Order-Hiding Range Query over Encrypted Data without Search Pattern Leakage

Yi Dou*, Henry C. B. Chan and Man Ho Au

Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

** Corresponding author: yi.dou@connect.polyu.hk*

For cloud data storage, data privacy and security are two key concerns. Although sensitive data can be encrypted before they are stored in the cloud, the encrypted data can hardly be processed efficiently. Hence, a lightweight solution is required to satisfy both high security and high efficiency requirements. In this paper, we study the problem of range query over encrypted data. The main idea is to transform the range comparison to a privacy-preserving set intersection operation. To protect record privacy, our scheme builds searchable encrypted indexes for records that are secure against inference attack. To ensure the privacy of range queries, non-deterministic encryption, which has not been achieved in range query before, is proposed to hide the search pattern of queries. During range comparison, our scheme neither leaks the order relationship between the upper/lower bound of a range query and the encrypted index, nor produces false positives in the query results. We have implemented our scheme and evaluated its performance in comparison with other schemes. The comparison results indicate that our scheme has a shorter index size and search time than the order-revealing encryption (ORE) scheme when the processing unit is large. Meanwhile, our scheme only leaks the access pattern, and is proved to be more secure than existing schemes.

Keywords: Cloud security; Cloud data storage; Data privacy; Range query; Searchable symmetric encryption

Received 02 September 2017; revised 17 May 2018

1. INTRODUCTION

Cloud computing plays an important role in Information Communications and Technology (ICT) infrastructure to enhance system sustainability and resource management. Using an outsourcing approach, data can be stored, accessed and processed flexibly through the cloud for various applications, such as financial services, public health services, and traffic services. To use a cloud data storage service, data owners need to upload data to the cloud provider. Hence, data privacy is a major concern. Instead of storing plaintext data, encrypted data can be stored in order to protect data privacy, but it is neither convenient nor efficient to process encrypted data (e.g., data searching). When someone wants to search the encrypted data using certain query conditions, he/she needs to download the encrypted data from the cloud, decrypt them and process the query locally. This method is obviously not desirable in terms of efficiency and energy usage. It is also not practical when the terminal's processing power or network bandwidth is limited, such as when processing data through a mobile phone.

Searchable symmetric encryption (SSE) is a promising technique to tackle the aforementioned problem [1]. Currently, the majority of SSE schemes are secure index-based. This means that each encrypted record is associated with a number of secure indexes. Each index is created based on a record to be queried using one of its attribute values. To allow a cloud server to search the encrypted records, a trapdoor is created for each query. During the search phase, the server finds the matched records by comparing trapdoors of queries with indexes of records.

Most current SSE schemes focus on handling keyword search over encrypted data [2, 3, 4, 5]. Some dynamic SSE schemes are designed for similarity search [6] and multi-keyword query [7] while achieving forward-privacy protection. Their main purpose is to reduce information leakage when an encrypted database is updated. However, they do not address the problem of comparison query and range query over encrypted

data, which is required for some applications. For example, to find the shortest path between two nodes in a social network or a computer network without disclosing network connections, the corresponding cloud server needs to perform comparisons on the encrypted path information [8, 9]. In other scenarios, such as medical record reviewing or financial auditing, records are sensitive and queries are usually based on range values (e.g., certain time periods or a particular range of IP addresses) [10]. They also require performing secure range queries over encrypted data. Compared with keyword queries, there are more technical challenges in designing an effective and secure scheme for range queries on encrypted data.

In general, there are two types of solutions to the secure range queries on encrypted data. The first type (e.g., [11, 12, 13, 14, 15, 16]) aims to provide faster search time while disclosing certain information (e.g., the ordering between unmatched records and trapdoors). The second type (e.g., [17, 18, 19]) is designed to achieve higher security at the expense of extra cost (e.g., longer search time, large index storage space, false positives in the query results). For instance, Fully Homomorphic Encryption (FHE), which allows arithmetic operations on ciphertexts can be applied to support privacy-preserving range queries [19]. Although FHE can achieve high security, its computational cost is high. In this paper, we design and evaluate a privacy-preserving range query scheme to address the above-mentioned limitations. Our main contributions are summarised as follows:

- We provide a range comparison method that does not disclose the different binary bit(s) between a record and a query.
- We develop a trapdoor generation algorithm that can hide query search patterns.
- We prove that our proposed scheme is secure under the best security model available, without the restriction of identical search patterns. Furthermore, our scheme does not produce false positives in the query results.
- We implement and compare our scheme with OPE [12] and ORE [16] schemes. Our scheme is on average over 16 times faster than the OPE scheme in index generation. For each range comparison, our scheme is on average 3.89 times faster than the ORE scheme when the processing unit is 2 bytes.

The rest of the paper is organised as follows. Related works are introduced in Section 2. Section 3 provides the scheme's general construction and security goal. Section 4 describes the building block utilised in the scheme. Section 5 presents the details of our proposed privacy-preserving range query scheme. Experimental setup and results are illustrated in Section 6. Section 7 analyses the security of the scheme and proves that it can achieve the defined security goals. Section 8 presents the conclusion and future work.

2. RELATED WORKS

We classify the existing schemes into two categories. The first type of scheme [11, 12, 13, 14, 15, 16] performs a range query $[a, b]$ by checking whether the data item is larger than the lower bound ($a \leq d_u$) and smaller than the upper bound ($d_u \leq b$) of a query. The comparison result directly discloses whether the unsatisfied data item is larger/smaller than the upper/lower bound of a query. The second type of scheme [17, 18] conducts a range query by treating the query range as a single keyword. This type of scheme only outputs whether the data values are within the query range or not (i.e., $d_u \in [a, b]$). The server is unable to learn the ordering of unsatisfied records. Therefore, it is more secure than the first type. However, it introduces plenty of unmatched query results, and requires a large amount of index storage space. To maintain similar search time complexity (i.e., as $O(\log\ N)$ or even faster), these schemes usually organise indexes of the entire dataset using special data structures.

Order Preserving Encryption (OPE). Order preserving encryption, one of the methods for supporting numerical comparison between ciphertexts, has been studied in [11, 12, 13, 14]. OPE ciphertexts are deterministic and preserve the numerical order between their plaintexts, that is $a \leq b$, if and only if $OPE(a) \leq OPE(b)$. Since the server can directly obtain the ordering of data items from their OPE ciphertexts, any comparison-based index structures (e.g., B+ Tree) used for indexing plaintext data can be directly applied to the OPE ciphertexts. Although OPE can achieve the same search efficiency as in the plaintext cases, it is unable to prevent "inference attacks" [20]. This attack mainly leverages the ordering and the frequency of data items disclosed from OPE ciphertexts.

Order Revealing Encryption (ORE). To prevent "inference attacks", Lewi and Wu proposed both a small-domain ORE scheme and a large-domain ORE scheme in [16]. For both schemes, the comparison is performed between the right ciphertext of data $Enc_R(d_u)$ and left ciphertext of the query boundary $Enc_L(a)/Enc_L(b)$. The small-domain ORE scheme can achieve the best-possible semantic security, which is robust against inference attacks. However, the length of each right ciphertext grows linearly in the size of the entire plaintext space. To improve comparison efficiency and also reduce the index size of the small-domain ORE scheme, the large-domain ORE scheme is designed to build indexes by grouping the message space of data into blocks. However, this approach leaks the first block that is different between two ciphertexts during the comparison. This leakage implies the ordering between data items. When the block size is relatively small, the server can precisely estimate

the relative ordering information. When the block size is large, however, the ciphertext size of ORE grows exponentially. The left ciphertext is deterministically created in the ORE scheme for each query condition. The search pattern of queries in the ORE scheme is leaked from trapdoors to the server. The ORE scheme also inherits the weakness of the comparison query. Throughout the searching result, the entire encrypted dataset can be divided into three ordered parts ($\mathcal{R}_1 < \mathcal{R}_2 \in [a, b] < \mathcal{R}_3$). The ORE scheme has been used in other scenarios. Shen et al. adopted ORE to support the approximate constrained shortest distance queries over encrypted graphs [9]. To filter out paths in an encrypted graph with the constrained total cost, the scheme encrypts the cost of each edge using ORE encryption, and designs an efficient tree-based ORE ciphertext comparison protocol. However, this scheme also inherits the weaknesses of the ORE scheme, such as disclosing the search patterns and order relationship among the costs.

Comparable Encryption (CE). Comparable encryption was proposed by Furukawa [21, 22]. The comparable encryption ciphertexts are semantically secure against inference attack, and they are unable to be directly compared with one another. To support a comparison, comparable encryption generates tokens for the query boundaries. In the token generation, comparable encryption adopts prefix-preserving encryption (PPE). When any two data items have the same first n digits of prefix strings, their PPE tokens must also have the same n high elements as their prefixes. Hence, the comparison seeks to check the equivalence between ciphertext and token, starting from the high elements until finding the first one that is different. Comparable encryption has a similar weakness to ORE, which leaks the length of the longest common prefix string of record value and query boundary [23]. Moreover, it tells the attacker about the comparison operator, since it separately compares data items with the upper and lower bounds of a range query. This leakage indicates the relative numerical differences between data items to the same query. Since tokens of queries are deterministic and prefix preserving, the repetition and numerical order between upper/lower bounds of issued queries are also leaked in comparable encryption.

Privacy Bloom filter tree (PBtree). Li et al. proposed a privacy-preserving range query scheme that can achieve index indistinguishability [17]. The scheme is to test whether in a range query $d_u \in [a, b]$, the prefix set of d_u and prefix union set of $[a, b]$ have the same elements [24]. This scheme can avoid a full dataset scan, and order leakage between left and right nodes during the binary search as well. It organises the prefix sets of data items in random order using a special complete binary tree called PBtree. Specifically, the root node stores the prefixes union of all of the data items. It then recursively splits the data items of each node into its left and right child node. This split only ensures that each child node has an equal number of data items. Since data items are randomly split, the left child data items are not necessarily smaller than those of the right child. To achieve the PBtree structure, this scheme sacrifices the storage cost of $O(N \log N \log M)$, where M is the domain size of data items. To improve the speed of checking the overlap between two prefix sets, the scheme employs the data structure of Bloom filter [25], but this creates false positives in the query results. Although the PBtree structure seeks to achieve a sublinear search time, the actual search time is $\Omega(\log N \log Q + R)$, where Q and R are the sizes of query range and result, respectively. This search time has no upper bound because of the random placement of the data items in the PBtree and possible false positive results [18]. This scheme is only proved to be secure under certain conditions (i.e., non-adaptive adversaries following Goh's definition [26]). Goh's security definition does not guarantee trapdoor privacy. The trapdoor generation is always deterministic, and thus the search pattern is disclosed [24].

Tree-like Directed Acyclic Graph (TDAG). Searchable Symmetric Encryption is widely used for keyword search over encrypted data. Inspired by this idea, Demertzis et al. proposed the concept of Range Searchable Symmetric Encryption (RSSE) [18]. This concept turns the problem of range query into a multi-keyword search problem, such that any secure SSE schemes can be employed to realize the RSSE concept. IND-CKA2 (indistinguishability against adaptive chosen keyword attacks) is the strongest security model available for SSE schemes, introduced by Curtmola et al. [27]. The scheme proposed by Demertzis et al. also satisfies the IND-CKA2, but has additional leakages. This means that an attacker can learn nothing more than the formulated leakages. In this scheme, a range query condition is replaced by several sub-ranges, with each sub-range represented by a keyword. Data items belonging to the same sub-range are considered to be documents containing the same keyword. To prevent multiple sub-ranges from being associated with the same data item, the scheme needs to duplicate records into all sub-ranges having its value. However, this duplication requirement generates a much larger dataset (i.e., compared to the original dataset). With the aim of reducing storage cost and the number of sub-ranges, the scheme designs a new structure called TDAG (i.e., Tree-like Directed Acyclic Graph). A TDAG tree is constructed by inserting a middle node between any two peer nodes on the binary tree of sub-ranges. During the search phase, each query is mapped to a single TDAG node based on the lowest common ancestor node that covers the query range. However, the TDAG structure creates an unacceptable number of false positive results when the data skew rate is high. These false positives reduce the search time to $O(N)$. Queries with similar range values are targeted to the same node on the TDAG structure, such that the search pattern is partially leaked.

TABLE 1. Comparison of privacy-preserving range query schemes

Scheme	Index Order	Comparison Operator	Search Pattern	Space Usage	Search Time	False Positive
OPE [12]	leak	leak	leak	$O(N)$	$O(\log N)$	no
ORE [16]	no / leak	leak	leak	$O(N\ 2^b\ (\log M)/b)$	$O(N)$ / $O(\log N)$	no
CE [21]	leak	leak	leak	$O(N\ \log M)$	$O(N)$	no
PBtree [17]	no	no	leak	$O(N\ \log N\ \log M)$	$\Omega(\log N\ \log Q + R)$	$O(R)$
TDAG [18]	no	no	partial leak	$O(N\ \log M)\ \times$ Size(record)	$O(N)$	$O(N)$
Our Scheme	no / leak	no	no	$O(N\ \log M)$	$O(N)$ / $O(\log N)$	no

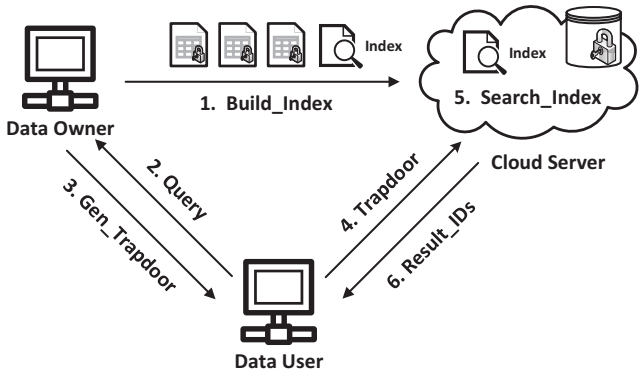

FIGURE 1. Architecture of range query on encrypted data in cloud computing.

Our Scheme. Table 1 shows the comparison of different privacy-preserving range query schemes. Parameter N is the dataset size, R is the result size, Q is the query range size, M is the domain size of data items, and b is the block size in bits of the ORE scheme. To summarise, the first three range query over encrypted data schemes [16, 12] provide higher search efficiency, but a lower security guarantee than those of the last two range query over encrypted data schemes [17, 18]. Compared to existing schemes, our scheme supports both types of range query (i.e., $a \leq d_u$ and $d_u \in [a, b]$) and can hide the comparison operator during the query, even using the first type of comparison approach. The search pattern of queries (i.e., non-deterministic trapdoor) is also concealed. Compared with the listed schemes, our scheme does not have a high space cost, and produces no false positives in the query results. Our scheme can support binary search (i.e., search time is $O(\log N)$) when ciphertexts are sorted before searching, which inevitably leaks the ordering of the index during the query.

3. PROBLEM STATEMENT

We present the system model, algorithm construction, and security definitions of our scheme in this section.

3.1. System Model

The system model discussed in our scheme is shown in Figure 1. There are three parties: a data owner, a data user, and a cloud server. The data owner would like to store a collection of sensitive records to the cloud. However, it does not fully trust the cloud provider. Thus, before uploading records to the cloud, it first encrypts each record. To provide the cloud with the ability to perform relational operations on encrypted records without decryption, the data owner associates each encrypted record with a secure searchable index generated using the attribute value of records to be queried. A valid data user submits a trapdoor - which is obtained from the data owner generated from plaintext query - to the server in the cloud. After obtaining the trapdoor, the server searches the matching records remotely via indexes, and returns the ID of satisfied records as the query results to the data user.

In our model, the data owner and authorised data user are regarded as fully trusted. They communicate through a secure channel. In practice, clouds are managed by well-established IT companies. In cases of attack, it is more likely for cloud providers to conduct passive attacks instead of active attacks on specific users. Therefore, in our scheme, we assume the cloud server to be a semi-honest (honest-but-curious) adversary, which is trusted to correctly execute required communication protocols and algorithms. At the same time, the cloud server actively deduces the sensitive information of records and the content of received queries. The semi-honest adversary model has commonly been adopted in existing privacy-preserving range query and keyword searchable symmetric encryption schemes [12, 17, 18, 24, 26, 27]. We make the same assumption as before.

3.2. Notation and Definition

Notations and functions in the rest of the paper are defined as follows.

- $\mathcal{R} = (r_1, ..., r_N)$ is a collection of records, where r_u is the ID of the uth record.

- $\mathcal{D} = (d_1, ..., d_N)$ is a collection of attribute values from $\mathcal{R}$. Each attribute value $d_u (1 \leq u \leq N)$ contained in a record r_u, where $d_u \in \mathbb{Z}_{2^\ell}$ and ℓ is the bit length of attribute values.
- $\mathcal{I} = (I_{d_1}, ..., I_{d_N})$ is a collection of encrypted searchable indexes based on $\mathcal{D}$, where I_{d_u} is the index of d_u.
- $|I_{d_u}|$ is the index size, which is the number of row vectors and polynomial random nonce pairs used in generating I_{d_u}.
- $Q = [w_L, w_H]$ is a range query, where $w_L, w_H \in \mathbb{Z}_{2^\ell}$, w_L is the lower bound, and w_H is the upper bound of query Q, respectively.
- T_Q is the secure trapdoor of the query Q.
- $|T_Q|$ is the trapdoor size, which is the number of column vectors in T_Q.
- $\mathcal{D}(Q)$ is the set of query Q's result on the collection $\mathcal{D}$, where the attribute values of records in $\mathcal{D}(Q)$ belongs to the interval $[w_L, w_H]$.

Before going into the details, we first define the following main algorithms of our scheme.

DEFINITION 3.1. *(*Privacy Preserving Range Query Scheme*): A searchable symmetric encryption range query scheme includes four algorithms.*

- $(\mathsf{sk}, \mathsf{params}) \leftarrow \textbf{Gen\_Key}(\lambda)$: is executed by the data owner. Taking as input security parameter λ, the algorithm generates secret key sk and system parameter params.
- $\mathcal{I} \leftarrow \textbf{Bld\_Index}(\mathsf{sk}, \mathcal{D})$: is run by the data owner to support the cloud server, with the capability of searching on $\mathcal{R}$. It takes as input secret key sk and the attribute value collection $\mathcal{D}$ and outputs encrypted searchable indexes $\mathcal{I}$.
- $T_Q \leftarrow \textbf{Gen\_Trapdoor}(\mathsf{sk}, Q)$: is run by the data owner to create a trapdoor for a given query Q. It takes both secret key sk and range query Q: $[w_L, w_H]$ as the input. It outputs trapdoor T_Q for query Q.
- $\mathcal{D}(Q) \leftarrow \textbf{Rag\_Search}(\mathcal{I}, T_Q)$: is run by the cloud server to determine the query result. It takes as input encrypted searchable indexes $\mathcal{I}$ and trapdoor T_Q. It outputs record set $\mathcal{D}(Q)$ as the query result.

Correctness. We consider a privacy-preserving range query scheme over a collection of records is correct, if for all records in $\mathcal{D}(Q)$,

$$\Pr[r_u \in \mathsf{Rag\_Search}(\mathcal{I}, T_Q) | d_u \in [w_L, w_H]] = 1 - \mathsf{negl}(\lambda).$$

3.3. Security Goals

Since the records are encrypted by the data owner before outsourcing to the cloud server, our scheme aims to preserve the privacy of the searchable index and trapdoor during and after queries from the following aspects:

1) The attribute value used to generate the index is part of the record contents. As a result, the cloud server should not learn attribute value d_u from its index I_{d_u} or from trapdoor T_Q of any issued query.
2) The cloud server should not deduce whether the indexes of two different records are built from the same attribute value (hide the frequency), and whether the attribute value of one record is larger or smaller than that of another record (hide the order).
3) The privacy of a trapdoor is inherently linked to the privacy of the index. Hence, the cloud server should not distinguish the value of w_L and w_H from its trapdoor T_Q and from the received record indexes.
4) The cloud server is unable to determine whether two trapdoors are created from the same query range or not (hide the search pattern), and whether the upper or lower bound of one query is larger or smaller than that of another query.
5) For each unmatched record $r_u \notin \mathcal{D}(Q)$, the cloud server should not know if $d_u > w_H$ or $d_u < w_L$ and in which bits of d_u and in which bits of w_H/w_L that differentiates d_u from query Q.

Since we are unable to enumerate all possible attacks, we introduce the following security definitions.

3.4. Security Model

Let $\mathsf{SSE}^{\mathsf{RAG}}$ = ($\mathsf{Gen\_Key}$, $\mathsf{Bld\_Index}$, $\mathsf{Gen\_Trapdoor}$, $\mathsf{Rag\_Search}$) be our privacy-preserving range query scheme. We analyse the security of $\mathsf{SSE}^{\mathsf{RAG}}$ under the game-based IND-CKA2 security model [27] with appropriate modifications. Since trapdoors are deterministically generated in Curtmola et al.'s scheme, adversaries in the game of IND-CKA2 model can only ask for the trapdoors of queries in pairs. And the IND-CKA2 model includes the search pattern of queries as one of its leakages. However, in our scheme $\mathsf{SSE}^{\mathsf{RAG}}$, the trapdoors are not deterministically created. The adversary in the game of our security model can make a request for the trapdoor of the single query.

To make the security definition of IND-CKA2 fit our stronger security guarantee, we relax the assumption of the same search pattern in IND-CKA2, and redefine two games in Section 3.4.1 and Section 3.4.2 to prove the security of indexes/ciphertext and the security of trapdoors separately. In each of the two games,

challenger $\mathcal{C}$ executes the actual algorithms in $\mathsf{SSE}^{\mathsf{RAG}}$. An adversary $\mathcal{A}$ adaptively sends queries to challenger $\mathcal{C}$ based on all previously obtained indexes, trapdoors, and search results. Before we present our security model, we first formulate the information leakages that arise from our scheme.

Definition 3.2. *(**Information Leakage**) Function $\mathcal{L}(\mathcal{D}, |T_Q|, Q)$ describes the leakage from a range query Q over a collection of attribute values $\mathcal{D}$, that is*

$$\mathcal{L}(\mathcal{D}, Q) = \langle |I_{d_1}|, ..., |I_{d_N}|, |T_Q|, \mathcal{D}(Q) \rangle.$$

comprised of the index size of attribute values in $\mathcal{D}$, trapdoor size $|T_Q|$, and the access pattern $\mathcal{D}(Q)$ (i.e., the record set of range query Q's result).

3.4.1. Ciphertext Indistinguishability Security

We use the following game to formally define the requirement that the adversary learns nothing about the attribute values beyond their index sizes and access patterns.

Game 1

- **Setup**: Challenger $\mathcal{C}$ creates a large collection of attribute values $\mathbb{D}$, a sequence of range queries $\mathbb{Q}$ and gives them to the adversary $\mathcal{A}$. $\mathcal{C}$ runs $\mathsf{Gen\_Key}(\lambda)$ to generate secret key sk and system parameter params. $\mathcal{C}$ keeps secret key sk and sends params to $\mathcal{A}$.
- **Phase 1**: Adversary adaptively issues q_1 pairs of requests based on past received indexes and trapdoors, where the request $1 \leq i \leq q_1$ is shown as follows:
  - Index generation request for an attribute value collection $\mathcal{D}_i \in \mathbb{D}$: The challenger runs $\mathsf{Bld\_Index}(\mathsf{sk}, \mathcal{D}_i)$ on a collection of attribute values $\mathcal{D}_i$, and forwards the index $\mathcal{I}_i$ to the adversary.
  - Trapdoor generation request for a range query $Q_i \in \mathbb{Q}$: The challenger runs $\mathsf{Gen\_Trapdoor}(\mathsf{sk}, Q_i)$ on a range query Q_i, and forwards the index T_{Q_i} to the adversary.
- **Challenge**: The adversary submits two plaintext collections of attribute values $\boldsymbol{\mathcal{D}_0}$ and $\boldsymbol{\mathcal{D}_1} \in \mathbb{D}$. The challenger randomly flips a bit $b \in \{0, 1\}$ and responds to the adversary with the index $\mathcal{I}_b \leftarrow \mathsf{Bld\_Index}(\mathsf{sk}, \mathcal{D}_b)$ as its challenge ciphertext.
- **Phase 2**: For request $q_1 + 1 \leq i \leq q$, adversary repeats the same process as in Phase 1 and finally obtains $\langle \boldsymbol{\mathcal{I}_1}, ..., \boldsymbol{\mathcal{I}_q}, \boldsymbol{\mathcal{I}_b} \rangle$ and $\langle \boldsymbol{T_{Q_1}}, ..., \boldsymbol{T_{Q_q}} \rangle$.
- **Guess**: With the restriction that $\boldsymbol{\mathcal{D}_0}$ and $\boldsymbol{\mathcal{D}_1}$ cause the same leakage under all chosen queries

$$\mathcal{L}(\boldsymbol{\mathcal{D}_0}, Q_1) = \mathcal{L}(\boldsymbol{\mathcal{D}_1}, Q_1),$$
$$.....$$
$$\mathcal{L}(\boldsymbol{\mathcal{D}_0}, Q_q) = \mathcal{L}(\boldsymbol{\mathcal{D}_1}, Q_q),$$

adversary guesses a bit $b' \in \{0, 1\}$. If $b = b'$, we consider that the adversary wins the index security game.

Definition 3.3. *(**Ciphertext Indistinguishability Security**): We say that $\mathsf{SSE}^{\mathsf{RAG}}$ is ciphertext/index secure in terms of adaptive indistinguishability if for all polynomial-sized adversary $\mathcal{A}$, the advantage in winning Game 1 is less than a negligible function of λ.*

$$\Pr[b' = b] - \frac{1}{2} \leq \mathsf{negl}(\lambda).$$

3.4.2. Trapdoor Indistinguishability Security

We use the following game to formally define the requirement that the adversary learns nothing about the range query values beyond their trapdoor sizes and access patterns.

Game 2

- **Setup**: Challenger $\mathcal{C}$ creates a large collection of attribute values $\mathbb{D}$, a sequence of range queries $\mathbb{Q}$ and gives them to the adversary $\mathcal{A}$. $\mathcal{C}$ runs $\mathsf{Gen\_Key}(\lambda)$ to generate secret key sk and system parameter params. $\mathcal{C}$ keeps secret key sk and sends params to $\mathcal{A}$.
- **Phase 1**: Adversary adaptively issues q_1 pairs of requests based on past received indexes and trapdoors, where the request $1 \leq i \leq q_1$ is shown as follows:
  - Index generation request for an attribute value collection $\mathcal{D}_i \in \mathbb{D}$: The challenger runs $\mathsf{Bld\_Index}(\mathsf{sk}, \mathcal{D}_i)$ on a collection of attribute values $\mathcal{D}_i$, and forwards the index $\mathcal{I}_i$ to the adversary.
  - Trapdoor generation request for a range query $Q_i \in \mathbb{Q}$: The challenger runs $\mathsf{Gen\_Trapdoor}(\mathsf{sk}, Q_i)$ on a range query Q_i, and forwards the index T_{Q_i} to the adversary.
- **Challenge**: The adversary submits two range queries $\boldsymbol{Q_0}$ and $\boldsymbol{Q_1} \in \mathbb{Q}$. The challenger randomly flips another bit $c \in \{0, 1\}$ and replies to the adversary with the trapdoor $T_{Q_c} \leftarrow \mathsf{Gen\_Trapdoor}(\mathsf{sk}, Q_c)$ as its challenge trapdoor.
- **Phase 2**: For request $q_1 + 1 \leq i \leq q$, adversary repeats the same process as in Phase 1 and finally obtains $\langle \boldsymbol{\mathcal{I}_1}, ..., \boldsymbol{\mathcal{I}_q} \rangle$ and $\langle \boldsymbol{T_{Q_1}}, ..., \boldsymbol{T_{Q_q}}, \boldsymbol{T_{Q_c}} \rangle$.
- **Guess**: With the restriction that $\boldsymbol{Q_0}$ and $\boldsymbol{Q_1}$ cause the same leakage under all chosen attribute value collections

$$\mathcal{L}(\mathcal{D}_1, \boldsymbol{Q_0}) = \mathcal{L}(\mathcal{D}_1, \boldsymbol{Q_1}),$$
$$.....$$
$$\mathcal{L}(\mathcal{D}_q, \boldsymbol{Q_0}) = \mathcal{L}(\mathcal{D}_q, \boldsymbol{Q_1}),$$

adversary guesses a bit $c^{'} \in \{0,1\}$. If $c = c^{'}$, we consider that the adversary wins the trapdoor security game.

DEFINITION 3.4. *(**Trapdoor Indistinguishability Security**): We say that* $\mathsf{SSE}^{\mathsf{RAG}}$ *is trapdoor secure in terms of adaptive indistinguishability if for all polynomial-sized adversary* $\mathcal{A}$*, the advantage in winning Game 2 is less than a negligible function of* λ*.*

$$\Pr[c^{'} = c] - \frac{1}{2} \leq \mathsf{negl}(\lambda).$$

DEFINITION 3.5. *(**Indistinguishability Security**): We say that scheme* $\mathsf{SSE}^{\mathsf{RAG}}$ *is both ciphertext and trapdoor secure in terms of adaptive indistinguishability if for all polynomial-sized adversary* $\mathcal{A}$*, the advantage in winning both Game 1 and Game 2 is less than a negligible function of* λ*.*

4. BUILDING BLOCK

The building block of our scheme is the 0/1 encoding first proposed by Lin and Tzeng to address the Millionaires' Problem [28]. The key idea of 0/1 encoding is to turn data comparison to the problem of finding the intersection of two sets. For a comparison $a > b$, it needs to find a common element between the 1-encoding set of a and 0-encoding set of b. Let $s = s_1 s_2 ... s_\ell$ denote the binary string of s and $s_1 s_2 ... s_h$ represent the binary string of the first h digits of s (i.e., the h-length prefix string of s). The prefix string set P_s of s is defined as follows

$$P_s = \{s_1 s_2 ... s_h | 1 \leq h \leq \ell\}. \tag{1}$$

For each prefix string $s_1 s_2 ... s_h$ ending up with $s_h = 0$, we write the binary string $s_1 s_2 ... s_{h-1} 1$ as one of the elements in the 0-encoding set S_s^0. For each prefix string $s_1 s_2 ... s_h$ ending up with $s_h = 1$, we directly write its prefix string $s_1 s_2 ... s_h$ as one of the elements in the 1-encoding set S_s^1. Two binary string sets S_s^0 and S_s^1 are defined as the 0-encoding and 1-encoding sets of s, such that

$$S_s^0 = \{s_1 s_2 ... s_{h-1} 1 | s_h = 0, 1 \leq h \leq \ell\}, \tag{2}$$

$$S_s^1 = \{s_1 s_2 ... s_h | s_h = 1, 1 \leq h \leq \ell\}, \tag{3}$$

where ℓ is the number of binary digits in the binary string of s. For any two numbers a and b with bit-length of ℓ, their 0-encoding set and 1-encoding sets have the following properties [28].

$$\begin{cases} a > b \iff S_a^1 \cap S_b^0 \neq \emptyset, \\ a \leq b \iff S_a^1 \cap S_b^0 = \emptyset. \end{cases} \tag{4}$$

When $S_a^1 \cap S_b^0 \neq \emptyset$, their common element is denoted as

$$a_1 a_2 ... a_{h-1} 1|_{a_h=1} = b_1 b_2 ... b_{h-1} 1|_{b_h=0}$$

which is equivalent to

$$a_1 a_2 ... a_{h-1} 0|_{a_h=1} = b_1 b_2 ... b_{h-1} 0|_{b_h=0}.$$

Then, the right side element $b_1 b_2 ... b_{h-1} 0|_{b_h=0}$ belongs to P_b the prefix string set of number b. Based on the left side element $a_1 a_2 ... a_{h-1} 0|_{a_h=1}$, we define the following new 1-encoding set $\widetilde{S_s^1}$

$$\widetilde{S_s^1} = \{s_1 s_2 ... s_{h-1} 0 | s_h = 1, 1 \leq h \leq \ell\}. \tag{5}$$

Since all elements of $\widetilde{S_s^1}$ end up with $a_h = 0$, we can convert the determination of $S_a^1 \cap S_b^0$ to the comparison between $\widetilde{S_a^1}$ and P_b, that is $S_a^1 \cap S_b^0 = \widetilde{S_a^1} \cap P_b$. Conversely, we can compare P_a with the 0-encoding set of b. Since each element of S_a^1 belongs to P_a and all elements of S_b^0 end up with $b_h = 1$, we can get $S_a^1 \cap S_b^0 = P_a \cap S_b^0$.

Therefore, $\widetilde{S_s^1}, S_s^0$ and P_s satisfy the same properties as the 0/1 encoding when comparing two numbers a and b.

$$\begin{cases} a > b \iff \widetilde{S_a^1} \cap P_b \neq \emptyset \text{ or } P_a \cap S_b^0 \neq \emptyset, \\ a \leq b \iff \widetilde{S_a^1} \cap P_b = \emptyset \text{ or } P_a \cap S_b^0 = \emptyset. \end{cases} \tag{6}$$

Here is an example of how to compare two numerical values. Let $a = 9 = (1001)_2$ and $b = 14 = (1110)_2$ denote two binary strings with 4 bits. Based on definitions in Equation (1), (2) and (5), we obtain

$$P_9 = \{1, \underline{10}, 100, 1001\}, S_9^0 = \{\underline{\underline{11}}, 101\}, \widetilde{S_9^1} = \{0, 1000\},$$

$$P_{14} = \{1, \underline{11}, 111, 1110\}, S_{14}^0 = \{1111\}, \widetilde{S_{14}^1} = \{0, \underline{10}, 110\}.$$

Since $\widetilde{S_{14}^1} \cap P_9 = \{10\} \neq \emptyset$ and $P_{14} \cap S_9^0 = \{11\} \neq \emptyset$, we learn that $14 > 9$. $\widetilde{S_9^1} \cap P_{14} = \emptyset$ and $P_9 \cap S_{14}^0 = \emptyset$, we learn that $9 \leq 14$.

Based on Equation (6), we obtain the following properties when comparing a value d with a closed interval $[w_L, w_H]$.

$$\begin{cases} d \notin [w_L, w_H] \\ \qquad \iff d < w_L \text{ and } d > w_H \\ \qquad \iff P_d \cap \widetilde{S_{w_L}^1} \neq \emptyset \text{ and } P_d \cap S_{w_H}^0 \neq \emptyset. \end{cases} \tag{7}$$

This means that we can use the same prefix string set of d to compare it with the different encoding sets of upper bound w_H and lower bound w_L of a range query.

5. OUR PRIVACY PRESERVING RANGE QUERY SCHEME

We present our privacy-preserving range query scheme in this section. We assume that the database records have already been encrypted before being stored in the cloud server.

5.1. Scheme Overview

As introduced in the previous sections, the cloud server will match the index of records with the query trapdoor to target the satisfied database records. To provide

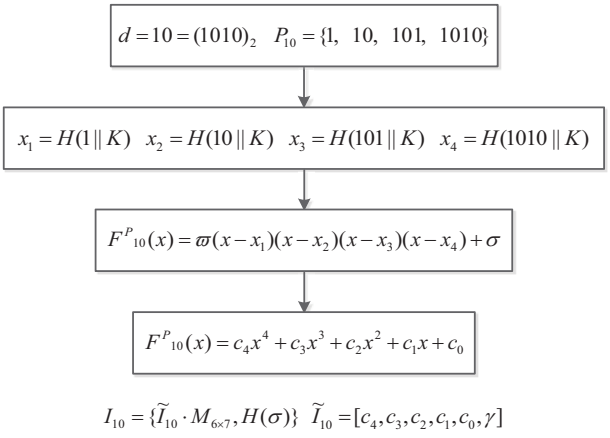

FIGURE 2. Example of index generation.

a security guarantee, our scheme needs to achieve indistinguishability of both the index and trapdoor. The building block of our scheme is 0/1 encoding introduced in Equation (4) and (6), which converts the data comparison into the calculation of the intersection between their corresponding 0/1 encoding set $\widetilde{S_a^1}/S_a^0$ and prefix string set P_b.

Inspired by the idea of private set intersection (PSI) [29], our scheme represents the encoding elements in P_b as the roots of polynomial function. During the comparison phase, the encoding elements in $\widetilde{S_a^1}$ and S_a^0 are plugged into the polynomial function. This design can prevent the server from knowing in which binary bit the two values differ. To hide the search pattern of different queries, we use an invertible matrix with random numbers, such that the cloud server is unable to distinguish between different queries. As the encoding elements are placed in shuffled order during the generation of the trapdoor, the cloud server cannot locate the intersection results from the upper or lower bound of the queries. The details of our scheme are indicated below.

5.2. Scheme Details

• **Gen_Key**(λ) : First, the Gen_Key algorithm is performed by the data owner to determine a modulus p and a secure keyed hash function $H : \{0,1\}^{\lambda} \times \{0,1\}^{\ell} \rightarrow \{0,1\}^{s(\lambda)}$, in which $s(\lambda)$ is a quantity polynomial of the security parameter λ. Then, Gen_Key generates a secret key K and a random invertible matrix $M_{(\ell+3)\times(\ell+3)}$, in which ℓ is the bit length of the attribute value to be indexed. The system parameter params is the pair (p, H) sent to the cloud server and the secret key sk is the pair (K, M) kept by the data owner.

• **Bld_Index**$(\mathsf{sk}, \mathcal{D})$: The Bld_Index algorithm is performed by the data owner to generate indexes for records. For each attribute value d_u in the collection of $\mathcal{D}$, Bld_Index algorithm executes the following steps. Figure 2 illustrates an example of index generation.

1) First, the Bld_Index algorithm computes the ℓ length binary strings for d_u and calculates the prefix string set $P_{d_u} = \{e_1, e_2, ..., e_\ell\}$ of d_u. Then, for each element $e_i \in P_{d_u} (1 \leq i \leq \ell)$, it computes the corresponding hash value $x_i = H(e_i||K) \pmod p$ by the secret key K generated in Gen_Key.

2) To hide in which bit of d_u that differs d_u from a range query, the algorithm constructs the following polynomial function $F_{d_u}^P(x)$ for all hash values $\{x_1, ..., x_\ell\}$. A pair of polynomial random nonces ϖ_u and σ_u is embedded in $F_{d_u}^P(x)$ in order to distinguish the indexes of different records with the same attribute value,

$$\begin{aligned} F_{d_u}^P(x) &= \varpi_u(x - x_1)(x - x_2)...(x - x_\ell) + \sigma_u \pmod p \\ &= c_\ell x^\ell + c_{\ell-1} x^{\ell-1} ... c_1 + c_0, \end{aligned} \tag{8}$$

where $\{x_1, ..., x_\ell\}$ are the hash values of elements in P_{d_u}.

3) First, the algorithm constructs a row vector $\tilde{I}_{d_u}$ with $\ell + 2$ elements, where $c_\ell, ..., c_0$ are the coefficients of $F_{d_u}^P(x)$, and γ_u is a random nonce:

$$\tilde{I}_{d_u} = [c_\ell, c_{\ell-1} ..., c_1, c_0, \gamma_u]. \tag{9}$$

Then, the algorithm constructs two matrixes $[M]_{(\ell+2)\times(\ell+3)}$ and $[M^{-1}]_{(\ell+3)\times(\ell+2)}$ based on the matrix M in sk. $[M]_{(\ell+2)\times(\ell+3)}$ is constructed by removing the $(\ell + 2)$th row of M and $[M^{-1}]_{(\ell+3)\times(\ell+2)}$ is obtained by deleting the $(\ell + 3)$th column of M^{-1}.

4) $\tilde{I}_{d_u}$ is right multiplied by matrix $[M]_{(\ell+2)\times(\ell+3)}$ to obtain

$$\begin{aligned} I_{d_u}^{'} &= \tilde{I}_{d_u} \cdot [M]_{(\ell+2)\times(\ell+3)} \\ &= [c_\ell, ..., c_1, c_0, 0, \gamma_u] \cdot [M]_{(\ell+3)\times(\ell+3)}. \end{aligned} \tag{10}$$

The Bld_Index algorithm outputs the encrypted searchable index of d_u as a 2-tuple of

$$I_{d_u} = \{I_{d_u}^{'}, H(\sigma_u)\} \tag{11}$$

where $H(\sigma_u)$ is the hash value of σ_u. Finally, the data owner submits encrypted searchable index $\mathcal{I} = (I_{d_1}, ..., I_{d_N})$ to the cloud sever.

• **Gen_Trapdoor**(sk, Q) : The Gen_Trapdoor algorithm is performed by the data owner to generate the trapdoor for a query $Q : [w_L, w_H]$ requested by the data user. Figure 3 illustrates an example of trapdoor generation.

1) The algorithm first computes two ℓ length binary strings for w_L and w_H. Then, it calculates new 1-encoding set $\widetilde{S_{w_L}^1}$ for lower bound w_L based on Equation (5) and the 0-encoding set $S_{w_H}^0$ for upper bound w_H based on Equation (2).

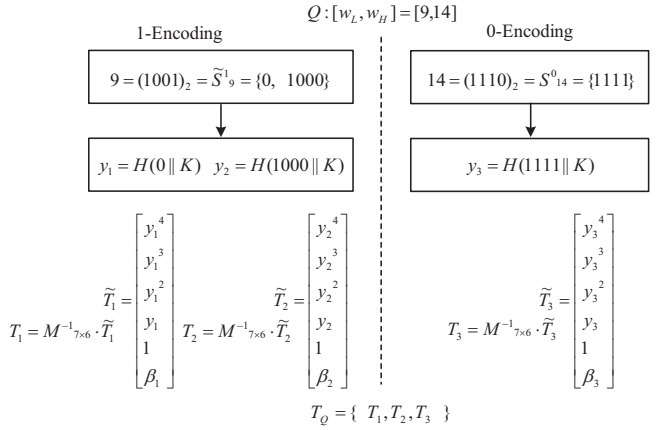

FIGURE 3. Example of trapdoor generation.

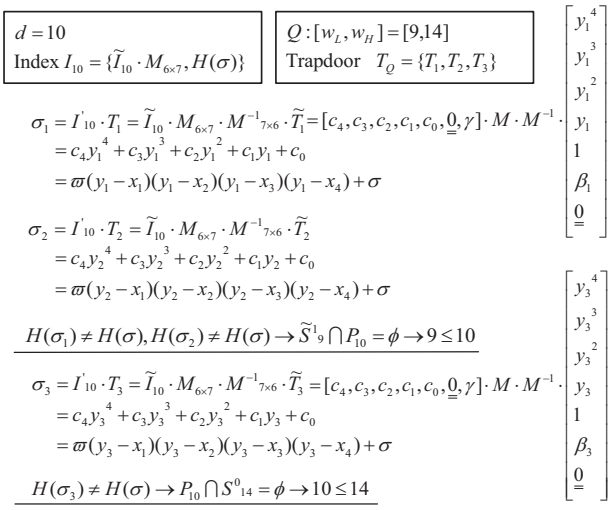

FIGURE 4. Example of comparison between index and trapdoor

2) We suppose that the total number of elements in $S^0_{w_H}$ and $\widetilde{S^1_{w_L}}$ is g. Then, for each element $e_j(1 \leq j \leq g)$ in $S^0_{w_H}$ and $\widetilde{S^1_{w_L}}$, the algorithm computes its hash value by the secret key K to obtain $y_j = H(e_j||K) \pmod p$. Note that $y_1, ..., y_g$ are placed in shuffled order, y_j does not correspond to the jth encoding element in $\widetilde{S^1_{w_L}}$ or $S^0_{w_H}$.

3) Since a hash function has the property of one-wayness and collision resistance, the set $H(P_{d_u}||K) = \{x_1, ..., x_\ell\}$ and set $H(\widetilde{S^1_{w_L}}||K) \cup H(S^0_{w_H}||K) = \{y_1, ..., y_g\}$ still satisfy the same properties in Equation (6), shown as follows:

$$\begin{cases} a > b \iff H(\widetilde{S^1_a}||K) \cap H(P_b||K) \neq \emptyset \\ \qquad \text{or } H(P_a||K) \cap H(S^0_b||K) \neq \emptyset, \\ a \leq b \iff H(\widetilde{S^1_a}||K) \cap H(P_b||K) = \emptyset \\ \qquad \text{or } H(P_a||K) \cap H(S^0_b||K) = \emptyset. \end{cases} \quad (12)$$

The union of intersection

$$[H(\widetilde{S^1_{w_L}}||K) \cap H(P_{d_u}||K)] \cup [H(P_{d_u}||K) \cap H(S^0_{w_H}||K)]$$

is the intersection $\{x_1, ..., x_\ell\} \cap \{y_1, ..., y_g\}$.

4) To distinguish the trapdoors of different queries with the same lower and upper bound values, the algorithm generates a random nonce β_j for each y_j and constructs a column vector $\tilde{T}_j$ with $\ell + 2$ elements, as follows

$$\tilde{T}_j = [y_j^\ell, y_j^{\ell-1}, ..., y_j, 1, \beta_j]^\top \pmod p. \quad (13)$$

Then, $\tilde{T}_j$ is left multiplied by matrix $[M^{-1}]_{(\ell+3)\times(\ell+2)}$ to obtain the jth trapdoor element, as follows

$$\begin{aligned} T_j &= [M^{-1}]_{(\ell+3)\times(\ell+2)} \cdot \tilde{T}_j \\ &= [M^{-1}]_{(\ell+3)\times(\ell+3)} \cdot [y_j^\ell, ..., y_j, 1, \beta_j, 0]^\top. \end{aligned} \quad (14)$$

Finally, the data owner returns g vectors as the trapdoor of query Q to the data user

$$T_Q = \{T_1, ..., T_g\}. \quad (15)$$

• **Rag_Search**$(\mathcal{I}, T_Q)$: The Rag_Search algorithm is conducted by the cloud server to determine records that satisfy the query Q. Figure 4 illustrates an example of search algorithm between index $\mathcal{I}_{d_u}$ and trapdoor T_Q. For each record $r_u \in \mathcal{R}$, the cloud server executes the following steps to determine whether it satisfies the query or not.

1) Based on the received index $I_{d_u} = \{I'_{d_u}, H(\sigma_u)\}$ and trapdoor T_Q, the cloud server calculates

$$\begin{aligned} \sigma_{uj} &= I'_{d_u} \cdot T_j \\ &= \tilde{I}_{d_u} \cdot [M]_{(\ell+2)\times(\ell+3)} \cdot [M^{-1}]_{(\ell+3)\times(\ell+2)} \cdot \tilde{T}_j \\ &= [c_\ell, ..., c_0, 0, \gamma_u] \cdot M \cdot M^{-1} \cdot [y_j^\ell, ..., y_j, 1, \beta_j, 0]^\top \\ &= c_\ell y_j^\ell + c_{\ell-1} y_j^{\ell-1} + ... + c_0 + 0 \cdot \beta_j + \gamma_u \cdot 0 \\ &= \varpi_u (y_j - x_1)(y_j - x_2)...(y_j - x_\ell) + \sigma_u \pmod p. \end{aligned}$$

The idea of this step is to plug each trapdoor element y_j into function $F^P_{d_u}(x)$. Once there is a trapdoor element T_j to obtain $H(\sigma_{uj}) = H(\sigma_u)$, the server learns $\{x_1, ..., x_\ell\} \cap \{y_j\} \neq \emptyset$, which means either

$$\begin{aligned} d_u < w_L \longleftarrow H(\widetilde{S^1_{w_L}}||K) \cap H(P_{d_u}||K) \neq \emptyset \\ \text{or } w_H < d_u \longleftarrow H(P_{d_u}||K) \cap H(S^0_{w_H}||K) \neq \emptyset \end{aligned}$$

The algorithm outputs $d_u \notin [w_L, w_H]$ that leads to the unsatisfied record d_u. Then, the algorithm directly moves to determine the next record.

2) When all of $H(\sigma_{u1}) \neq H(\sigma_u), ..., H(\sigma_{ug}) \neq H(\sigma_u)$, it indicates that

$$w_L \leq d_u \longleftarrow H(\widetilde{S^1_{w_L}}||K) \cap H(P_{d_u}||K) = \emptyset$$
$$d_u \leq w_H \longleftarrow H(P_{d_u}||K) \cap H(S^0_{w_H}||K) = \emptyset.$$

The algorithm outputs $d_u \in [w_L, w_H]$ and inserts r_u into $\mathcal{D}(Q)$ as the query result.

After scanning all of the records in $\mathcal{R}$, the cloud server returns query result $\mathcal{D}(Q)$ to the data user.

5.3. Index Generation Optimization

The main time cost of Bld_Index and Rag_Search algorithm is in the calculation of polynomial function $F^P_{d_u}(x)$. To improve their execution speeds, we reduce the comparing units into smaller groups by following the grouping method. Instead of constructing a single polynomial function $F^P_{d_u}(x)$ based on the hash value of the entire elements $\{x_1, ..., x_\ell\}$, we first shuffle the prefix elements, then evenly partition them into several groups. Then, we construct several polynomial functions based on the x_i in each group, such that each polynomial function degree is smaller. During the search phase, trapdoor elements are checked with each group of x_i to find the first unsatisfied group. That is, each trapdoor element y_j is plugged into each group of polynomial function. The purpose of this grouping is to reduce the degree of $F^P_{d_u}(x)$ and the number of elements in $I^{'}_{d_u}$ and T_j. It accelerates the speed of Bld_Index (index building) and Rag_Search (trapdoor comparison).

The following shows an example of the grouping method. The prefix strings of an 8-bit attribute value d are partitioned into 2 groups in random order $\{x_1, x_8, x_2, x_4 \mid x_5, x_7, x_3, x_6\}$. It constructs a polynomial function for each group with the same random nonce σ. Finally, the index of d is the cascading of coefficients $\{[I^{'}_d]_1 \mid\mid [I^{'}_d]_2, H(\sigma)\}$ in each polynomial function. Since the elements are grouped in shuffled order, the comparison in grouping method provides the same security guarantee as for the original scheme. That is, the server is unable to learn which binary bit differs between the attribute value and the query.

$$\varpi_1(x - x_1)(x - x_8)(x - x_2)(x - x_4) + \sigma$$
$$[I^{'}_d]_1 = [c_{4,1}, ..., c_{0,1}, \gamma_1] \cdot M_{6\times 7},$$
$$\varpi_2(x - x_5)(x - x_7)(x - x_3)(x - x_6) + \sigma$$
$$[I^{'}_d]_2 = [c_{4,2}, ..., c_{0,2}, \gamma_2] \cdot M_{6\times 7}.$$

6. IMPLEMENTATION AND EVALUATION

Apart from the security improvements, in this section we implement our scheme and evaluate its practicality with various parameter settings. Specifically, we assess the performance of our scheme against the OPE scheme [12] and ORE [16] scheme to illustrate the benefits and costs of our scheme's security enhancements. As shown in Table 1, comparable encryption (CE) is a variant of the OPE and ORE schemes. However, it leaks the numerical difference of indexes during the search by default, which is less secure than the small-domain ORE. Therefore, in this section we do not compare the search efficiency of our scheme with comparable encryption. The TDAG [18] scheme and PBtree [17] scheme rely on special tree structures built on the entire dataset to enhance query efficiency. Their searching speeds are highly related to the query choice and distribution of attribute domain. Both the TDAG and PBtree schemes produce false positives to the range query results. However, our scheme only addresses how to build secure indexes for attribute values and trapdoor for the range query condition. There is no false positive in the query results of our scheme. Hence, we do not assess the efficiency of our scheme in comparison with that of the TDAG and PBtree schemes. Certainly, all of the tree structures proposed in the TDAG and PBtree schemes for the entire dataset can be directly applied to our scheme, which can achieve the same efficiency with better security.

6.1. Experimental Settings

6.1.1. Implementation

Our scheme was implemented in C. For the cryptographic details, we chose the security parameter λ as 128 bits. For the sake of fairness, we used the PRF function implemented in the ORE scheme [16, 30] as the hash function H in our scheme, which is an AES-128 construction. Unlike the PRF function, we took the output of our AES-based hash function to be the domain of $\{0,1\}^{128}$. We then converted the outputs of H as *mpz_t* integers and used GMP-5.0.1 library [31] for all of the arithmetic operations. We chose the maximum value of 128 bits as the modulus $p = 2^{128} - 1$ and random invertible matrix M was generated with integer entries. All of the following experiments were conducted on a computer running macOS Sierra 10.12.5 with 4GB memory and a 1.3-GHz Intel Core i5 CPU. For the evaluation of the ORE scheme [16], we directly used the C implementation of FastORE [30]. For the evaluation of the OPE scheme [12], we used the C++ implementation from CryptDB [32, 33].

Range query is widely used in different scenarios, in which the query attribute is one of the important factors affecting performance. Specifically, date and time are often used range query attributes (i.e., search for the records during the time period from '01/07/2017 00:00:00' to '01/08/2017 00:00:00'). Since date and time are usually displayed in long format, the schemes use integers with longer bit length to represent this type of attribute value. For instance, Li et al. converted the check-in time attribute field in the Gowalla dataset (a geo-social network dataset [34]) to 32-bit integers [17]. Another type of range query attribute has a relatively

TABLE 2. Parameter settings for figures

Figure	Attribute Value (bits)	Group Size (bits)
5,6	16, 24, 32, 48 ,64	2, 8, 16
7,9,11		12, 16
8,10	64	4, 8, 12, 14, 16
12	16	

small domain (i.e., product price, employee salary, and student scores), which can be represented as integers with shorter bit length. For example, the annual salary field in the USPS dataset (a dataset of employee records of the US Postal Service [35]) can be represented as 24-bit integers.

To assess the performance of our scheme under different scenarios, we randomly choose the attribute value and the upper or lower bound of a query as the integers with different bit lengths. Our scheme used the grouping method introduced in Section 5.3. The group size is the number of elements involved in building each group of indexes. To discuss the performance of schemes under group size, we also varied the group size within the maximum bit length of attribute. Each measurement was found by taking the mean value over 50-10^7 iterations. Table 2 lists the parameter settings of each experimental result figure.

6.2. Experimental Result

6.2.1. ***Evaluation with different parameter settings***

The index size and building time of our scheme for different-sized attribute values and groups are illustrated in Figure 5 and Figure 6, respectively. The bars in both figures have the same trends. That is, the index building time and size increase with the increased bit length of attribute value. For the same bit length of attribute value, the index building time increases with increased group size, as shown in Figure 5. Whereas for the same bit length of the attribute value, the index size decreases with increased group size, shown in Figure 6. The opposite results are due to the larger group size, resulting in a longer time spent on constructing the index, but with a smaller number of groups. Each group brings one more set of polynomial coefficients to the entire index. The index size becomes smaller when there is a smaller number of groups. Hence, in our scheme there exists a trade-off between index building time and index size when choosing group size.

6.2.2. ***Evaluation of the index building time***

The index building time comparison between OPE [12], ORE [16] schemes and our scheme for different-sized attribute values and groups is shown in Figure 7 and Figure 8, respectively. The random oracle in ORE and our scheme use the same AES-based construction. The

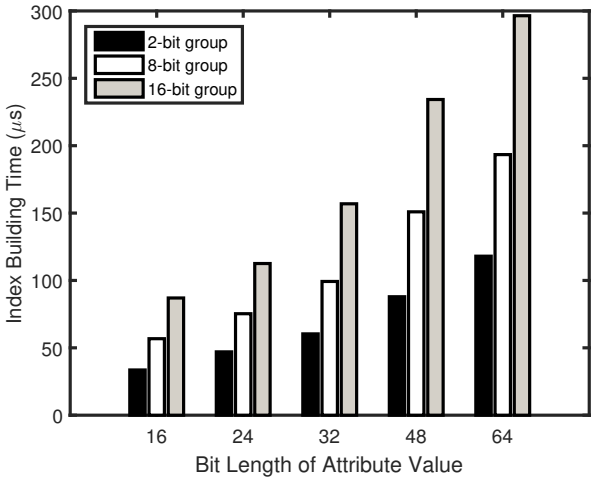

FIGURE 5. Index building time of our scheme.

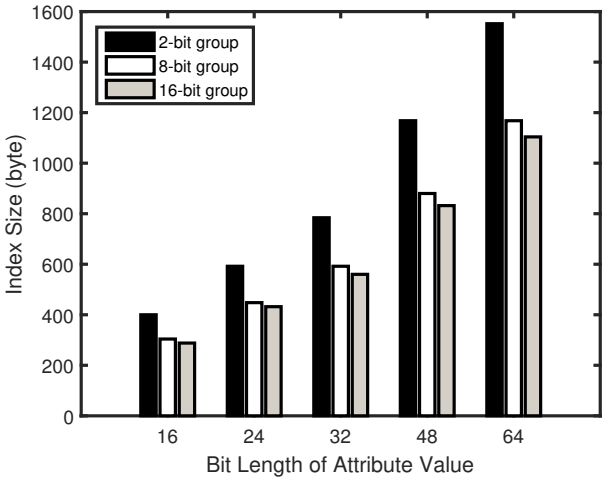

FIGURE 6. Index size of our scheme.

group size for the ORE scheme is the number of bits in each block. Each group or block is also the comparing unit for ORE and our scheme. As shown in Figure 7, among the three schemes, our scheme has the minimum index building time, while the ORE scheme with the 16-bit group has the maximum index building time. The time cost of the OPE scheme is slower than the ORE scheme with 12-bit, but still much faster than the ORE scheme with 16-bit groups. Specifically, the index building time of our scheme is on average over 16 times faster compared to the OPE scheme, and on average over 6 times faster compared to the ORE scheme with 12-bit groups. We continue to discuss the influence of group size on both the ORE scheme and our scheme. The index building time of our scheme and the ORE scheme under different group sizes is shown in Figure 8. Note that the index building time of the ORE scheme grows more rapidly than our scheme when the group size is larger than 8 bits.

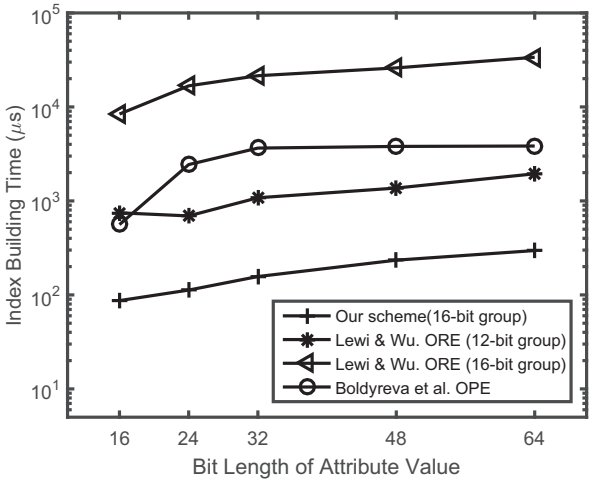

FIGURE 7. Index building time of our scheme, OPE and ORE schemes with different-sized attribute values.

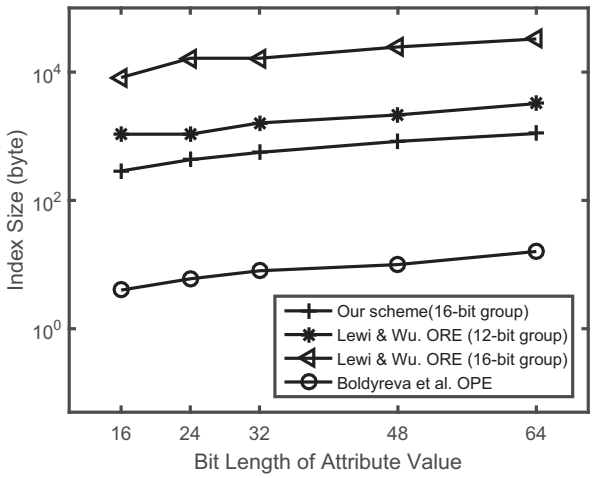

FIGURE 9. Index size of our scheme, OPE and ORE schemes with different-sized attribute values.

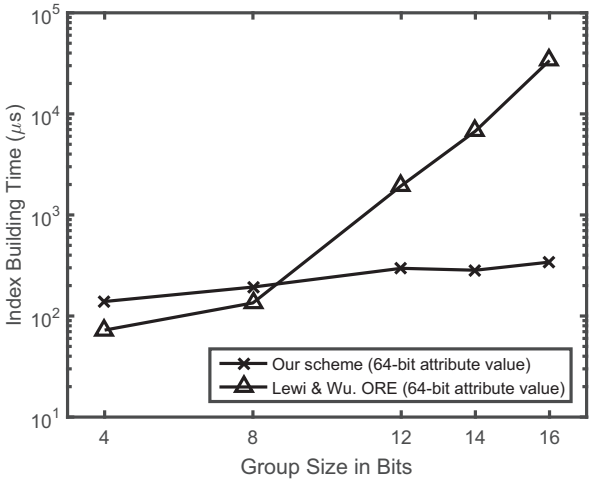

FIGURE 8. Influence of group size on the index building time of our scheme and ORE scheme.

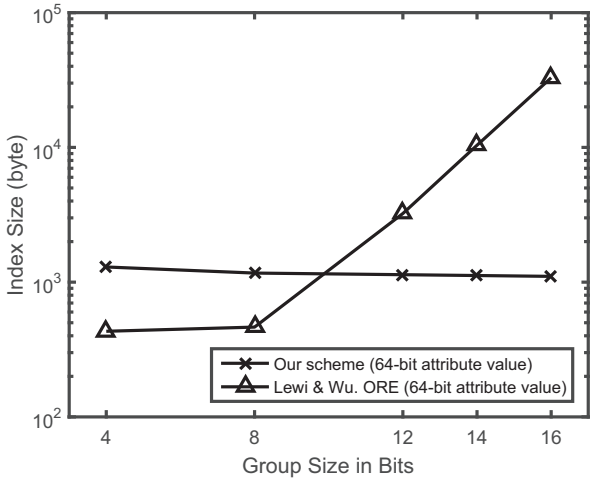

FIGURE 10. Influence of group size on the index size of our scheme and ORE scheme.

6.2.3. ***Evaluation of the index size***

The index size comparison between OPE [12], ORE [16] schemes and our scheme for different-sized attribute values and groups is shown in Figure 9 and Figure 10, respectively. In Figure 9, the index of the OPE scheme is the smallest due to its ciphertext still being a numerical value. The ORE scheme with the 16-bit group also has the largest index size. The index size of our scheme with the 16-bit group is on average 2.92 times smaller, compared to the ORE scheme with the 12-bit group. Figure 10 illustrates the index size of ORE and our scheme under a different group size. The index size of the ORE scheme also grows quickly when the group size is larger than 8 bits. However, the index size of our scheme decreases with increased group size.

As a similar trend reflected from Figure 8 and Figure 10, the index generation of the ORE scheme is only efficient by setting a relatively small block size. The main reason is that the ORE scheme relies on a small-domain ORE construction in each group/block to achieve the best possible security. That is, the indexes in each ORE group leak nothing but the ordering of their plaintexts. The cost of constructing the small-domain ORE indexes grows linearly in the size of group message space. For a b-bit group, the index size of each ORE group is linear with $2^b - 1$ [16]. Whereas in our scheme, the index of each group is linear with the bit length of the group b. Therefore, when the group size is large, the index generation time and index size of our scheme are much faster and smaller than those of the ORE scheme.

6.2.4. ***Evaluation of the search time***

Our scheme leaks strictly less information than the OPE [12] and ORE [16] schemes. In this experiment, we evaluate how much search efficiency has been sacrificed

due to the security improvements in our scheme. Figure 11 compares the search times of the OPE and ORE schemes and ours with different-sized attribute values. The search time is the time used to compare an index with a trapdoor value of a range query's upper or lower bound. The three lower lines (i.e., including marker symbols $+$, $*$, and $\circ$) in Figure 11 indicate that our scheme is slower than OPE and ORE (12-bit group) schemes. This is because our scheme is designed to hide more sensitive information than the other two schemes. Hence, it requires a longer time to complete a range comparison. The search time of the OPE scheme is the fastest, since it directly compares two numerical values. Nevertheless, it leaks much more information than the ORE scheme, and indeed, much more information than our scheme does, as shown in Table 1. Hence, OPE-encrypted indexes are vulnerable to inference attack, which directly leaks the frequency and order relationship of attribute values.

To hide the bit of an attribute value that differs its index from the trapdoor, our scheme builds the index from a single function $F_{d_u}^P(x)$ using the entire prefix strings of the attribute value d_u. In the grouping method proposed in Section 5.3, we shuffle the prefix strings of each attribute value before partitioning them into different groups. Thus, the group of index elements that stops algorithm Rag_Search does not correspond to the same group of bits in the plaintext of d_u. As a result, our scheme does not leak the relative differences between attribute values to the same query in the range comparison process. Meanwhile, the server in our scheme has to compare the trapdoor with all or part of the attribute value's prefix strings, which takes a longer time. In the ORE scheme, its index elements keep the same binary order of the plaintext of the attribute value. Once the index differentiates the trapdoor in a group/block of high bits, the ORE scheme will stop the comparison. Consequently, the ORE scheme is faster than our scheme when the group size is small (e.g., no more than 12 bits). However, the ORE scheme tells the server about the first bit or group of bits that differs between an index and a trapdoor. This leakage shows the relative distances between different attribute values to the same query. Even for the best secure setting of the ORE scheme (i.e., small-domain ORE), our scheme still provides stronger security. The ORE comparison result inevitably discloses whether an unmatched attribute value is larger than the upper bound or smaller than the lower bound of a range query, while the trapdoor generated in our scheme is not deterministic, which can prevent the search pattern leakage of the ORE scheme.

From the two upper lines (i.e., including marker symbols $\lhd$ and $+$) in Figure 11, interestingly, we can observe an opposite trend. The search time of our scheme is 3.89 times faster than the ORE scheme when the group size is 16 bits. To further explain the reason for this result, we discuss the influence of group size

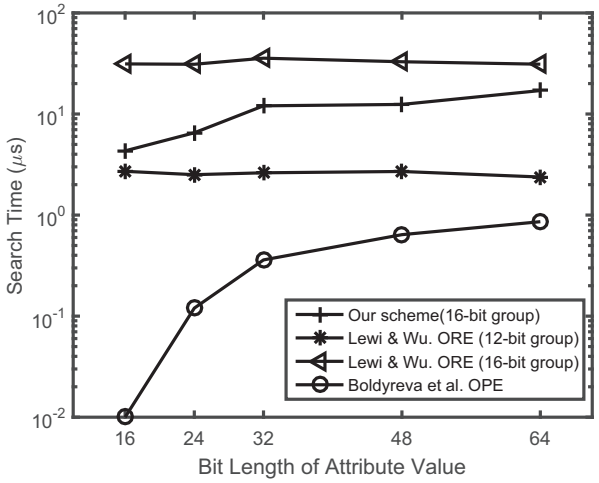

FIGURE 11. Search time of our scheme, OPE and ORE schemes with different-sized attribute values.

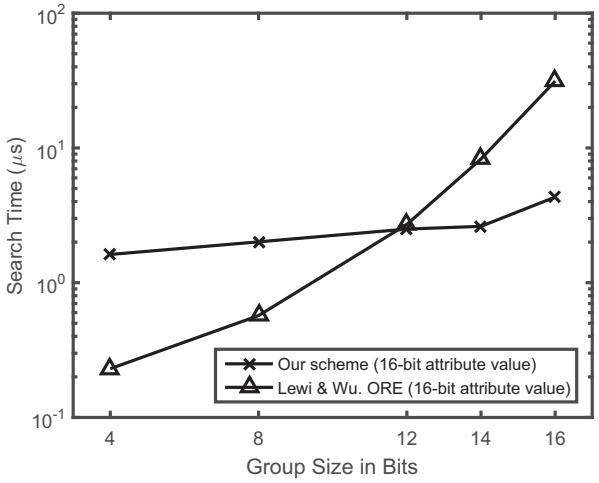

FIGURE 12. Influence of group size on the search time of our scheme and ORE scheme.

on both the ORE and our scheme in Figure 12. Since the OPE scheme does not process the range comparison in groups of bits, we cannot test the search time of the OPE scheme in Figure 12. It shows that our scheme is both more secure and faster, compared to the ORE scheme, when the group size is larger than 12 bits. This is because the ORE scheme has adopted the small-domain ORE construction in each group, such that it needs to compare the trapdoor with 2^b index elements in each b-bit group. When the group size increases, the ORE scheme provides higher security, but at the expense of ORE's search efficiency, which declines greatly. In our scheme, on the contrary, each b-bit group has exactly b index elements. The increasing group size has less impact on the search speed of our scheme. Additionally, our scheme provides the same security guarantees under different group sizes, as discussed before. Therefore, we can conclude that under the ORE

scheme, it is difficult to achieve both security and search efficiency simultaneously. To achieve higher security, the ORE scheme must sacrifice more efficiency than our scheme.

7. SECURITY ANALYSIS

Motivated by the approach in [17], we analyse the security of our scheme in this section to prove that it achieves the defined security goals.

7.1. Ciphertext Indistinguishability Proof

Theorem 7.1. *The privacy-preserving range query scheme* $\mathsf{SSE}^{\mathsf{RAG}}$ *is ciphertext indistinguishability secure with the leakage function* $\mathcal{L}$ *from Definition 3.2, assuming that the keyed hash function* H *is a secure pseudo-random function.*

Proof: We use contradiction to prove the Theorem 7.1. Supposing that our scheme $\mathsf{SSE}^{\mathsf{RAG}}$ is not ciphertext indistinguishability secure, then there exists a polynomial-sized adversary $\mathcal{A}_1$ that can win Game 1 in Section 3.4.1 with an advantage greater than $\mathsf{negl}(\lambda)$. We construct a polynomial-sized adversary $\mathcal{B}_1$, which uses $\mathcal{A}_1$ as a subroutine to break the pseudo-randomness of function H.

Specifically, adversary $\mathcal{B}_1$ plays with a challenger $\mathcal{C}$ in a pseudo-randomness game. At the same time, $\mathcal{B}_1$ interacts with $\mathcal{A}_1$ by attempting to "fake" the challenger in Game 1. Before answering the queries of $\mathcal{A}_1$, the challenger gives $\mathcal{B}_1$ a function f and algorithm $\mathsf{Bld\_Index}$ and $\mathsf{Gen\_Trapdoor}$. However, the keyed hash functions H in $\mathsf{Bld\_Index}$ and $\mathsf{Gen\_Trapdoor}$ are replaced with the function f, which is either pseudo-random or truly random and takes as input $\{0,1\}^{\lambda}$ and outputs $\{0,1\}^{s(\lambda)}$.

Next, we describe how the adversary $\mathcal{B}_1$ provides the view for $\mathcal{A}_1$ and answers $\mathcal{A}_1$'s queries in the following phases.

- **Setup**: Based on the function f, adversary $\mathcal{B}_1$ generates a large collection of attribute values $\mathbb{D}$ and a sequence of range queries $\mathbb{Q}$, constructs a system parameter params and sends them to the adversary $\mathcal{A}_1$.

- **Phase 1**: Adversary $\mathcal{A}_1$ adaptively sends $\mathcal{B}_1$ an attribute value collection $\mathcal{D}_i$ and a range query Q_i as it did in Phase 1 of Game 1. $\mathcal{B}_1$ replies to $\mathcal{A}_1$ with indexes $\mathcal{I}_i$ and trapdoor T_{Q_i} by running algorithm $\mathsf{Bld\_Index}$ and $\mathsf{Gen\_Trapdoor}$ given by the challenger, where $1 \leq i \leq q_1$.

- **Challenge**: Adversary $\mathcal{A}_1$ submits two collections of attribute values $\mathcal{D}_0$ and $\mathcal{D}_1$. $\mathcal{B}_1$ randomly chooses a bit $b \in \{0,1\}$ and replies to $\mathcal{A}_1$ with index $\mathcal{I}_b \leftarrow \mathsf{Bld\_Index}(\mathsf{sk}, \mathcal{D}_b)$ as before.

- **Phase 2**: For request $q_1+1 \leq i \leq q$, adversary $\mathcal{A}_1$ repeats the same process as in Phase 1 and finally obtains $\langle \mathcal{I}_1, ..., \mathcal{I}_q, \mathcal{I}_b \rangle$ and $\langle T_{Q_1}, ..., T_{Q_q} \rangle$.

  All range queries are chosen under the restriction of $\mathcal{L}(\mathcal{D}_0, Q_i) = \mathcal{L}(\mathcal{D}_1, Q_i)$, where $1 \leq i \leq q$.

- **Guess**: After q requests, $\mathcal{A}_1$ outputs a bit b'. If $b' = b$, then $\mathcal{B}_1$ outputs 1 to the challenger in the pseudo-randomness game. This means that $\mathcal{B}_1$ guesses that function f is pseudo-random. If $b' \neq b$, then $\mathcal{B}_1$ outputs 0 to the challenger. This means that $\mathcal{B}_1$ guesses that function f is truly random.

Next, we prove the following two claims to indicate that $\mathcal{B}_1$ can distinguish whether function f is pseudo-random or truly random with non-negligible probability over 1/2.

Claim1. If f is a pseudo-random function, then

$$\Pr[\mathcal{B}_1^f = 1 | f : \{0,1\}^{\lambda} \times \{0,1\}^{\ell} \rightarrow \{0,1\}^{s(\lambda)}] > \frac{1}{2} + \mathsf{negl}(\lambda).$$

Claim2. If f is a truly random function, then

$$\Pr[\mathcal{B}_1^f = 0 | f : \{0,1\}^{\ell} \rightarrow \{0,1\}^{s(\lambda)}] = \frac{1}{2}.$$

Claim 1 Proof: If f is a pseudo-random function, then $\mathcal{A}_1$'s observation is identical to what is viewed during Game 1 defined in Section 3.4.1. Since we have assumed that $\mathcal{A}_1$ can win Game 1 with an advantage greater than $\mathsf{negl}(\lambda)$, then adversary $\mathcal{B}_1$ can also output 1 with an advantage greater than $\mathsf{negl}(\lambda)$. Thus, we prove Claim 1.

7.1.1. Claim 2 Proof

Claim 2 means that adversary $\mathcal{A}_1$ cannot distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$ when the function f is truly random. Next, we prove Claim 2 from the following aspects.

• Indexes of any attribute values $\langle \mathcal{I}_1, ..., \mathcal{I}_q \rangle$ and $\mathcal{I}_b$ reveal no difference between $\mathcal{D}_0$ and $\mathcal{D}_1$ to $\mathcal{A}_1$.

In the index generation, algorithm $\mathsf{Bld\_Index}$ uses the function f to map the prefix string of an attribute value into a string. If f is a truly random function, the output of f is a random string. Then, upon these random strings, algorithm $\mathsf{Bld\_Index}$ assigns different records with different random nonces. Specifically, random nonces ϖ_u and σ_u are embedded in constructing the function $F_{d_u}^P(x)$. And random nonce γ_u is used in the matrix multiplication. The last element of index I_{d_u} is the $f(\sigma_u)$. Hence, each index is identical to a series of random strings. Adversary $\mathcal{A}_1$ is unable to detect that the secret key sk, queried attribute value d_u have not been used. Given two different indexes $I_{d_{u1}}$ and $I_{d_{u2}}$, it is infeasible for $\mathcal{A}_1$ to determine whether they are created from the same attribute value ($d_{u1} = d_{u2}$) or two different attribute values ($d_{u1} \neq d_{u2}$). Additionally, the index sizes of attribute values in $\mathcal{D}_0$ and $\mathcal{D}_1$ are required to be the same. $\mathcal{A}_1$ also cannot distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$ based on the index sizes shown in the $\mathcal{I}_b$.

• **Trapdoors of any range queries $\langle T_{Q_1}, ..., T_{Q_q}\rangle$ reveal no difference between $\mathcal{D}_0$ and $\mathcal{D}_1$ to $\mathcal{A}_1$.**

In algorithm Gen_Trapdoor, each element of trapdoor T_{Q_i} ($Q_i = [w_{iL}, w_{iH}]$) is initially constructed using the new 1-encoding of w_{iL} or 0-encoding of w_{iH}. This encoding approach is different from that used in index generation. Then, algorithm Gen_Trapdoor uses function f to map each encoding into a string. As indicated before, when f is a truly random function, its output is a random string. Later, each trapdoor element T_{Q_i} is assigned with a different random nonce β_j used in the matrix multiplication. Hence, elements of any trapdoor to adversary $\mathcal{A}_1$ are identical to a series of random strings. Therefore, it is infeasible for $\mathcal{A}_1$ to correlate any trapdoor values T_{Q_i} with the attribute values in $\mathcal{D}_0$ and $\mathcal{D}_1$.

• **Matched records in $\mathcal{D}_b(Q_i) \leftarrow \mathsf{Rag\_Search}(\mathcal{I}_b, T_{Q_i})$ reveal no difference between $\mathcal{D}_0$ and $\mathcal{D}_1$ to $\mathcal{A}_1$.**

$\mathcal{D}_0$ and $\mathcal{D}_1$ are required to have the same access pattern under all issued queries, that is, $\mathcal{D}_0(Q_i) = \mathcal{D}_1(Q_i)$. And every prefix string of a satisfied attribute value has no common elements with any encodings of a range query, that is $H(\sigma_{u1}) \neq H(\sigma_u), ..., H(\sigma_{ug}) \neq H(\sigma_u)$. Hence, $\mathcal{A}_1$ cannot trivially distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$ based on their matched records under any range queries.

• **The case of how each unmatched record in $\mathcal{D}_b$ fails to be returned by a range query reveals no difference between $\mathcal{D}_0$ and $\mathcal{D}_1$ to $\mathcal{A}_1$.**

Based on the property shown in Equation (7), there is no difference when comparing the index with the trapdoor elements of upper bound or lower bound of the same range query. That is, $\mathcal{A}_1$ cannot distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$ based on the difference of $d_u > w_{iH}$ or $d_u < w_{iL}$. In addition, function $F^P_{d_u}(x)$ is constructed from all hashed prefix strings of d_u. The $\mathcal{A}_1$ cannot detect in which bit of d_u that differs d_u from Q_i. In the grouping method proposed in Section 5.3, we shuffle the prefix elements of each attribute value before partitioning them into different groups. Thus, the group of index elements that stops algorithm Rag_Search does not correspond to the same group of bits of d_u and leaks no difference between $\mathcal{D}_0$ and $\mathcal{D}_1$.

If f is a truly random function, $\mathcal{B}_1$ can correctly guess f with a probability of 1/2. Thus, we prove Claim 2.

The function f given by the challenger is either pseudo-random or truly random with the same probability of 1/2. Combining Claim 1 with Claim 2, we obtain

$$\frac{1}{2}\Pr[\mathcal{B}_1^f = 1 | f : \{0,1\}^\lambda \times \{0,1\}^\ell \rightarrow \{0,1\}^{s(\lambda)}] + \frac{1}{2}\Pr[\mathcal{B}_1^f = 0 | f : \{0,1\}^\ell \rightarrow \{0,1\}^{s(\lambda)}] > \frac{1}{2} + \frac{\mathsf{negl}(\lambda)}{2}.$$

This result indicates that $\mathcal{B}_1$ can distinguish a pseudo-random function from a truly random function with an advantage greater than $\mathsf{negl}(\lambda)$. However, this result contradicts the property of a pseudo-random function, which is impossible. Thus, we prove that an adversary $\mathcal{A}_1$ that can win in Game 1 with non-negligible probability over 1/2 does not exist. Therefore, our scheme $\mathsf{SSE}^{\mathsf{RAG}}$ is ciphertext indistinguishability secure.

7.2. Trapdoor Indistinguishability Proof

THEOREM 7.2. *The privacy-preserving range query scheme $\mathsf{SSE}^{\mathsf{RAG}}$ is trapdoor indistinguishability secure with the leakage function $\mathcal{L}$ from Definition 3.2, assuming that the keyed hash function H is a secure pseudo-random function.*

Proof: We use contradiction to prove the Theorem 7.2. Supposing that our scheme $\mathsf{SSE}^{\mathsf{RAG}}$ is not trapdoor indistinguishability secure, then there exists a polynomial-sized adversary $\mathcal{A}_2$ that wins in Game 2, defined in Section 3.4.2 with non-negligible probability over 1/2. We construct a polynomial-sized adversary $\mathcal{B}_2$, which uses $\mathcal{A}_2$ as a subroutine to break the pseudo-randomness of function H.

Specifically, adversary $\mathcal{B}_2$ plays with a challenger $\mathcal{C}$ in a pseudo-randomness game. At the same time, $\mathcal{B}_2$ interacts with $\mathcal{A}_2$ by attempting to "fake" the challenger in Game 2. Before answering the queries of $\mathcal{A}_2$, the challenger gives $\mathcal{B}_2$ a function z and algorithm Bld_Index and Gen_Trapdoor. However, the keyed hash functions H in Bld_Index and Gen_Trapdoor are replaced with the function z, which is either pseudo-random or truly random, and takes as input $\{0,1\}^\lambda$ and outputs $\{0,1\}^{s(\lambda)}$.

Next, we describe how the adversary $\mathcal{B}_2$ provides the view for $\mathcal{A}_2$ and answers $\mathcal{A}_2$'s queries in the following phases.

- **Setup**: Based on the function z, adversary $\mathcal{B}_2$ generates a large collection of attribute values $\mathbb{D}$ and a sequence of range queries $\mathbb{Q}$, constructs a system parameter params and sends them to the adversary $\mathcal{A}_2$.
- **Phase 1**: Adversary $\mathcal{A}_2$ adaptively sends $\mathcal{B}_2$ an attribute value collection $\mathcal{D}_i$ and a range query Q_i as it did in Phase 1 of Game 2. $\mathcal{B}_2$ replies to $\mathcal{A}_2$ with indexes $\mathcal{I}_i$ and trapdoor T_{Q_i} by running algorithm Bld_Index and Gen_Trapdoor given by the challenger, where $1 \leq i \leq q_1$.
- **Challenge**: Adversary $\mathcal{A}_2$ picks two range queries Q_0 and Q_1. $\mathcal{B}_2$ randomly samples a bit $c \in \{0,1\}$ and replies to $\mathcal{A}_2$ with a trapdoor $T_{Q_c} \leftarrow \mathsf{Gen\_Trapdoor}(\mathsf{sk}, Q_c)$ as before.
- **Phase 2**: For request $q_1 + 1 \leq i \leq q$, adversary $\mathcal{A}_2$ repeats the same process as in Phase 1 and finally obtains $\langle \mathcal{I}_1, ..., \mathcal{I}_q\rangle$ and $\langle T_{Q_1}, ..., T_{Q_q}, T_{Q_c}\rangle$.

  All collections of attribute values are chosen under the restriction of $\mathcal{L}(\mathcal{D}_i, Q_0) = \mathcal{L}(\mathcal{D}_i, Q_1)$, where

$1 \leq i \leq q$.

- **Guess**: After q requests, $\mathcal{A}_2$ outputs a bit $c^{'}$. If $c^{'} = c$, then $\mathcal{B}_2$ outputs 1 to the challenger in the pseudo-randomness game. This means that $\mathcal{B}_2$ guesses that z is a pseudo-random function. If $c^{'} \neq c$, then $\mathcal{B}_2$ outputs 0 to the challenger. This means that $\mathcal{B}_2$ guesses that z is a truly random function.

Next, we prove the following two claims to indicate that $\mathcal{B}_2$ can distinguish whether function z is pseudo-random or truly random with an advantage greater than $\mathsf{negl}(\lambda)$.

Claim3. If z is a pseudo-random function, then

$$\Pr[\mathcal{B}_2^z = 1 | z : \{0,1\}^\lambda \times \{0,1\}^\ell \rightarrow \{0,1\}^{s(\lambda)}] > \frac{1}{2} + \mathsf{negl}(\lambda).$$

Claim4. If z is a truly random function, then

$$\Pr[\mathcal{B}_2^z = 0 | z : \{0,1\}^\ell \rightarrow \{0,1\}^{s(\lambda)}] = \frac{1}{2}.$$

Claim 3 Proof: If z is a pseudo-random function, then $\mathcal{A}_2$'s observation is identical to what is viewed during Game 2 defined in Section 3.4.2. Since we have assumed that $\mathcal{A}_2$ can win Game 2 with non-negligible probability over 1/2, then adversary $\mathcal{B}_2$ can also output 1 with non-negligible probability over 1/2. Thus, we prove Claim 3.

7.2.1. Claim 4 Proof

Claim 4 means that adversary $\mathcal{A}_2$ cannot distinguish Q_0 from Q_1 when the function z is truly random. Next, we prove Claim 4 from the following aspects.

• Trapdoors of any range queries $\langle T_{Q_1}, ..., T_{Q_q} \rangle$ and T_{Q_c} reveal no difference between Q_0 and Q_1 to $\mathcal{A}_2$.

As indicated in the second point of Section 7.1.1, algorithm Gen_Trapdoor uses function z and different random nonce β_j in generating each trapdoor element T_{Q_i}. When the function z is truly random, the elements of any trapdoor are identical to a series of random strings. Adversary $\mathcal{A}_2$ is unable to detect that the secret key sk, the upper bound w_{iH} and lower bound w_{iL} have not been used. Given two different trapdoors $T_{Q_{i1}}$ and $T_{Q_{i2}}$, it is infeasible for $\mathcal{A}_2$ to determine whether they are created from the same query range or not, and whether the upper or lower bound of Q_{i1} is larger or smaller than that of Q_{i2}. Additionally, the trapdoor sizes of Q_0 and Q_1 are required to be the same. $\mathcal{A}_2$ cannot distinguish Q_0 from Q_1 based on the trapdoor size of T_{Q_c}.

• Indexes of any attribute values $\langle \mathcal{I}_1, ..., \mathcal{I}_q \rangle$ reveal no difference between Q_0 and Q_1 to $\mathcal{A}_2$

As mentioned in the first two points of Section 7.1.1, both algorithm Bld_Index and Gen_Trapdoor add different random nonces when generating each index and trapdoor element. Adversary $\mathcal{A}_2$ is unable to correlate any index value I_{d_u} with the trapdoor of Q_0 or Q_1.

• Matched records in $\mathcal{D}_i(Q_c) \leftarrow \mathsf{Rag\_Search}(\mathcal{I}_i, T_{Q_c})$ reveal no difference between Q_0 and Q_1 to $\mathcal{A}_2$.

Q_0 and Q_1 are required to have the same access pattern with all issued collections of attribute values, that is, $\mathcal{D}_i(Q_0) = \mathcal{D}_i(Q_1)$. And every trapdoor element has no common elements with any prefix string of a satisfied attribute value, that is $H(\sigma_{u1}) \neq H(\sigma_u), ..., H(\sigma_{ug}) \neq H(\sigma_u)$. Hence, $\mathcal{A}_2$ cannot trivially distinguish Q_0 from Q_1 based on their matched records in any attribute value collections.

• The case of how each unmatched record fails to be returned by the range query Q_c reveals no difference between Q_0 and Q_1 to $\mathcal{A}_2$.

To hide which trapdoor element differs between the trapdoor and index, algorithm Gen_Trapdoor places the trapdoor elements of each issued range query in shuffled order. That is, trapdoor element T_j causing the $H(\sigma_{uj}) = H(\sigma_u)$ does not correspond to the same bit of lower/upper bound of a range query. Hence, the trapdoor element that differs in each unmatched attribute value from query Q_c does not leak any difference between Q_0 and Q_1 to $\mathcal{A}_2$.

If z is a truly random function, $\mathcal{B}_2$ can correctly guess z with a probability of 1/2. Thus, we prove Claim 4.

The function z given by the challenger is either pseudo-random or truly random, with the same probability of 1/2. Combining Claim 3 with Claim 4, we obtain

$$\frac{1}{2}\Pr[\mathcal{B}_2^z = 1 | f : \{0,1\}^\lambda \times \{0,1\}^\ell \rightarrow \{0,1\}^{s(\lambda)}] + \frac{1}{2}\Pr[\mathcal{B}_2^z = 0 | f : \{0,1\}^\ell \rightarrow \{0,1\}^{s(\lambda)}] > \frac{1}{2} + \frac{\mathsf{negl}(\lambda)}{2}.$$

This result indicates that $\mathcal{B}_2$ can distinguish a pseudo-random function from a truly random function with non-negligible probability over 1/2. However, this result contradicts the property of a pseudo-random function, which is impossible. Thus, we prove that there does not exist an adversary $\mathcal{A}_2$ that can win in Game 2 with an advantage greater than $\mathsf{negl}(\lambda)$. Thus, our scheme $\mathsf{SSE}^{\mathsf{RAG}}$ is trapdoor indistinguishability secure.

To sum up, based on the Definition in 3.5, we can conclude that our privacy-preserving range query scheme $\mathsf{SSE}^{\mathsf{RAG}}$ is both ciphertext and trapdoor indistinguishability secure with the leakage function $\mathcal{L}$ from Definition 3.2, assuming that the keyed hash function H is a secure pseudo-random function.

8. CONCLUSION

In this paper, we have designed an order-hiding range query scheme. Our scheme solves the security leakage

problem in existing secure range query schemes. To hide the statistical relationships among indexes, our scheme adopts the 0/1 encoding technique and constructs the indexes as the coefficients of the randomized polynomial function. To avoid leaking the comparison operator and search pattern, our scheme introduces a random invertible matrix in the generation of query trapdoors. We formally analyse sensitive information leakage in our scheme, and have proved it is secure under an IND-CKA2 security definition without restriction of the same search pattern. We implemented and assessed the performance of our scheme. The comparison results show that although the ORE scheme has a shorter index size and search time with small processing units, it is slower and has a longer index size than our scheme when the processing unit is large. On average, the index building time of our scheme is more than 16 times faster than the OPE scheme. Meanwhile, our scheme only leaks the access pattern, and is proved to be more secure than existing schemes.

ACKNOWLEDGMENT

This work is supported by the Department of Computing, The Hong Kong Polytechnic University.

REFERENCES

[1] Bösch, C., Hartel, P. H., Jonker, W., and Peter, A. (2014) A survey of provably secure searchable encryption. *ACM Comput. Surv.*, **47**, 18:1–18:51.

[2] Wang, B., Yu, S., Lou, W., and Hou, Y. T. (2014) Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*, pp. 2112–2120. IEEE.

[3] Wang, B., Song, W., Lou, W., and Hou, Y. T. (2015) Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee. *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, pp. 2092–2100. IEEE.

[4] Sun, W., Yu, S., Lou, W., Hou, Y. T., and Li, H. (2014) Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*, pp. 226–234. IEEE.

[5] Cao, N., Wang, C., Li, M., Ren, K., and Lou, W. (2014) Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.*, **25**, 222–233.

[6] Wang, Q., He, M., Du, M., Chow, S. S. M., Lai, R. W. F., and Zou, Q. (2018) Searchable encryption over feature-rich data. *IEEE Trans. Dependable and Secure Computing*, **15**, 496–510.

[7] Du, M., Wang, Q., He, M., and Weng, J. (2018) Privacy-preserving indexing and query processing for secure dynamic cloud storage. *IEEE Trans. Information Forensics and Security*, **13**, 2320–2332.

[8] Wang, Q., Ren, K., Du, M., Li, Q., and Mohaisen, A. (2017) Secgdb: Graph encryption for exact shortest distance queries with efficient updates. *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, Cham, pp. 79–97. Springer.

[9] Shen, M., Ma, B., Zhu, L., Mijumbi, R., Du, X., and Hu, J. (2018) Cloud-based approximate constrained shortest distance queries over encrypted graphs with privacy protection. *IEEE Trans. Information Forensics and Security*, **13**, 940–953.

[10] Boneh, D. and Waters, B. (2007) Conjunctive, subset, and range queries on encrypted data. *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, Berlin, Heidelberg, pp. 535–554. Springer.

[11] Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. (2004) Order-preserving encryption for numeric data. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, New York, NY, USA, pp. 563–574. ACM.

[12] Boldyreva, A., Chenette, N., Lee, Y., and O'Neill, A. (2009) Order-preserving symmetric encryption. *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, Berlin, Heidelberg, pp. 224–241. Springer.

[13] Popa, R. A., Li, F. H., and Zeldovich, N. (2013) An ideal-security protocol for order-preserving encoding. *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pp. 463–477. IEEE.

[14] Kerschbaum, F. and Schröpfer, A. (2014) Optimal average-complexity ideal-security order-preserving encryption. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, New York, NY, USA, pp. 275–286. ACM.

[15] Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., and Zimmerman, J. (2015) Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, Berlin, Heidelberg, pp. 563–594. Springer.

[16] Lewi, K. and Wu, D. J. (2016) Order-revealing encryption: New constructions, applications, and lower bounds. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, New York, NY, USA, pp. 1167–1178. ACM.

[17] Li, R., Liu, A. X., Wang, A. L., and Bruhadeshwar, B. (2016) Fast and scalable range query processing with strong privacy protection for cloud computing. *IEEE/ACM Trans. Netw.*, **24**, 2305–2318.

[18] Demertzis, I., Papadopoulos, S., Papapetrou, O., Deligiannakis, A., and Garofalakis, M. N. (2016)

Practical private range search revisited. *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, New York, NY, USA, pp. 185–198. ACM.

[19] Gai, K. and Qiu, M. (2017) Blend arithmetic operations on tensor-based fully homomorphic encryption over real numbers. *IEEE Trans. Industrial Informatics*, **PP**, 1–1.

[20] Naveed, M., Kamara, S., and Wright, C. V. (2015) Inference attacks on property-preserving encrypted databases. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, New York, NY, USA, pp. 644–655. ACM.

[21] Furukawa, J. (2013) Request-based comparable encryption. *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, Berlin, Heidelberg, pp. 129–146. Springer.

[22] Furukawa, J. (2014) Short comparable encryption. *Cryptology and Network Security - 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*, Cham, pp. 337–352. Springer.

[23] Horst, C., Kikuchi, R., and Xagawa, K. (2017) Cryptanalysis of comparable encryption in sigmod'16. *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, New York, NY, USA, pp. 1069–1084. ACM.

[24] Li, J. and Omiecinski, E. (2005) Efficiency and security trade-off in supporting range queries on encrypted databases. *Data and Applications Security XIX, 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Storrs, CT, USA, August 7-10, 2005, Proceedings*, Berlin, Heidelberg, pp. 69–83. Springer.

[25] Bloom, B. H. (1970) Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, **13**, 422–426.

[26] Goh, E. (2003) Secure indexes. *IACR Cryptology ePrint Archive*, **2003**, 216.

[27] Curtmola, R., Garay, J. A., Kamara, S., and Ostrovsky, R. (2006) Searchable symmetric encryption: improved definitions and efficient constructions. *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, New York, NY, USA, pp. 79–88. ACM.

[28] Lin, H. and Tzeng, W. (2005) An efficient solution to the millionaires' problem based on homomorphic encryption. *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, Berlin, Heidelberg, pp. 456–466. Springer.

[29] Freedman, M. J., Nissim, K., and Pinkas, B. (2004) Efficient private matching and set intersection. *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, Berlin, Heidelberg, pp. 1–19. Springer.

[30] Lewi, K. (2016). Fastore-an implementation of order-revealing encryption. `https://github.com/kevinlewi/fastore`. Last accessed 20 December 2017.

[31] Granlund, T. and the GMP development team (2012). Gnu mp: The gnu multiple precision arithmetic library. `http://gmplib.org`. Last accessed 20 December 2017.

[32] Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011) Cryptdb: protecting confidentiality with encrypted query processing. *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, New York, NY, USA, pp. 85–100. ACM.

[33] Group, C. (2014). Cryptdb-a database system that can process sql queries over encrypted data. `https://github.com/CryptDB/cryptdb`. Last accessed 20 December 2017.

[34] Leskovec, J. and Krevl, A. (2014). Snap datasets: Stanford large network dataset collection (gowalla). `https://snap.stanford.edu/data/loc-gowalla.html`. Last accessed 20 December 2017.

[35] Chang, C.-C. and Lin, C.-J. (2011). Libsvm: a library for support vector machines (usps). `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#usps`. Last accessed 20 December 2017.