x + 1)^i)\) and \(C' = (H(x')^i, H(x' + 1)^i)\) for plaintexts \(x\) and \(x'\) in binary values respectively. To compare two ciphertexts generated by different clients, a user first receives a comparison key \(CK = (g^{s'}, g^{s'})\) from a trusted center and checks whether \(x + 1 = x'\) or not by comparing \(e(H(x + 1)^i, g^{s'}) = e(H(x')^i, g^{s'})\). To extend the comparison of binary values to large values, we modify the encoding method of Chenette et al. that uses the prefixes of a plaintext. Let \(m = x_1x_2\ldots x_n \in \{0, 1\}^n\) be a plaintext. For each \(i \in [n]\), the encryption algorithm encodes two strings \(E_{i, 0} = x_1x_2\ldots x_{i-1}||0x_i\) and \(E_{i, 1} = x_1x_2\ldots x_{i-1}||0x_i + 1\) and evaluates \(C_{i, 0} = H(E_{i, 0})^i\) and \(C_{i, 1} = H(E_{i, 1})^i\). For example, the third bit of \(m = 101\) is encoded as \(E_{3, 0} = 10||01 = 1001\) and \(E_{3, 1} = 10||0 + 1 = 1010\). 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 \(i^*\) 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.

**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 \[5,13\]. 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.

**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.

1.2 Related Work

**Order-Preserving Encryption.** The concept of OPE was introduced by Agrawal et al. \[1\] 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. \[2\], 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 \[12,17,18\].

**Order-Revealing Encryption.** Boneh et al. \[4\] 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. \[6\] 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 et al. [13] constructed an IND-CPA 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-CPA security. Cash et al. [5] 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.

**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. [7] and Grubbs et al. [11] 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.

## 2 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.

### 2.1 Notation

Let \( [n] \) be the set of \( \{1, \ldots, n\} \) and \( [k,n] \) be the set of \( \{k, \ldots, n\} \). Let \( \text{cmp}(m,m') \) be a comparison function that returns 1 if \( m < m' \) and returns 0, otherwise. Let \( \text{ind}(m,m') \) 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 \( \text{prefix}(m,i) \) be a prefix function that takes as input a plaintext \( m = x_1x_2 \cdots x_n \in \{0,1\}^n \) and an index \( i \) and returns \( x_1x_2 \cdots x_{i-1} \) as the prefix of \( x_i \).

### 2.2 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 [4]. 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. [6].

**Definition 2.1 (ORE).** An ORE scheme consists of three algorithms, **Setup**, **Encrypt**, **Compare** which are defined over a well-ordered domain \( D \) as follows:

1. **Setup**(\( 1^\lambda \)). The setup algorithm takes as input a security parameter \( \lambda \) and outputs a secret key \( SK \).

2. **Encrypt**(\( m,SK \)). The encryption algorithm takes as input a plaintext \( m \in D \) and the secret key \( SK \) and outputs a ciphertext \( CT \).

3. **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 \( \text{Compare}(CT,CT') = \text{cmp}(m,m') \).
The best possible security of ORE, which is IND-OCPA, was defined by Boneh et al. [4]. The simulation-based security of ORE with additional leakage $L$ was defined by Chenette et al. [5].

### 2.3 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 2.2 (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 $\lambda$ 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 $m \in D$, 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:

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

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 $L_5$ 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 $L_S((j_1, m_1), \ldots, (j_q, m_q))$. 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 $CK_{j_1} \neq j_2$ was not exposed. After that, if the adversary requests $CK_{j_1, j_2}$ causing the leakage $L_S((j_1, m_1), (j_2, m_2))$, 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 $CK_{j_1, j_2}$ 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 $L_S((j_1, m_1), (j_2, m_2), (j_3, m_3))$. 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 $L_S$ is defined as follows.

**Definition 2.3 (Static SIM-Security with Leakage).** For a security parameter $\lambda$, let $A$ be an adversary and $B$ 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 $L_S(\cdot)$ be a leakage function. The experiments of $REAL^{MC-ORE}_A(\lambda)$ and $SIM^{MC-ORE}_{A,B,L}(\lambda)$ are defined as follows:

\[
\begin{align*}
REAL^{MC-ORE}_A(\lambda) & = 1. (st_B, S, ((j_1, m_1), \ldots, (j_q, m_q))) \leftarrow A(1^\lambda) \\
& \text{2. } (PP, MK) \leftarrow \text{Setup}(1^\lambda, N) \\
& \text{3. } CK_{j, k} \leftarrow \text{GenCmpKey}(j, k, MK, PP), \forall (j, k) \in S \\
& \text{4. for } 1 \leq i \leq q, \\
& \quad CT_{j_i} \leftarrow \text{Encrypt}(m_i, SK_{j_i}, PP) \\
& \text{5. Output (CT}_{j_1}, \ldots, CT_{j_q}) \text{ and st}_A \\
SIM^{MC-ORE}_{A,B,L}(\lambda) & = 1. (st_A, S, ((j_1, m_1), \ldots, (j_q, m_q))) \leftarrow A(1^\lambda) \\
& \text{2. } (st_B, PP) \leftarrow B(1^\lambda, N) \\
& \text{3. } CK_{j, k} \leftarrow B(st_B), \forall (j, k) \in S \\
& \text{4. for } 1 \leq i \leq q, \\
& \quad (st_B, CT_{j_i}) \leftarrow B(st_B, L_S((j_1, m_1), \ldots, (j_i, m_i))) \\
& \text{5. Output (CT}_{j_1}, \ldots, CT_{j_q}) \text{ and st}_A
\end{align*}
\]

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

**Remark 2.4.** For $S = \{(j, k)\}_{j, k \in [N]}$ of index tuples where the comparison key $CK_{j, k}$ is revealed, let $L_S$ be the following leakage function:

\[
L_S((j_1, m_1), \ldots, (j_q, m_q)) = \{\text{cmp}(m_i, m_{i'}) : 1 \leq i < i' \leq q, j_{i'} = j_i \text{ or } (j_{i'}, j_i) \in S\}.
\]

If an MC-ORE scheme is secure with leakage $L_S$, then it is IND-CPA secure.
3 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.

3.1 Asymmetric Bilinear Groups

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

1. Bilinearity: $\forall u \in G, \forall \hat{v} \in \hat{G}$ and $\forall a, b \in \mathbb{Z}_p$, $e(u^a, \hat{v}^b) = e(u, \hat{v})^{ab}$.

2. Non-degeneracy: $\exists g \in G, \hat{g} \in \hat{G}$ such that $e(g, \hat{g})$ has order $p$ in $T$.

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

**Assumption 3.1** (External Diffie-Hellman, XDH). Let $(p, G, \hat{G}, T, e)$ be a tuple randomly generated by $G_{as}(\lambda)$ where $p$ is a prime order of the groups. Let $g, \hat{g}$ be random generators of groups $G, \hat{G}$, respectively. The XDH assumption is that the decisional Diffie-Hellman (DDH) assumption holds in $G$. That is, if the challenge tuple

$$D = \left((p, G, \hat{G}, T, e), g, \hat{g}, g^a, \hat{g}^b\right)$$

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

3.2 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 $L_S$ is defined as follows:

$$L_S((j_1, m_1), \ldots, (j_q, m_q)) = \{\text{cmp}(m_i, m_i), \text{ind}(m_i, m_i) : 1 \leq i < q, j_i = j, \text{or } (j_i, j_i) \in S\}.$$  

If $S = \emptyset$, then $L_S$ becomes equal to the leakage function $L$ defined by Chenette et al. [6]. 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$(\lambda, N)$. This algorithm first generates bilinear groups $G, \hat{G}, T$ of prime order $p$ with group generators $g \in G$ and $\hat{g} \in \hat{G}$. 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 $PP = ((p, G, \hat{G}, T, e), g, \hat{g}, H)$ where $H : \{0, 1\}^* \to G$ is a full-domain hash function.

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


MC-ORE.Encrypt($m, SK_j, PP$). Let $m = x_1x_2 \cdots x_n \in \{0, 1\}^n$ and $SK_j = s_j$. For each $i \in [n]$, it computes $C_{i,0} = H(\text{prefix}(m,i)[0:x_i])^s_j$ and $C_{i,1} = H(\text{prefix}(m,i)[(0x_i + 1)])^s_j$ where $\|\|$ is the concatenation of two bit strings. It outputs a ciphertext $CT_j = \{(C_{0,0}, C_{1,1})\|i\| [n]\}$.

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

3.3 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:

$$E_0(i,m) = \text{prefix}(m,i)[0:x_i], \quad E_1(i,m) = \text{prefix}(m,i)[(0x_i + 1)].$$

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 $CT_j = \{(C_{i,0}, C_{i,1})\|i\| [n]\}$ and $CT'_k = \{(C'_{i,0}, C'_{i,1})\|i\| [n]\}$ be ciphertexts on messages $m = x_1x_2 \cdots x_n \in \{0, 1\}^n$ and $m' = x'_1x'_2 \cdots x'_n \in \{0, 1\}^n$. 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

$$C_{i,0} = H(E_0(i,m))^s_j = H(E_0(i,m'))^s_j = C'_{i,0} \forall i < i^*$$
$$C_{r,1} = H(E_1(i^*, m'))^s_j = H(E_0(i^*, m'))^s_j = C'_{r,1}.$$
3.4 Security Analysis

We prove the security of the basic MC-ORE scheme with the leakage function $L_S$ in the ST-SIM security model. We define a sequence of experiments from $H_0$ corresponding to the real experiment to $H_3$ 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 $H_3$, the ciphertexts are simulated with respect to the leakage function $L_S$, and consequently $H_3$ corresponds to the ideal experiment. The details are given as follows.

**Theorem 3.2.** The basic MC-ORE scheme is ST-SIM secure with the leakage function $L_S$ 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 $L_S$ in which the adversary has no advantage. The hybrid experiments $H_0, H_1, H_2, \text{ and } H_3$ are defined as follows:

- $H_0$: This experiment corresponds to the real world experiment.
- $H_1$: This experiment is similar to $H_0$ except that the ciphertext $CT_j$ such that $(j, j') \notin S$ for any client index $j'$ is generated by using random elements.
- $H_2$: This experiment is similar to $H_1$ except that the ciphertext $CT_j$ such that $(j, j') \in S$ for some client index $j'$ is generated by using random elements.
- $H_3$: In this experiment, the ciphertexts are generated with the leakage function $L_S$ and the rest are same to $H_2$. This experiment corresponds to the ideal world experiment.

From the following Lemmas 3.4, 3.5, and 3.6 that claim the indistinguishability of the above experiments, we have that $H_0$ and $H_3$ are computationally indistinguishable.

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

**Assumption 3.3 (Multi-External Diffie-Hellman, mXDH).** Let $(p, G, \hat{G}, G_T, e)$ be a tuple randomly generated by $G_{as}(1^k)$ where $p$ is a prime order of the groups. Let $g, \hat{g}$ be random generators of groups $G, \hat{G}$, respectively. The mXDH assumption is that if the challenge tuple

$$D = ((p, G, \hat{G}, G_T, e), g, \hat{g}, g^a, \{g^{b_{i,1}}, \cdots, g^{b_{i,n}}\}_{i \in [t]})$$

and $T$ are given, no PPT algorithm $A$ can distinguish $T_0 = \{\{g^{a_{b_{i,1}}}, \cdots, g^{a_{b_{i,n}}}\}_{i \in [t]}\}$ from $T_1 = \{\{g^{\hat{c}_{i,1}}, \cdots, g^{\hat{c}_{i,n}}\}_{i \in [t]}\}$ with more than a negligible advantage. The advantage of $A$ is defined as $\text{Adv}_A^{\text{mXDH}}(\lambda) = |\Pr[A(D, T_0) = 0] - \Pr[A(D, T_1) = 0]|$ where the probability is taken over random choices of $a, (b_{i,1}, \cdots, b_{i,n}), (c_{i,1}, \cdots, c_{i,n}) \in \mathbb{Z}_p$ 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 [15].
Lemma 3.4. The hybrid experiments $H_0$ and $H_1$ are computationally indistinguishable to the polynomial-time adversary assuming that the mDXH assumption holds.

Proof. To prove this lemma, we additionally define a sequence of hybrid experiments $H_0 = H_{0,0}, H_{0,1}, \ldots,$ $H_{0,q} = H_1$ for some $q$ as follows.

$H_{0,\mu}$: Let $I = (j_1, \cdots, j_\mu)$ be a tuple of challenge client index. For all $j_i \in I$ such that $(j_i, *) \notin S$, let $j^*_1, \cdots, j^*_\mu \in I$ be distinct client indices where $\tilde{q} \leq \mu$. Let $SI_\mu = \{i \in [q] : j_i = j^*_\mu\}$ be an index set of some client indices where $\mu \in [\tilde{q}]$. In this experiment, we change the generation of the $\mu$-th ciphertext set with the index set $SI_\mu$. If $\ell \leq \mu$, the ciphertexts in the $\ell$-th ciphertext set with $SI_\mu$ are changed to be random elements. Otherwise, the ciphertexts in the $\ell$-th ciphertext set with $SI_\mu$ 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 $H_{0,\mu-1}$ and $H_{0,\mu}$ 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 $A$ that distinguishes $H_{0,\mu-1}$ from $H_{0,\mu}$ with non-negligible advantage. A simulator $B$ that solves the mDXH assumption using $A$ is given: a challenge tuple $D = ((p, G, \hat{G}, G_T, e), g, \hat{g}, g^\beta, \{g^b, \cdots, g^{b_{|\tilde{q}|}}\}_{i \in [\tilde{q}]})$ and $T = (\{X_1, \cdots, X_{2\tilde{q}}\}_{i \in [\tilde{q}]})$. $B$ interacts with $A$ as follows.

Let $(stA, S, ((j_1, m_1), \cdots, (j_q, m_q)))$ be the output of $A$ and $SI_{\mu}$ be the target index set of $j_\mu^*$. The simulator $B$ 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 $j_\mu^*$, it implicitly sets $SK_{j_\mu^*} = \alpha$. Now, $B$ 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, $B$ maintains a random oracle table $T_H$ for the consistency of a simulation. Initially, $B$ fixes some hash queries for the simulation of the ciphertext with the challenge tuple $(j_i, m_i)$ such that $i \in SI_\mu$, which is output of $A$. For the first message $m_1$, $B$ sets $h_{k,0} = g^{b_{1,\ell+1}}, h_{k,1} = g^{b_{2,\ell}}$ 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 $m_i$, $B$ first finds the biggest index $d = \text{ind}(m_i, m_{i'})$ 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 $\text{cmp}(m_i, m_{i'}) = 1$, then $B$ sets $h_{d,0} = g^{b_{d,\ell+1}}, h_{d,1} = h'_{d,0}$ and otherwise, it sets $h_{d,0} = h'_{d,1}, h_{d,1} = g^{b_{d,\ell}}$. Next, it sets $h_{k,0} = g^{b_{1,\ell+1}}, h_{k,1} = g^{b_{2,\ell}}$ for all $k \in [d + 1, n]$. It adds the tuples $(E_0(k, m_{i}), h_{k,0}), (E_1(k, m_{i}), h_{k,1})$ 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 $\beta \in \{0, 1\}$, $B$ 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 $h \in G$ and adds the tuple $(E_{\beta}(k, m), h)$ to $T_H$. Finally, it gives $h$ to $A$ as a response.

To handle the creation of ciphertexts, $B$ carefully uses the hash table and the challenge elements in the assumption. Let $((j_1, m_1), \cdots, (j_q, m_q))$ be the challenge tuples. If $(j_i, *) \in S$, then $B$ 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 $i \in SI_\ell$ for some $\ell \in [\tilde{q}]$. $B$ creates a set of ciphertexts with the index set $SI_\ell$ for each $\ell \in [\tilde{q}]$ as follows:

- **Case $\ell < \mu$**: $B$ creates the ciphertext for the index $i \in SI_\ell$ sequentially. For the smallest index $i \in SI_\ell$, it chooses random elements $R_{k,0}, R_{k,1} \in G$ for all $k \in [n]$ and creates $CT_i = \{(R_{k,0}, R_{k,1})\}_{k \in [n]}$. For the next index $i$, it first finds the biggest index $d = \text{ind}(m_i, m_{i'})$ for any $i' < i$. It sets $C_{k,0} = C_{k,0}^{j_i}, C_{k,1} = C_{k,1}^{j_i}$ for all $k \in [d - 1]$ where $CT_{j_i} = \{(C_{d,0}^{j_i}, C_{d,1}^{j_i})\}_{k \in [n]}$. If $\text{cmp}(m_i, m_{i'}) = 1$, then it chooses a random element $R_{d,0} \in G$ and sets $C_{d,0} = R_{d,0}, C_{d,1} = C_{d,0}$. Otherwise, it chooses a random element $R_{d,1} \in G$ and sets $C_{d,0} = C_{d,1}, C_{d,1} = R_{d,1}$. Next, it chooses random elements $R_{k,0}, R_{k,1} \in G$ 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 $CT_{j_i} = \{(C_{k,0}^{j_i}, C_{k,1}^{j_i})\}_{k \in [n]}$. At last, it creates the $\ell$-th ciphertext set $CT_{SI_\ell} = \{(CT_{j_i})_{j \in SI_\ell}\}$.
• **Case \( \ell = \mu \):** \( B \) creates the ciphertext for the index \( i \in SL_\mu \) sequentially. For the smallest index \( i \in SL_\ell \), \( B \) sets \( C_{k,0} = X_{1,2k-1}, C_{k,1} = X_{1,2k} \) for all \( k \in [n] \) and creates the ciphertext \( CT_{j,\ell} = \{(C_{k,0}, C_{k,1})\}_{k \in [n]} \).

For the next index \( i \), \( B \) first finds the biggest index \( d = \text{ind}(m_I, m_F) \) for any \( \ell' < i \). It sets \( C_{k,0} = C_{k,0}^\prime, C_{k,1} = C_{k,1}^\prime \) for all \( k \in [d - 1] \) where \( CT_{j,\ell'} = \{(C_{k,0}^\prime, C_{k,1}^\prime)\}_{k \in [n]} \). If \( \text{cmp}(m_I, m_F) = 1 \), then it sets \( C_{d,0} = X_{1,2d-1}, C_{d,1} = C_{d,0}^\prime \) and otherwise, it sets \( C_{d,0} = C_{d,0}^\prime, C_{d,1} = X_{1,2d}^\prime \). Next, it sets \( C_{k,0} = X_{1,2k-1}, C_{k,1} = X_{1,2k}^\prime \) for all \( k \in [d + 1, n] \). It creates the ciphertext \( CT_{j,\ell} = \{(C_{k,0}, C_{k,1})\}_{k \in [n]} \). At last, it creates the \( \mu \)-th ciphertext set \( CT_{SI_\mu} = \{(CT_{j,\ell})\}_{\ell \in [\mu]} \). Note that it does not know the secret key \( a \).

• **Case \( \ell > \mu \):** It creates the ciphertext set \( CT_{SI_\mu} \) by running the MC-ORE.Encrypt algorithm with hash queries.

If \( T = (\{g^{a_1 h_1}, \ldots, g^{a_2 h_2}\}, \ldots, \{g^{a_{m-1} h_{m-1}}, \ldots, g^{a_m h_m}\}) \), then \( CT_{SI_\mu} \) are ciphertexts in \( H_0, \mu \). Otherwise, \( CT_{SI_\mu} \) are ciphertexts in \( H_{0, \mu} \). By the mXDH assumption, two experiments \( H_{0, \mu - 1} \) and \( H_{0, \mu} \) are computationally indistinguishable.

**Lemma 3.5.** The hybrid experiments \( H_1 \) and \( H_2 \) are computationally indistinguishable to the polynomial-time adversary assuming that the mXDH assumption holds.

**Proof.** We additionally define a sequence of hybrid experiments \( H_1 = H_{1,0}, H_{1,1}, \ldots, H_{1,q} = H_2 \) for some \( q \) as follows.

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

Suppose there exists an adversary \( A \) that distinguishes \( H_{1, \mu - 1} \) from \( H_{1, \mu} \) with non-negligible advantage. A simulator \( B \) that solves the mXDH assumption using \( A \) is given: a challenge tuple \( D = ((p, G, \hat{G}, G_T, e), g, g^a, g^b, (g^{a_1 h_1}, \ldots, g^{a_m h_m})_{i \in [I]} \) and \( T = (\{X_{1,1}, \ldots, X_{1,n}\}_{i \in [I]}) \). \( B \) interacts with \( A \) as follows.

Let \( (s_A, S, (j_1, m_1), \ldots, (j_q, m_q)) \) be the output of \( A \) and \( RI_\mu \) be the target index set. The simulator \( B \) first sets the secret keys of clients as follows. For each \( j = j_i \), if \( i \notin RI_\mu \), 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, \( B \) can generate a comparison key \( CK_{j,k} = (g^{s_j}, g^{s_k}) \) 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, \( B \) maintains a random oracle table \( T_H \) for the consistency of a simulation. This simulation is same to the proof of the Theorem 3.4 except that \( B \) fixes some hash queries for the simulation of the ciphertext with the challenge tuple \( (j_i, m_i) \) such that \( i \in RI_\mu \).

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

• **Case \( \ell < \mu \):** \( B \) creates the ciphertext for the index \( i \in RI_\ell \) sequentially. For the smallest index \( i \in RI_\ell \), it chooses random elements \( R_{k,0}, R_{k,1} \in \hat{G} \) and computes \( C_{k,0} = R_{k,0}^i, C_{k,1} = R_{k,1}^i \) for all
k \in [n]$. It creates $CT_j = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. For the next index $i$, $B$ first finds the biggest index $d = \text{ind}(m_i, m_p)$ for any $i' < i$ and computes $s = s_{j_i}/s_{j_{i'}}$. It computes $C_{k,0} = C_{s_i}^{d_{k,0}}, C_{k,1} = C_{s_i}^{d_{k,1}}$ for all $k \in [d - 1]$ where $CT_{j_p} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. If $\text{cmp}(m_i, m_p) = 1$, then it chooses a random element $R_{d,0} \in \mathbb{G}$ and computes $C_{d,0} = R_{d,0}^{s_{j_i}}, C_{d,1} = C_{d,0}^{s_{j_{i'}}}$. Otherwise, it chooses a random element $R_{d,1} \in \mathbb{G}$ and computes $C_{d,0} = C_{s_i}^{d_{k,0}}, C_{d,1} = R_{d,1}^{s_{j_{i'}}}$. 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 $CT_j = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. At last, it creates the $\ell$-th ciphertext set $CT_{R_{\ell}} = (CT_j)_{j \in R_{\ell}}$.

- **Case $\ell = \mu$:** $B$ creates the ciphertext for the index $i \in R_{\mu}$ sequentially. For the smallest index $i \in R_{\mu}$, $B$ 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 $CT_j = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. For the next index $i$, $B$ first finds the biggest index $d = \text{ind}(m_i, m_p)$ for any $i' < i$ and it computes $s = s_{j_i}/s_{j_{i'}}$. It computes $C_{k,0} = C_{s_i}^{d_{k,0}}, C_{k,1} = C_{s_i}^{d_{k,1}}$ for all $k \in [d - 1]$ where $CT_{j_i} = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. If $\text{cmp}(m_i, m_p) = 1$, then $B$ computes $C_{d,0} = X_{1,2d-1}^{s_{j_i}}, C_{d,1} = C_{s_i}^{d_{k,0}}$ and otherwise, it computes $C_{d,0} = C_{s_i}^{d_{k,0}}, C_{d,1} = X_{1,2d}^{s_{j_{i'}}}$. Next, it computes $C_{k,0} = X_{1,2k-1}^{s_{j_i}}, C_{k,1} = X_{1,2k}^{s_{j_{i'}}}$ for all $k \in [d + 1, n]$. Then, it creates the ciphertext $CT_j = (\{C_{k,0}, C_{k,1}\}_{k \in [n]})$. At last, it creates the $\mu$-th ciphertext set $CT_{R_{\mu}} = (CT_j)_{j \in R_{\mu}}$. Note that it does not know the secret key $a$.

- **Case $\ell > \mu$:** It creates the ciphertext set $CT_{R_{\ell}}$ by running the MC-ORE.Encrypt algorithm with hash queries.

If $T = \{(g_{ab,1}^{s_{j_i}}), \ldots, g_{ab,2n}^{s_{j_i}}\}$, then $CT_{R_{\ell}}$ are ciphertexts in $H_1, \mu-1$. Otherwise, $CT_{R_{\mu}}$ are ciphertexts in $H_1, \mu$. By the mXDH assumption, two experiments $H_1, \mu-1$ and $H_1, \mu$ are computationally indistinguishable.

**Lemma 3.6.** The hybrid experiments $H_2$ and $H_3$ 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 $A$ that distinguishes $H_2$ from $H_3$ with non-negligible advantage. We construct an efficient simulator $B$ for which the two distributions $H_2$ and $H_3$ are statistically indistinguishable.

Let $(st_A, S, ((j_1, m_1), \ldots, (j_q, m_q)))$ be the output of $A$. $B$ first outputs random public parameters $PP$ with the initial state $st_A$. 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, $B$ 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 $\beta \in \{0, 1\}$, $B$ 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 $h \in A$ as a response.

To handle the creation of ciphertexts, $B$ also maintains a ciphertext table $T_{CT}$ for consistency of the simulation. Let $I = (j_1, \ldots, j_q)$ be a tuple of challenge client index. For all $j_i \in I$ such that $(j_i, \ast) \notin S$, let $j_{i_1}^*, \ldots, j_{i_{j_i}}^* \in I$ be distinct client indices and $SI_{\mu} = \{i \in [q] : j_i = j_{i_1}^*\}$ be an index set of same client indices where $\mu \in [\hat{q}]$. For all $j_i \in I$ such that $(j_i, \ast) \in S$, let $RI_{\mu} = \{i \in [q] : j_i \ast s$ are co-related indices $\}$ be an index set of co-related client indices where $\mu \in [\hat{q}]$. $B$ simulates the creation of a set of ciphertexts with a client index set $SI_{\mu}$ or $RI_{\mu}$ by using $st_B$ and $S(L((j_1, m_1), \ldots, (j_q, m_q)))$ as follows:

- For the creation of the ciphertexts with each set $SI_{\mu}$, $B$ initiates the ciphertext table $T_{CT}$. For the smallest index $i \in SI_{\mu}$, $B$ 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 \((i, (c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1}))\) to \( T_{CT} \) and creates \( CT_i = \{ (c_{k,0}, C_k) \}_{k \in [n]} \).

For the next index \( i \in S_{\ell} \), it creates the ciphertext sequentially as follows. It first finds the biggest index \( b = \text{ind}(m_i, m_{\ell}) \) for any \( i' < i \) and then finds a tuple \((i', (c'_{1,0}, c'_{1,1}), \ldots, (c'_{n,0}, c'_{n,1}))\) 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. 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 \( \text{cmp}(m_i, m_{\ell}) = 1 \), it sets \( c_{b,0} = c'_{b,0} \) and chooses a random element \( c_{b,0} \in \mathbb{G} \). Otherwise, it sets \( c_{b,0} = c'_{b,0} \) and chooses a random element \( c_{b,1} \in \mathbb{G} \).

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

For the creation of the ciphertext with each set \( RI_{\mu} \), \( \mathcal{B} \) initiates the ciphertext table \( T_{CT} \). For the smallest index \( i \in RI_{\mu} \), \( \mathcal{B} \) chooses random elements \((c_{k,0}, c_{k,1}) \in \mathbb{G} \times \mathbb{G}\) and computes \((c_{k,0}, C_k) = (c_{k,0}, c_{k,1})\) for all \( k \in [n] \). It adds the tuple \((i, (c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1}))\) to \( T_{CT} \) and creates \( CT_i = \{ (c_{k,0}, C_k) \}_{k \in [n]} \). For the next index \( i \in RI_{\mu} \), it creates the ciphertext sequentially as follows. It first finds the biggest index \( b = \text{ind}(m_i, m_{\ell}) \) for any \( i' < i \) and then finds a tuple \((i', (c'_{1,0}, c'_{1,1}), \ldots, (c'_{n,0}, c'_{n,1}))\) 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 ciphertext \( CT_{RI_{\mu}} \). Then, \( \mathcal{B} \) computes \((c_{k,0}, C_k) = (c_{k,0}, c_{k,1})\) for all \( k \in [n] \). It creates \( CT_i = \{ (c_{k,0}, C_k) \}_{k \in [n]} \) and adds the tuple \((i, (c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1}))\) to \( T_{CT} \). At last, it creates the ciphertext set \( CT_{RI_{\mu}} = \{ (CT_i) \}_{i \in RI_{\mu}} \).

**Correctness of the Simulation.** To show the correctness of the simulation, we prove that the distributions

\( ((CT_{SI_{\ell}}, \ldots, CT_{SI_{\mu}}), (CT_{RI_{\ell}}, \ldots, CT_{RI_{\mu}})) \) and

\( ((CT_{SI_{\ell}}, \ldots, CT_{SI_{\mu}}), (CT_{RI_{\ell}}, \ldots, CT_{RI_{\mu}})) \)

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.

- \( \forall \ell \in [\hat{q}_1], \forall \ell' \in [\hat{q}_2], \) \( CT_{SI_{\ell}} \) and \( CT_{RI_{\ell'}} \) are distributed independently.

- \( \forall \ell \in [\hat{q}_1], \forall \ell' \in [\hat{q}_2], \) \( CT_{SI_{\ell}} \equiv CT_{SI_{\ell'}} \) and \( CT_{RI_{\ell}} \equiv CT_{RI_{\ell'}} \).

The first condition is simply proved since each ciphertext for \( SI_{\ell} \) and \( RI_{\ell'} \) are simulated independently. Next, we use induction to prove that the second condition holds as follows.

- For each \( \ell \in [\hat{q}_1] \), let \( CT_{SI_{\ell}} = (CT_1, \ldots, CT_I) \) and \( \overline{CT_{SI_{\ell}}} = (\overline{CT_1}, \ldots, \overline{CT_I}) \). Obviously, the statement is true for \( i = 1 \). Assume that it is true for \( i - 1 \) and we must prove that \( (CT_1, \ldots, CT_I) \equiv (\overline{CT_1}, \ldots, \overline{CT_I}) \).

Suppose that \( CT_i, CT_{\ell} \) are the ciphertexts of \( m, m' \) where \( i' < i \). For the biggest index \( b = \text{ind}(m, m') \), if \( b = n + 1 \), then \( CT_i \) and \( CT_{\ell} \) are the ciphertexts of the same message. In the simulation, \( \mathcal{B} \) finds the tuple \((- (c'_{1,0}, c'_{1,1}), \ldots, (c'_{n,0}, c'_{n,1}))\) from the table \( T_{CT} \) and uses it to simulate the ciphertext \( CT_i \) by setting \((c_{k,0}, C_k) = (c'_{k,0}, c'_{k,1})\) for all \( k \in [n] \). Then, we have

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

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 remainder part \((C_{b,0}, C_{b,1})\), \( \mathcal{B} \) simulates \( c_{b,0} = c_{b,0} \) if \( \text{cmp}(m, m') = 1 \). Then we have

\[
C_{b,0} = c_{b,0} = c_{b,1} = C_{b,1}.
\]
Since we assumed that $CT_\ell$ and $\overline{CT}_\ell$ are identically distributed, by induction, $CT_i$ and $\overline{CT}_i$ are identically distributed.

- For each $\ell \in [\hat{q}_2]$, let $CT_{RL_\ell} = (CT_1, \ldots, CT_i)$ and $\overline{CT}_{RL_\ell} = (\overline{CT}_1, \ldots, \overline{CT}_i)$. Obviously, the statement is true for $i = 1$. Assume that it is true for $i - 1$ and we must prove that $(CT_1, \ldots, CT_i) \equiv (\overline{CT}_1, \ldots, \overline{CT}_i)$.

Suppose that $CT_i, \overline{CT}_i$ are the ciphertexts of $(j, m), (j', m')$ where $j' < i$. For the biggest index $b = \text{ind}(m, m')$, if $b = n + 1$, then $CT_j$ and $CT_j'$ are the ciphertexts of the same message. In the simulation, $B$ finds the tuple $(-, (c_{1,0}', c_{1,1}'), \ldots, (c_{b,0}', c_{b,1}'))$ from the table $T_{CT}$ and uses it to simulate the ciphertext $CT_j$ by computing $(c_{k,0}, C_{k,1}) = (\hat{c}_{k,0}'^{s_j'}, \hat{c}_{k,1}'^{s_j'})$ for all $k \in [n]$. Let $CK_{j,j'} = (K_0, K_1)$ and we have

$$e(C_{k,0}, K_0) = e(c_{k,0}', K_0) = e(c_{k,0}'^{s_j'}, K_0) = e(c_{k,0}'^{s_j'}, K_1) = e(C_{k,0}'^{s_j'}, K_1) \forall k \in [n].$$

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 $k \in [b, n]$, $(c_{k,0}, C_{k,1})$ has been simulated by using random elements. For the remain part $(c_{b,0}, C_{b,1})$, $B$ simulates $c_{b,1} = c_{b,0}'$ if $\text{cmp}(m, m') = 1$. Then we have

$$e(C_{b,1}, K_0) = e(c_{b,1}', K_0) = e(c_{b,0}'^{s_j'}, K_0) = e(c_{b,0}'^{s_j'}, K_1) = e(C_{b,0}'^{s_j'}, K_1).$$

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

In addition, suppose that the tuple $(E_0(k, m), h)$ is in $T_H$ and $(i, (c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1}))$ is in $T_{CT}$ for some $i$ such that $m_i = m$. By the Lemmas 3.4 and 3.5, A cannot find out that $h$ and $c_{k,0}$ are different. This completes the correctness of simulation. \hfill $\square$

### 3.5 Extensions

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

**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 a 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 itself 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 $CK = ((\hat{g}^{s_j})', (\hat{g}^{s_k})')$ 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.

**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 $\hat{g}^{r s_j}$ and $\hat{g}^{r s_k}$. 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 $g_r$ for some random exponent $r$. 

15
4 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.

4.1 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 \cite{5,13}, 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.

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 2.2 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 key.

4.1.1 Construction

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 $\mathcal{L}^{EORE}_S$ is defined as follows:

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

Our EORE scheme with leakage $\mathcal{L}^{EORE}_S$ 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 random exponents $s_j, a_j \in \mathbb{Z}_p$ and computes $h_j = g^{s_j}$ 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 $PP = (\langle p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e \rangle,\hat{g},\{h_j\}_{j\in[N]},H)$ where $H : \{0,1\}^* \rightarrow \mathbb{G}$ is a full-domain hash function.

**EORE.GenKey**$(j, MK, PP)$. Let $MK = \{(s_1,\ldots,s_N),\{h_1,\ldots,h_N\}\}$. It outputs a secret key $SK_j = s_j$.

**EORE.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 $F_{i,0} = H(\text{prefix}(m,i)||0x_i)^{s_i}$ and $F_{i,1} = H(\text{prefix}(m,i)||1x_i + 1)^{s_i}$. 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 $K_{0,0} = \hat{g}^{rs_j}, K_{0,1} = \hat{h}_j^{rs_j}$ and $K_{1,0} = \hat{g}^{rs_k}, K_{1,1} = \hat{h}_j^{rs_k}$. It outputs the comparison key $CK_{j,k} = \{(K_{b,0},K_{b,1})_{b\in\{0,1\}}\}$. 

16
Remark 4.1. The leakage function $L^EORE$ is same to the leakage function $L^EORE$ of the basic MC-ORE scheme except that it excludes the condition $j_r = j_i$. 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.

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 $L_j^{ORE}$ be the leakage function of the underlying ORE scheme corresponding to the client index $j$ and $L_S^{EORE}$ be the leakage function of our EORE scheme. A leakage function $L_S^{MC-ORE}$ is defined as follows:

$$L_S^{MC-ORE}(\{(j_1, m_1), \ldots, (j_q, m_q)\}) = \{L_j^{ORE}(m_{i_1}, \ldots, m_{i_p}) \cup L_S^{EORE} : j = j_{i_1} = \ldots = j_{i_p}\},$$

where the sequence sets $M_j = \{m_{i_1}, \ldots, m_{i_p}\}$ satisfy $\bigcap M_j = \emptyset$ and $\bigcup M_j = \{m_1, \ldots, m_q\}$. Here, if $S = \emptyset$, meaning that any comparison key is not revealed, then $L_S^{MC-ORE}$ becomes equal to the reduced leakage functions $\{L_j^{ORE}\}$ 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 ORE scheme for the same client. That is, $L_S^{MC-ORE}$ will be at most $L_S^{EORE}$. Our MC-ORE scheme with leakage $L_S^{MC-ORE}$ that combines an ORE scheme and our EORE scheme is described as follows:

\begin{itemize}
  \item **MC-ORE.Setup**$(1^\lambda, N)$. It obtains $MK_{EORE}$ and $PP_{EORE}$ by running **ORE.Setup**$(1^\lambda, N)$ and outputs $MK = MK_{EORE}$ and $PP = PP_{EORE}$.
  
  \item **MC-ORE.GenKey**(j, MK, PP). It runs **ORE.Setup**$(1^\lambda)$ and **ORE.GenKey**(j, MK, PP) to obtain $SK_{ORE,j}$ and $SK_{EORE,j}$, respectively. It outputs a secret key $SK_j = (SK_{ORE,j}, SK_{EORE,j})$.
  
  \item **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 **ORE.Encrypt**(m, SK_{EORE,j}, PP) respectively. It outputs a ciphertext $CT_j = (OC_j, EC_j)$.
  
  \item **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).
  
  \item **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 **ORE.GenCmpKey**(j, k, MK, PP).
  
  \item **MC-ORE.CompareMC**(CT_j, CT'_j, CK_{j,k}, PP). Let $CT_j = (OC_j, EC_j)$ and $CT'_k = (OC'_k, EC'_k)$ for the same client index $j$. It returns the result of **ORE.CompareMC**(EC_j, EC'_k, CK_{j,k}, PP).
\end{itemize}
4.2 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 $CK_{j,k} = (K_{0,0}, K_{0,1}, K_{1,0}, K_{1,1}) = (g^{rs_k}, \hat{h}_k^{rs_j}, g^{rs_j}, \hat{h}_j^{rs_k})$ 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

$$e(C_{i,0,0,1}, K_{1,0}) / e(C_{i,0,1,1}, K_{1,1}) = e(H(E_0(i, m)), g^{\hat{h}_j^{rs_k}}) / e(g^{\hat{h}_k^{rs_j}}, g^{\hat{h}_j^{rs_k}}) = e(H(E_0(i, m)), g^{\hat{h}_j^{rs_k}}) / e(g^{\hat{h}_k^{rs_j}}, g^{\hat{h}_j^{rs_k}})$$

$$= e(H(E_0(i, m'), g^{\hat{h}_j^{rs_k}}) / e(g^{\hat{h}_k^{rs_j}}, g^{\hat{h}_j^{rs_k}}) = e(C'_{i,0,0,1}, K_{0,0}) / e(C'_{i,0,1,1}, K_{0,1}) \quad \forall i < i^*,$n

$$e(C_{i,0,0,1}, K_{1,0}) / e(C_{i,0,1,1}, K_{1,1}) = e(H(E_1(i, m)), g^{\hat{h}_j^{rs_k}}) / e(g^{\hat{h}_k^{rs_j}}, g^{\hat{h}_j^{rs_k}}) = e(H(E_1(i, m), g^{\hat{h}_j^{rs_k}}) / e(g^{\hat{h}_k^{rs_j}}, g^{\hat{h}_j^{rs_k}}) = e(C'_{i,0,0,1}, K_{0,0}) / e(C'_{i,0,1,1}, K_{0,1}).$$

4.3 Security Analysis

We now prove the security of the enhanced MC-ORE scheme with the leakage function $L_S^{MC-ORE}$ 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 $H_0$ corresponding to the real experiment to $H_4$ 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 $L_j^{ORE}$. 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 respect to the leakage function $L_S^{EORE}$, and consequently $H_4$ corresponds to the ideal experiment. The details are given as follows.

**Theorem 4.2.** The enhanced MC-ORE scheme is ST-SIM secure with the leakage function $L_S^{MC-ORE}$ in the random oracle model if the ORE scheme is SIM secure with the leakage function $L_j^{ORE}$, the basic MC-ORE scheme is ST-SIM secure with the leakage function $L_S$, 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 $L_{MC-ORE}^S$ in which the adversary has no advantage. The hybrid experiments $H_0, H_1, H_2, H_3,$ and $H_4$ are defined as follows:

$H_0$ : This experiment corresponds to the real world experiment.

$H_1$ : In this experiment, the ORE ciphertexts $OC_j$ are generated with the leakage function $L_j^{ORE}$ and the rest are same to $H_0$. We have that $H_0$ and $H_1$ are indistinguishable if the underlying ORE scheme is secure with respect to the leakage function $L_j^{ORE}$.

$H_2$ : This experiment is similar to $H_1$ except that the EORE ciphertext $EC_j$ such that $(j, j') \notin S$ for any client index $j'$ is generated by using random elements.

$H_3$ : This experiment is similar to $H_2$ except that the EORE ciphertext $EC_j$ such that $(j, j') \in S$ for some client indices $j'$ is generated by using random elements.
**H₄**: In this experiment, the EORE ciphertext ECⱼ, such that (ⱼi, j) ∈ S for some client indices j is generated with the leakage function Lₑ⁰⁰ᵣₑ and the rest are same to H₃. This experiment corresponds to the ideal world experiment.

From the following Lemmas 4.3, 4.4, 4.5, and 4.6 that claim the indistinguishability of the experiments, we have that H₀ and H₄ are computationally indistinguishable.

**Lemma 4.3.** The hybrid experiments H₀ and H₄ are computationally indistinguishable to the polynomial-time adversary if the underlying ORE scheme is SIM secure with the leakage function Lₑ⁰⁰ᵣₑ.

**Proof.** The proof of this lemma is simple since a ciphertext CT is a ciphertext in H₀. A simulator can use the simulator of the ORE scheme for the generation of OCj and it can generate other elements in ECj by the randomly chosen master key of an EORE scheme.

**Lemma 4.4.** The hybrid experiments H₁ and H₂ 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 H₁ = H₁,0, H₁,1, ⋱, H₁,q = H₂ as follows.

H₁,µ : In this experiment, we change the generation of the µ-th ciphertext if (jµ, *) ∉ S. If i ≤ µ and (jµ, *) ∉ S, the i-th EORE ciphertext ECⱼi, is generated by using random elements. Otherwise, the i-th EORE ciphertext ECⱼi is generated by running the normal encryption algorithm. Note that H₁,µ−1 and H₁,µ are trivially equal if (jµ, *) ∈ S.

Without loss of generality, we assume that (jµ, *) ∉ S. Suppose there exists an adversary A that distinguishes H₁,µ−1 from H₁,µ with non-negligible advantage. A simulator B that solves the mXDH assumption using A is given: a challenge tuple D = (<<(p, G, Gₚ, Gₚ, Gₚ, Gₚ, m₁, ⋱, mₙ, 1)>) and T = (X₁, ⋱, X₂ₙ). B interacts with A as follows.

Let (st, A, ((j₁, m₁), ⋱, (j₉, m₉))) be the output of A. The simulator B first sets the public parameters corresponding to the client index. For each j ∈ [N], if j ≠ jµ, S chooses a random exponent αj ∈ Zₚ and computes hₗ = g^αj. For the target client jµ, it sets hₗ = g^αµ. Next, for each j ∈ [N], B chooses a random exponent sj ∈ Zₚ and sets the secret key SKj = sj. It can generate the comparison key CKj, for any tuple (j, k) ∈ S since it knows secret keys sj and sk.

Let (j₉, m₉) be the i-th ciphertext query for a client index j₉. Let Eq(m, k) = prefix(m, k)||0xₖ and E₁(m, k) = prefix(m, k)||((0xₖ + 1) be encoded messages where m = x₁ ⋱ xₙ ∈ {0, 1}ⁿ. If (jₙ, *) ∈ S, then B simply creates a ciphertext by running the EORE.Encrypt algorithm since it know the secret key sₙ. If (j₉, *) ∉ S, then B creates the i-th EORE ciphertext ECⱼ₉ as follows:

- **Case** i < µ: It chooses random elements Rₖ,0 = (Rₖ,0,0, Rₖ,1,0), Rₖ,1 = (Rₖ,0,1, Rₖ,1,1) ∈ G × G for all k ∈ [n] and creates ECⱼ₉ = (⎨Rₖ,0, Rₖ,1⎬)k∈[n].

- **Case** i = µ: For each β ∈ {0, 1}, it computes F₁,β = H(β(k, m₁))^αₙ for all k ∈ [n]. For each Fₙ,β and Fₙ,₁, it sets Cₖ,0 = (Fₙ,0, X₂ₖ, g^bₙ) and Cₖ,₁ = (Fₙ,₁, X₂ₖ, g^b₂ₙ) and creates ECⱼ₉ = (⎨Cₖ,0, Cₖ,₁⎬)k∈[n].

- **Case** i > µ: It creates the EORE ciphertext ECⱼ₉ by running the EORE.Encrypt algorithm.

If T = (gᵃᵇ₁, ⋱, gᵃᵇ₂ₙ), then ECⱼ₉ is a ciphertext in H₁,µ−1. Otherwise, ECⱼ₉ is a ciphertext in H₁,µ. By the mXDH assumption, two experiments H₁,µ−1 and H₁,µ are computationally indistinguishable.
Lemma 4.5. The hybrid experiments $H_2$ and $H_3$ are computationally indistinguishable to the polynomial-time adversary assuming that the mXDH assumption holds.

Proof. We additionally define a sequence of hybrid experiments $H_2 = H_{2,0}, H_{2,1}, \ldots, H_{2,\tilde{q}} = H_3$ for some $\tilde{q}$ as follows.

$H_{2,\mu}$: Let $I = (j_1, \ldots, j_{\tilde{q}})$ be a tuple of challenge client index and $RI_{\mu} = \{i \in [\tilde{q}]: j_is are co-related indices\}$ be an index set of co-related client indices where $\mu \in [\tilde{q}]$. In this experiment, we change the generation of the $\mu$-th EORE ciphertext set with the index set $RI_{\mu}$. If $\ell \leq \mu$, the EORE ciphertexts in the $\ell$-th ciphertext set with $RI_\ell$ are changed to be random elements. Otherwise, the ciphertexts in the $\ell$-th ciphertext set with $RI_\ell$ 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 $H_{2,\mu-1}$ and $H_{2,\mu}$ are equally generated by using random elements.

Suppose there exists an adversary $A$ that distinguishes $H_{2,\mu-1}$ from $H_{2,\mu}$ with non-negligible advantage. A simulator $B$ that solves the mXDH assumption using $A$ is given: a challenge tuple $D = ((p, G, \hat{G}, \hat{G}_T, e), g, \hat{g}, g^\alpha, \{g^{b1}, \ldots, g^{b2n}\}_{t \in [n]}$ and $T = (\{X_{1,1}, \ldots, X_{1,2n}\}_{t \in [n]}$. $B$ runs the simulator $B_{bMC-ORE}$ of the Lemma 3.5 as a sub-simulator by submitting the challenge tuple of the mXDH assumption. Then $B$ that interacts with $A$ is described as follows.

Let $(st_A, S, ((j_1, m_1), \ldots, (j_{\tilde{q}}, m_{\tilde{q}})))$ be the output of $A$. The simulator $B$ first sets the public parameters corresponding to the client index. For each $j \in [N]$, $S$ chooses a random exponent $\alpha_j \in \mathbb{Z}_p$ and computes $h_j = g^{\alpha_j}$. For each tuple $(j,k) \in S$, $B$ obtains $CK_{j,k} = (K_0, K_1)$ by running $B_{bMC-ORE}$ and computes the comparison key $CK_{j,k} = (K_0, K_0^{\alpha_i}, K_1, K_1^{\alpha_i})$.

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

By the Lemma 3.5 two experiments $H_{2,\mu-1}$ and $H_{2,\mu}$ are computationally indistinguishable.

Lemma 4.6. The hybrid experiments $H_3$ and $H_4$ are indistinguishable to the polynomial-time adversary with the leakage function $L_S^{ORE}$ in the random oracle model.

Proof. Suppose there exists an adversary $A$ that distinguishes $H_3$ from $H_4$ with non-negligible advantage. We construct an efficient simulator $B$ for which the two distributions $H_3$ and $H_4$ are statistically indistinguishable. $B$ runs the simulator $B_{bMC-ORE}$ of the Lemma 3.6 as a sub-simulator.

Let $(st_A, S, ((j_1, m_1), \ldots, (j_{\tilde{q}}, m_{\tilde{q}})))$ be the output of $A$. The simulator $B$ first sets the public parameters corresponding to the client index. For each $j \in [N]$, $B$ chooses a random exponent $\alpha_j \in \mathbb{Z}_p$ and computes $h_j = g^{\alpha_j}$. For each tuple $(j,k) \in S$, $B$ obtains $CK_{j,k} = (K_0, K_1)$ by running $B_{bMC-ORE}$ and computes the comparison key $CK_{j,k} = (K_0, K_0^{\alpha_i}, K_1, K_1^{\alpha_i})$.

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

By the Lemma 3.6 the distributions $(CT_1', \ldots, CT_{\tilde{q}}')$ and $(\overline{CT_1'}, \ldots, \overline{CT_{\tilde{q}}'})$ of the basic MC-ORE ciphertexts output in $H_2$ and $H_3$ of the Theorem 3.2 are indistinguishable. Thus, it can be easily derived that the distributions $(EC_1, \ldots, EC_{\tilde{q}})$ and $(\overline{EC_1}, \ldots, \overline{EC_{\tilde{q}}})$ in $H_3$ and $H_4$ are also indistinguishable. Since $OC_j$ and $EC_j$ have been generated with respect to $L_j^{ORE}$ and $L_S^{ORE}$, $H_4$ corresponds to the ideal experiment.
5 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.

5.1 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.

5.2 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
Table 2: Performance comparison between range query methods

| $R$ | Simple Method (sec) | BinSearch Method (sec) | Hybrid Method (sec) |
|-----|---------------------|------------------------|---------------------|
| $2^8$ | 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                |

$CT = \{C_i,0, C_i,1\}_{i \in [n]}$ 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 $e(C_{n/4}, K_1)$ and $e(C_{n/4}, K_0)$ 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 on a message $m'_i$ 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 $m_i < m'_i$. 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 $m_i < m'_i$, then the server adds $L$ to the subset $S$ and repeats the above process by setting $D_j = R$. If $m'_i < m_i$, 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 $m_1, \cdots, m_{100}$ and $m'$ within a specific range $[0, R]$, and then we encrypted $m_1, \cdots, m_{100}$ 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 executes 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.

6 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.

References

[1] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In Gerhard Weikum, Arnd Christian König, and Stefan Deßloch, editors, ACM SIGMOD International Conference on Management of Data, pages 563–574. ACM, 2004.

[2] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O’Neill. Order-preserving symmetric encryption. In Antoine Joux, editor, Advances in Cryptology - EUROCRYPT 2009, volume 5479 of Lecture Notes in Computer Science, pages 224–241. Springer, 2009.

[3] Alexandra Boldyreva, Nathan Chenette, and Adam O’Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In Phillip Rogaway, editor, Advances in Cryptology - CRYPTO 2011, volume 6841 of Lecture Notes in Computer Science, pages 578–595. Springer, 2011.

[4] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In Elisabeth Oswald and Marc Fischlin, editors, Advances in Cryptology - EUROCRYPT 2015, volume 9057 of Lecture Notes in Computer Science, pages 563–594. Springer, 2015.

[5] David Cash, Feng-Hao Liu, Adam O’Neill, and Cong Zhang. Reducing the leakage in practical order-revealing encryption. Cryptology ePrint Archive, Report 2016/661, 2016.

[6] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. Practical order-revealing encryption with limited leakage. In Thomas Peyrin, editor, Fast Software Encryption - FSE 2016, volume 9783 of Lecture Notes in Computer Science, pages 474–493, 2016.

[7] F. Betül Durak, Thomas M. DuBuisson, and David Cash. What else is revealed by order-revealing encryption? In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, ACM Conference on Computer and Communications Security - CCS 2016, pages 1155–1166. ACM, 2016.
[8] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In FOCS 2013, pages 40–49. IEEE Computer Society, 2013.

[9] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, ACM Symposium on Theory of Computing - STOC 2009, pages 169–178. ACM, 2009.

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

[11] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In IEEE Symposium on Security and Privacy - SP 2017, pages 655–672. IEEE Computer Society, 2017.

[12] Florian Kerschbaum and Axel Schröpf. Optimal average-complexity ideal-security order-preserving encryption. In ACM Conference on Computer and Communications Security - CCS 2014, pages 275–286. ACM, 2014.

[13] Kevin Lewi and David J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, ACM Conference on Computer and Communications Security - CCS 2016, pages 1167–1178. ACM, 2016.

[14] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, Advances in Cryptology - EUROCRYPT ’99, volume 1592 of Lecture Notes in Computer Science, pages 327–346. Springer, 1999.

[15] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. J. ACM, 51(2):231–262, 2004.

[16] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In ACM Conference on Computer and Communications Security - CCS 2015, pages 644–655. ACM, 2015.

[17] Raluca Ada Popa, Frank H. Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In IEEE Symposium on Security and Privacy - SP 2013, pages 463–477. IEEE Computer Society, 2013.

[18] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. POPE: partial order preserving encoding. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, ACM Conference on Computer and Communications Security - CCS 2016, pages 1131–1142. ACM, 2016.

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