Privacy-Enhancing Range Query Processing over Encrypted Cloud Databases

Jialin Chi[1,2(✉)], Cheng Hong[1], Min Zhang[1], and Zhenfeng Zhang[1]

[1] Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China
{chijialin,hongcheng,mzhang,zfzhang}@tca.iscas.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, China

Abstract. The Database-as-a-Service (DAS) model allowing users to outsource data to the clouds has been a promising paradigm. Since users' data may contain private information and the cloud servers may not be fully trusted, it is desirable to encrypt the data before outsourcing and as a result, the functionality and efficiency has to be sacrificed. In this paper, we propose a privacy-enhancing range query processing scheme by utilizing polynomials and kNN technique. We prove that our scheme is secure under the widely adopted honest-but-curious model and the known background model. Since the secure indexes and trapdoors are indistinguishable and unlinkable, the data privacy can be protected even when the cloud server possesses additional information, such as the attribute domain and the distribution of this domain. In addition, results of experiments validating our proposed scheme are also provided.

Keywords: Database-as-a-Service · Cloud database · Range query

1 Introduction

With the rapid developments of networking and Internet technologies, Database-as-a-Service (DAS) [1] has been a promising paradigm, such as Amazon Web Services [2] providing both relational and NoSQL cloud-based database services. In the DAS model, organizations could outsource their data to service providers and retrieve data anytime and anywhere, as long as they have access to the Internet. In other words, the DAS model provides an approach for corporations to share the hardware and software resources as well as the expertise of database professionals, thereby cutting the cost of maintaining their own DBMSs. On the other hand, since data is stored at the third-party server and most organizations view their data as a valuable asset, at least two privacy issues arise. First, data needs to be protected from thefts from outsiders who may break into the providers' websites and scan disks. For instance, in 2014, hackers broke into the computers of Community Health Systems which operates 206 hospitals across the United States and stole data on 4.5 million patients [3]. Second, data also needs to be protected against the service providers, if they cannot

J. Wang et al. (Eds.): WISE 2015, Part II, LNCS 9419, pp. 63–77, 2015.
DOI: 10.1007/978-3-319-26187-4_5

be fully trusted. For example, an engineer in Google's Seattle offices broke into the Gmail and Google Voice accounts of several children in 2010 [4]. To guard data security and privacy, a straightforward solution is to encrypt data before outsourcing. When performing search, all encrypted data is returned and corporations could execute the query at the client after decrypting it. Obviously, this naive approach is impractical for incurring high decryption and network workloads for organizations.

The system model considered in this paper is comprised of three fundamental parties: the *data owner*, the *data user* and the *cloud server*, as illustrated in Fig. 1. The data owner outsources his/her data to the clouds in encrypted form, together with the secure index applied to enable the searching capability of the cloud server. To search over the encrypted data, the data user obtains the secure trapdoor from the data owner through search control mechanisms, e.g., broadcast encryption [8] and then submits it to the cloud server. Upon receiving the trapdoor, the cloud server searches the secure index and returns the matching encrypted data to the data user. Finally, the access control mechanism [5] is used to manage decryption capabilities. However, the search control and access control mechanisms are out of the scope of this paper. For a simple example, the data user can store his/her own data and query his/her own data on the cloud server. In this architecture, the data owner and the data user are trusted while the cloud server hosted by service providers cannot be fully trusted.

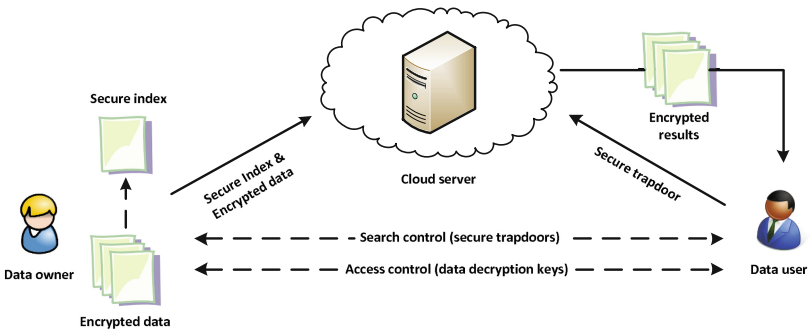

Fig. 1. Architecture of the range query processing over encrypted cloud databases

The problem discussed in this paper is range query which is a major type of database queries. The data in a relational DBMS are represented as tuples (i.e., records or rows) and tuples having the same attributes are organized in a table. For example, the Member data in Table 1 are personal information tuples of a special interest group and all tuples are identified by attributes *ID*, *NAME*, *AGE* and *ADDRESS* where the underlined attribute *ID* denotes the key of this table. For the attribute which can be represented as numerical values (e.g., *AGE*), consider a range query specified by an interval (e.g., *select* $*$ *from Member where* $AGE \in [20, 30)$), the matching results are the tuples whose attribute value falls into the

Table 1. Member

ID	NAME	AGE	ADDRESS
1	John	25	Chicago
2	Eric	21	Miami
3	Thomas	27	New York
4	Franklin	33	San Francisco
5	Aaron	18	Boston

Table 2. Enc_Member

Enc_tuple	I^S_{AGE}
SfahFrwierh3Jsejjsdfblklr5fsfWer	α
uDmsmdfOwe05u5jgNfjkgkroeRledlcf	β
mcuj48djhg2EdfslFlsfqjosdjRlslMd	γ
Ijfgkf8djkfsldDiseemJKllfnlfsj8k	η
osdfnsfklJfsnKdfannlK0nlfjalnn2z	π

interval (e.g., $ID \in \{1, 2, 3\}$). In addition, equal query can be viewed as a special kind of range query. Since computers can handle only inherently finite and discrete numbers, the attribute values and the lower and upper bounds of range queries can be assumed as nonnegative integers without loss of generality [25].

To protect the data privacy, there are several granularity choices for encryption, such as encrypting at the level of individual table, attribute, tuple and cell. Encrypting at the level of individual table or attribute implies that the entire table should be returned as the result, although it is more efficient to encrypt and decrypt the data. On the contrary, encrypting at the level of individual cell will incur high encryption and decryption workloads, while providing more efficient query processing. As in previous works, our proposed scheme performs encryption at the tuple level. To provide cloud servers with the ability to check whether one tuple matches the range query, we also associate with each encrypted tuple multiple secure indexes built based on the attribute values to be queried. Hence, each plaintext table can be stored as a table with one attribute for the encrypted tuple and several additional attributes for the secure indexes. More specifically, the plaintext tuple $t(A_1, \ldots, A_n)$ will be mapped onto a new tuple $t^S(Enc(t), I^S_1, \ldots, I^S_m)$ where $m \leqslant n$. The attribute $Enc(t)$ is utilized to store the encrypted tuple and I^S_i corresponds to the secure index over some A_j. In addition, the encryption function $Enc()$ is treated like a black box in this paper. For instance, as shown in Table 2, the indexed table Enc_Member contains the attribute *Enc_tuple* representing encrypted tuples and I^S_{AGE} representing secure indexes over attribute AGE. Here, the I^S_{AGE} attribute values are denoted as Greek letters.

The main challenge in this context is how to enable efficient range query processing over encrypted cloud databases without sacrificing privacy. In real situations, the service providers may possess background knowledge (e.g. the attribute domain and the distribution of this domain) which can be obtained from similar databases or historical data. So it is desirable that the cloud server can't learn more than the secure indexes, the secure trapdoors and the encrypted results even with such information. However, existing works [17–26] disclose useful statistical properties of data items through indexes or query processing, so the cloud server could learn additional information than allowed such as the exact attribute value of each tuple. Besides, to avoid incurring high decryption workload and network workload, it is better to reduce the interactions between server and client as well as the unmatching data items contained in results.

Our Contribution. In this paper, we propose a privacy-enhancing range query processing scheme. Our approach utilizes canonical ranges and polynomials to build indexes and trapdoors for attribute values and range queries. Then kNN technique is applied to encrypt the indexes and trapdoors. We provide thorough security analysis that the proposed scheme is secure under the honest-but-curious (HBC) model and the known background model. Since the indexes and trapdoors built in our scheme are indistinguishable and unlinkable, security and privacy can be protected even when the cloud servers possess the distribution of the attribute domain. Furthermore, we evaluate the performance of our scheme.

Organization. The rest of the paper is organized as follows. Section 2 presents the previous works related to our proposed scheme. The threat model and security goals are discussed in Sect. 3. In Sect. 4, we describe our privacy-enhancing range query processing in detail. Section 5 shows the security analysis and Sect. 6 gives our experimental results. We conclude the paper in Sect. 7.

2 Related Work

Previous works related to our scheme mainly fall into two categories: range queries, where search conditions are represented as intervals, and keyword queries, where search conditions are denoted as single or multiple keywords. The first keyword search scheme is proposed by Song *et al.* [6]. After this work, many novel schemes have been designed to improve the functionality and efficiency. Instead of scanning every word, [7,8] build secure indexes based on documents and corresponding search protocols to improve efficiency. Subsequent works [9–12] propose schemes to support multi-keyword retrieval, i.e., conjunctive or disjunctive keyword search. Moreover, [13–16] extend the search capability to fuzzy keyword search which can tolerate errors in the query to some extent. However, above works are limited and insufficient in executing range queries over encrypted data.

There are several works designed to enable privacy preserving and yet efficient range query processing. The bucketization technique proposed in [17] partitions the attribute domain into multiple buckets (in an equi-depth or equi-width manner) and each bucket is identified by a unique tag which can be realized by a

collision-free hash function. The bucket tag is maintained on the cloud servers as an index together with the encrypted tuples in this bucket. The trapdoor of a range query consists of the tags of buckets that overlap with the search condition and then all the encrypted tuples indexed by these tags will be returned. As mentioned in [18], the security limitation of data partitioning approaches is that the cloud servers can statistically estimate the attribute values of tuples and the lower and upper bounds of queries through domain knowledge and historical results. In addition, since false positives are incurred, i.e., the results may contain tuples that don't satisfy the query, users have to decrypt all the encrypted results and filter the unmatching items. Hence, the bucket size is positively related to security and negatively related to efficiency. In particular, increasing the bucket size can help to improve the data security, but will incur more unmatching results. There are several works focusing on how to trade off the security and efficiency. In [18], Hore *et al.* claim an optimal bucketization algorithm to maximize the efficiency as well as two measures of privacy, and propose a re-bucketization technique that yields bounded overhead while maximizing the defined notions of privacy. Subsequently, Wang *et al.* [19] introduce new security and efficiency metrics based on probability distribution variance and overlap ratio respectively, and design a 2-phase local overlap bucket (LOB) algorithm.

In [20], Agrawal *et al.* describe the first order preserving encryption approach for range queries over encrypted data, followed by [21] which proposes a provable secure scheme to achieve the same functionality. The schemes based on order preserving encryption are deterministic, since ciphertext must keep the same numerical ordering as plaintext. Hence, the cloud servers can obtain the relationship between the attribute values of two tuples directly from their corresponding indexes. In addition, these approaches assume that the distribution of attribute domain remains fixed and the encryption function is conscious of this distribution. The advantage of order preserving encryption schemes is that all results returned are matching tuples, so there is no additional workload incurred to the client.

In [22], damiani *et al.* design to build encrypted B^+-tree by encrypting the plaintext B^+-tree at node level. As the encrypted tree is not visible, interactions between server and client are required from the root to the leaf, and the number of interactions is equal to the depth of the tree. Obviously, it will increase decryption and network workloads for data users. In [23], Lu constructs the first provably secure logarithmic search mechanism. Pang *et al.* [24] show that the privacy of Damiani's encrypted B^+-tree can be defeated by monitoring the I/O activities on the server or tracking the sequence of nodes retrieved during range query processing. So they propose a privacy-enhancing PB^+-tree index which groups the nodes in each index level randomly into buckets, but this approach incurs higher I/O cost and computation overheads on the server. Since the ciphertexts are sorted in B^+-tree, the cloud server who possesses the distribution of domain can guess the attribute value of each tuple with high probability.

Besides the schemes based on bucketization technique, order preserving encryption and B^+-tree, there are also several works relying on other mechanisms. Li and Omiecinski [25] adapt a prefix-preserving encryption scheme to create index and this scheme is subject to certain attacks. In [26], Li *et al.* propose the first range query processing scheme that achieves index indistinguishability by utilizing PBtree (where "P" stands for privacy and "B" stands for Bloom filter) which has the property of node indistinguishability and structure indistinguishability. However, since the cloud sever can learn the tuples belonging to each prefix during the query processing, if the distribution of attribute domain is obtained, each plaintext value and the prefixes in trapdoors will be disclosed. In addition, the results in [26] may contain unmatching data items because this scheme utilizes Bloom filter to store each node's prefix family.

3 Problem Formulation

3.1 Threat Model

In this paper, we only consider attacks from the server providers while the data owner and the data user are trusted. We assume that the server is honest-but-curious (HBC) [27]. That means the cloud server will honestly and correctly follow the protocols, but may attempt to infer and analyze more information than allowed. Furthermore, we also consider the known background model, i.e., the cloud server possesses additional information about the data. For example, the attribute domain and its statistical information such as the distribution of this domain which may be obtained from similar databases or historical data.

3.2 Design Goals

Our design bears the following security and performance goals.

Security Goals

Index Confidentiality. The cloud server can't learn the exact attribute value of each tuple from its corresponding secure index. In addition, the secure index generation should be randomized instead of deterministic, i.e., the secure indexes are indistinguishable. For example, when given two secure indexes, the cloud server can't obtain the relationship between their corresponding attribute values.

Trapdoor Confidentiality. The cloud server can't learn the upper and lower bounds as well as the size of range query from its corresponding secure trapdoor. In addition, the secure trapdoor generation should be randomized instead of deterministic, i.e., the secure indexes are unlinkable. For instance, the cloud server can't deduce the relationship of any given trapdoors.

Query Processing Confidentiality. During the query processing, the cloud server can't obtain more than what can be derived from the results, even with background knowledge such as the attribute domain and the distribution of this domain.

Usability and Efficiency. For range query processing over clouds, since client devices mostly have limited storage and computing resources compared with cloud servers, it is better to process as much of the work as possible at the server, without having to decrypt the data. Besides, to avoid high decryption and network workloads, it is desirable to minimize the number of unmatching data items to be returned and interactions between client and server.

4 Privacy-Enhancing Range Query Processing

4.1 Main Idea

To design a privacy-enhancing and well-functioning method for range query, we focus on three important and closely inter-related aspects: (1) data structure utilized to build indexes for tuples and trapdoors for search conditions; (2) effective and efficient search algorithm that can check whether one tuple matches the given search condition; (3) privacy mechanisms that can be integrated with the above two aspects so that the privacy of indexes, trapdoors and query processing can be protected simultaneously. In this subsection, we describe the key ideas to realize data structure and search algorithm without privacy mechanisms. The more detailed scheme will be discussed in the next subsection.

In this paper, we assume the attribute to be queried is A whose value domain is $\mathbb{Z}_{2^n}$. Our proposed scheme first converts the attribute value $t.A$ of tuple t and the range query $\mathcal{Q} = [a, b)$ into canonical range representation which is also applied in [29]. Then use polynomials to construct index $\mathcal{I}$ for value $t.A$ and trapdoor $\mathcal{T}$ for search condition $\mathcal{Q}$. The detailed steps are explained as follows.

Canonical Range Representation of Value/Query. Consider the range query with $\mathbb{Z}_{2^n}$, a canonical range with level $i \in \mathbb{Z}_n$ is $[x2^i, (x+1)2^i)$ for some integer $x \in \mathbb{Z}_{2^{n-i}}$. There are 2^{n-i} canonical ranges in level i and the total number of different canonical ranges is $\sum_{i=0}^{n-1} 2^{n-i} = 2^{n+1} - 2$. Hence, we can identify each canonical range as a unique integer $cr \in \mathbb{N}_{2^{n+1}-2}$, which can be realized by a collision-free hash function $h : (i, x) \rightarrow \mathbb{N}_{2^{n+1}-2}$. The corresponding level of cr is denoted as $level(cr)$.

In particular, given the attribute value $t.A \in \mathbb{Z}_{2^n}$, for each level $i \in \mathbb{Z}_n$, compute x_i such that $t.A \in [x_i 2^i, (x_i + 1)2^i)$. The total number of canonical ranges containing $t.A$ is n and there is only one canonical range for each level. Hence the attribute value $t.A$ can be represented as $\text{CRA} = \{cra_0, cra_1, \ldots cra_{n-1}\}$ where $cra_i \in \mathbb{N}_{2^{n+1}-2}$. Given the range query $\mathcal{Q} = [a, b)$, for each level $i \in \mathbb{Z}_n$, find, if any, the minimum y_i such that $[y_i 2^i, (y_i + 1)2^i) \in [a, b)$ and the maximum z_i such that $[z_i 2^i, (z_i + 1)2^i) \in [a, b)$. If every element of one canonical range belongs to another canonical range, then the small range should be deleted from the range set. The total number of canonical ranges inside the given query is not fixed since it depends on the interval's upper and lower bounds. Hence the search condition $\mathcal{Q}$ can be represented as $\text{CRQ} = \{crq_0, crq_1, \ldots\}$ where $crq_i \in \mathbb{N}_{2^{n+1}-2}$. To check whether t is a matching tuple, i.e., $t.A \in \mathcal{Q}$, we only need to compute

CRA $\wedge$ CRQ. More specifically, If CRA $\wedge$ CRQ $\neq \varnothing$, then $t.A \in \mathcal{Q}$ and t is a matching tuple; otherwise, $t.A \notin \mathcal{Q}$ and t is an unmatching tuple.

For example, assume $n = 4$ and $h : (i, x) \rightarrow \mathbb{N}_{2^5-2}$. Given the attribute value $t.A = 9$, the canonical ranges containing $t.A$ are $\{[9, 10), [8, 10), [8, 12), [8, 16)\}$ and CRA $= \{h(0, 9), h(1, 4), h(2, 2), h(3, 1)\}$. Given the range query $\mathcal{Q} = [6, 11)$, the total canonical ranges inside the interval are $\{[6, 7), [10, 11), [6, 8), [8, 10)\}$ and CRQ $= \{h(0, 10), h(1, 3), h(1, 4)\}$ after deleting $h(0, 6)$. Since CRA $\wedge$ CRQ $= \{h(1, 4)\}$, we can obtain that $t.A \in \mathcal{Q}$ and t is a matching tuple.

Polynomial Representation of Index/Trapdoor. In this paper, we convert the canonical range set CRA of attribute value $t.A$ and CRQ of range query $\mathcal{Q} = [a, b)$ into polynomials. Before converting, we first put the n levels into M buckets randomly where M is an factor of n and each bucket has $m = n/M$ levels. Levels in the i^{th} bucket can be denoted as $\mathrm{B}_i = \{l_{i,0}, l_{i,1}, \ldots, l_{i,m-1}\}$. Second, we construct a $m+1$-degree polynomial, whose $m+1$ roots are denoted as $\mathrm{R} = \{a_0, \ldots a_m\}$:

$$\mathbf{P}_{\mathrm{R}}(x) = (x - a_0)(x - a_1) \cdots (x - a_m) = \sum_{i=0}^{m+1} \alpha_i x^i \tag{1}$$

Then produce a m-variable polynomial based on $\mathbf{P}_{\mathrm{R}}(x)$:

$$\mathbf{F}_{\mathrm{R}}(x_0, x_1, \ldots, x_{m-1}) = \mathbf{P}_{\mathrm{R}}(x_0)\mathbf{P}_{\mathrm{R}}(x_1) \cdots \mathbf{P}_{\mathrm{R}}(x_{m-1}) \tag{2}$$

Combine the terms of $\mathbf{F}_{\mathrm{R}}(x_0, x_1, \ldots, x_{m-1})$ only if they have the exact same coefficient. Then we utilize the coefficients U_{R} to build indexes and the variable parts V_{X} to generate trapdoors, where $\mathrm{R} = \{a_0, \ldots, a_m\}$ and $\mathrm{X} = \{x_0, \ldots, x_{m-1}\}$. For instance, assume $m = 2$ and construct a 2-variable polynomial $\mathbf{F}_{(a_0,a_1,a_2)}(x_1, x_2)$ based on the 3-degree polynomial $\mathbf{P}_{(a_0,a_1,a_2)}(x)$. The coefficients and variables of each term in $\mathbf{F}_{(a_0,a_1,a_2)}(x_1, x_2)$ are shown in Table 3.

The detailed protocols to construct indexes and trapdoors are described as follows:

Index. For each bucket $i \in \mathbb{Z}_M$, we build a sub-index $\mathcal{I}_i$ for attribute value $t.A$. First generate a random integer $r_i \notin \mathbb{Z}_{2^{n+1}-1}$. The canonical ranges containing $t.A$ that fall into the i^{th} bucket together with r_i are denoted as $\mathrm{CRA}_i = \{cra_{i,0}, \ldots, cra_{i,m-1}, r_i\}$, where $level(cra_{i,j}) \in \mathrm{B}_i$ and $\mathrm{B}_i = \{l_{i,0}, l_{i,1}, \ldots, l_{i,m-1}\}$ stands for the levels in the i^{th} bucket. Then use the coefficients $\mathrm{U}_{\mathrm{CRA}_i}$ to represent the sub-index $\mathcal{I}_i$, where the root set $\{a_0, \ldots, a_m\}$ is replaced by $\mathrm{CRA}_i = \{cra_{i,0}, \ldots, cra_{i,m-1}, r_i\}$.

Trapdoor. For each bucket $i \in \mathbb{Z}_M$, we build two sub-trapdoors $\mathcal{T}_{i,0}$ and $\mathcal{T}_{i,1}$ for the range query $\mathcal{Q} = [a, b)$. The canonical ranges inside $\mathcal{Q}$ that fall into the i^{th} bucket are $\mathrm{CRQ}_i = \{crq_{i,j}\}$ where $level(crq_{i,j}) \in \mathrm{B}_i$. Then add zeros to CRQ_i such that $|\mathrm{CRQ}_i| = 2m$ and split the set into two sets $\mathrm{CRQ}_{i,0}$ and $\mathrm{CRQ}_{i,1}$, where $|\mathrm{CRQ}_{i,0}| = |\mathrm{CRQ}_{i,1}| = m$. Then we use the variable parts $\mathrm{V}_{\mathrm{CRQ}_{i,0}}$ and $\mathrm{V}_{\mathrm{CRQ}_{i,1}}$ to construct two sub-trapdoors $\mathcal{T}_{i,0}$ and $\mathcal{T}_{i,1}$ respectively, where $\{x_0, \ldots, x_{m-1}\}$ is replaced by $\{crq_{i,0,0}, \ldots, crq_{i,0,m-1}\}$ and $\{crq_{i,1,0}, \ldots, crq_{i,1,m-1}\}$.

Table 3. The coefficients and variables of $\mathbf{F}_{(a_0,a_1,a_2)}(x_1, x_2)$

Coefficients	Variables
$a_0a_1 + a_0a_2 + a_1a_2$	$x_0^3x_1 + x_0x_1^3$
$(a_0a_1 + a_0a_2 + a_1a_2)^2$	x_0x_1
$-(a_0a_1a_2)$	$x_0^3 + x_1^3$
$(a_0a_1a_2)^2$	1
$-(a_0a_1 + a_0a_2 + a_1a_2)(a_0a_1a_2)$	$x_0 + x_1$
$(a_0 + a_1 + a_2)^2$	$x_0^2x_1^2$
$-(a_0 + a_1 + a_2)$	$x_0^3x_1^2 + x_0^2x_1^3$
1	$x_0^3x_1^3$
$-(a_0 + a_1 + a_2)(a_0a_1 + a_0a_2 + a_1a_2)$	$x_0^2x_1 + x_0x_1^2$
$a_0a_1a_2(a_0 + a_1 + a_2)$	$x_0^2 + x_1^2$

If there exists one canonical range cr satisfying $cr \in \mathrm{CRA}_i \wedge \mathrm{CRQ}_{i,k}$, and we assume $cr = cra_{i,0} = crq_{i,k,0}$. Since cr is a root of $\mathbf{P}_{\mathrm{CRA}_i}(x)$, i.e.,

$$\mathbf{P}_{\mathrm{CRA}_i}(cr) = 0 \tag{3}$$

then

$$\begin{aligned} \mathcal{I}_i\mathcal{T}_{j,k} &= \mathbf{F}_{\mathrm{CRA}_i}(cr, crq_{i,k,1}, \ldots, crq_{i,k,m-1}) \\ &= \mathbf{P}_{\mathrm{CRA}_i}(cr)\mathbf{P}_{\mathrm{CRA}_i}(crq_{i,k,1}) \cdots \mathbf{P}_{\mathrm{CRA}_i}(crq_{i,k,m-1}) \\ &= 0 \end{aligned} \tag{4}$$

The whole index $\mathcal{I}$ can be denoted as $\{\mathcal{I}_0, \ldots, \mathcal{I}_{M-1}\}$ while the whole trapdoor can be represented as $\{\mathcal{T}_{0,0}, \mathcal{T}_{0,1}, \ldots, \mathcal{T}_{M-1,0}, \mathcal{T}_{M-1,1}\}$.

4.2 Scheme Construction

In this subsection, we introduce the detailed process of our proposed scheme and pay more attention to the privacy mechanisms. In particular, we adopt the secure k-nearest neighbor (kNN) scheme proposed by Wong *et al.* [28] to encrypt the index $\mathcal{I}$ and trapdoor $\mathcal{T}$. The steps are explained as follows:

Setup. In this initialization phase, for each bucket $i \in \mathbb{Z}_M$, the data owner takes a security parameter λ_i and outputs the secret key SK_i, which includes: (1) a d-bit randomly generated vector S_i, (2) two invertible random matrices $M_{i,1}, M_{i,2} \in R^{d\times d}$, where d is the length of $\mathcal{I}_i$ and $\mathcal{T}_i$. Hence, SK_i can be denoted as a 3-tuple $\{S_i, M_{i,1}, M_{i,2}\}$.

BuildIndex. For each sub-index $\mathcal{I}_i$, the data owner encrypts the vector using the secret key SK_i. First, split $\mathcal{I}_i$ into two random vectors as $\{\mathcal{I}_i', \mathcal{I}_i''\}$ following the rule: for each element $\mathcal{I}_i[j], 0 \leqslant j \leqslant d-1$, if the j^{th} bit of S_i is 1, set $\mathcal{I}_i'[j] = \mathcal{I}_i''[j] = \mathcal{I}_i[j]$; if the j^{th} bit of S_i is 0, $\mathcal{I}_i'[j]$ and $\mathcal{I}_i''[j]$ are set to two

random numbers so that their sum is equal to $\mathcal{I}_i[j]$. Finally, the encrypted sub-index vector $\mathcal{I}_i^S$ is built as $\{M_{i,1}^T\mathcal{I}_i', M_{i,2}^T\mathcal{I}_i''\}$. The entire secure index $\mathcal{I}^S$ for attribute value $t.A$ is $\{\mathcal{I}_0^S, \ldots, \mathcal{I}_{M-1}^S\}$ and then the data user sends it to the cloud server.

BuildTrapdoor. For each sub-trapdoor $\mathcal{T}_{i,k}, k \in \{0, 1\}$, the data owner encrypts the vector using the secret key SK_i. First, split $\mathcal{T}_{i,k}$ into two random vectors as $\{\mathcal{T}_{i,k}', \mathcal{T}_{i,k}''\}$ following the rule: for each element $\mathcal{T}_{i,k}[j], \leqslant j \leqslant d-1$, if the j^{th} bit of S_i is 0, set $\mathcal{T}_{i,k}'[j] = \mathcal{T}_{i,k}''[j] = \mathcal{T}_{i,k}[j]$; if the j^{th} bit of S_i is 1, $\mathcal{T}_{i,k}'[j]$ and $\mathcal{T}_{i,k}''[j]$ are set to two random numbers so that their sum is equal to $\mathcal{T}_{i,k}[j]$. Finally, the encrypted sub-trapdoor vector $\mathcal{T}_{i,k}^S$ is built as $\{M_{i,1}^{-1}\mathcal{T}_{i,k}', M_{i,2}^{-1}\mathcal{T}_{i,k}''\}$. The entire secure trapdoor $\mathcal{T}^S$ for the search condition is $\{\mathcal{T}_{0,0}^S, \mathcal{T}_{0,1}^S, \ldots, \mathcal{T}_{M-1,0}^S, \mathcal{T}_{M-1,1}^S\}$ and then the data user sends it to the cloud server.

RangeQuery. With the secure trapdoor $\mathcal{T}^S$, the cloud server computes the inner product of $\mathcal{I}_i^S$ and $\mathcal{T}_{i,k}^S$ for each sub-trapdoor and check whether the result is zero.

$$\begin{aligned} \mathcal{I}_i^S\mathcal{T}_{i,k}^S &= \{M_{i,1}^T\mathcal{I}_i', M_{i,2}^T\mathcal{I}_i''\} \cdot \{M_{i,1}^{-1}\mathcal{T}_{i,k}', M_{i,2}^{-1}\mathcal{T}_{i,k}''\} \\ &= \mathcal{I}_i' \cdot \mathcal{T}_{i,k}' + \mathcal{I}_i'' \cdot \mathcal{T}_{i,k}'' \\ &= \mathcal{I}_i \cdot \mathcal{T}_{i,k} \end{aligned} \tag{5}$$

If there is some $\mathcal{T}_{i,k}^S$ that satisfies $\mathcal{I}_i^S \cdot \mathcal{T}_{i,k}^S = 0$, then tuple t is a matching tuple and should be returned to the data user; otherwise, t is an unmatching tuple. Then check the next tuple.

Discussion. In above protocols, the number of sub-trapdoors of $\mathcal{T}^S$ is $2M$, so we have to compute $2M$ inner products for each tuple. To improve the performance, if the number of canonical ranges in the j^{th} bucket is less than m, we can add zeros to the set so that $|\mathrm{CRQ}_i| = m$. Then we only need to construct one sub-trapdoor according to bucket j and hence reduce the computation workload.

5 Security Analysis

In this section, we discuss the security issues of our proposed scheme under the HBC model and the known background model.

Index Confidentiality. When constructing the secure index $\mathcal{I}^S$ for attribute value $t.A$, we insert random integer r_i to each canonical range set CRA_i used to generate the sub-index $\mathcal{I}_i$. In addition, each sub-index is also encrypted by using the secret key SK_i. Thus, as long as SK_i is kept private by the data owner, the secure index $\mathcal{I}^S$ is a totally obfuscated vector and even two tuples have the same attribute value, their corresponding secure indexes are different. As a result, the cloud server can only use the secure index to check whether one tuple is matching without directly learning any additional information, such as the exact value and the relationship between two tuples. Then the confidentiality of index can be protected.

Trapdoor Confidentiality. When generating the secure trapdoor $\mathcal{T}^S$ for range query $\mathcal{Q} = [a, b)$, each sub-trapdoor $\mathcal{T}_{i,k}$ is encrypted by the secret key SK_i and then the secure trapdoor is indistinguishable from a random vector. Thus, as long as SK_i is kept private, the cloud server can't obtain the lower and upper bounds as well as the size of the query. In addition, the relationship between two secure trapdoors can't be determined. Then the confidentiality of trapdoor can be protected.

Query Processing Confidentiality. During the query processing, the cloud server only obtains whether a tuple matches the query and which secure sub-trapdoor is matched. Since there are m levels in a bucket, even two tuples both match the same sub-trapdoor, the cloud server can't determine whether they belong to the same canonical ranges. Thus the relationship between two tuples won't be disclosed during the processing. In addition, the more levels in one bucket, the more secure the scheme is. If there is only one level in one bucket, the cloud server can obtain that the matching tuples corresponding to the same sub-trapdoor are close, so m should satisfy $m \geqslant 2$.

Privacy Against Statistics Analysis. In addition to the privacy of index, trapdoor and query processing, we also consider several certain statistics analyses. Since the adversary model in this paper is honest-but-curious, the cloud server can store the search history of each tuple t such as $history_t = \{Tag_{time}, Tag_{subtd}\}$, where Tag_{time} represents the time when the range query is required and Tag_{subtd} denotes which sub-trapdoor is matched. As the history becomes longer, the cloud server may learn that $t.A = t'.A$ with high probability, if $history_t$ and $history_{t'}$ are the same. In addition, if the number of matching tuples corresponding to the secure sub-trapdoor $\mathcal{T}_i^S$ is small, the cloud server may infer that there may be only one small canonical range used to build $\mathcal{T}_i^S$ and the attribute values of these tuples are close.

To prevent these certain attacks, we can add dummy integers to the canonical range set used to build trapdoor, since we have inserted random numbers to the canonical ranges applied to construct index. In particular, if the canonical range set $\mathrm{CRQ}_{i,k}$ used to build the sub-trapdoor satisfies $\sum_{j=0}^{m-1} |crq_{i,k,j}| < 2^m$ where $|crq_{i,k,j}|$ represents the size of $crq_{i,k,j}$, then we use dummy integers r to replace one zero element in the set $\mathrm{CRQ}_{i,k}$. As a result, the tuple t whose sub-index $\mathcal{I}_i$ has been added r will be returned no matter whether its attribute value $t.A$ satisfies $t.A \in Q$. Because of the random numbers added to indexes, the similarity between two histories of tuples t and t' has been broken even they correspond to the same attribute value. The limitation of this method is that the data user has to decrypt all the tuples received from the server and filter the unmatching results. The number of unmatching data items in results depends on the total number of random integers that can be chosen.

6 Performance Evaluation

We implement our proposed scheme in JAVA on desktop PC running Windows 7 Professional with *3.4 GHz Intel(R) Core(TM) i7-3770* processor and 4 GB RAM inside. Each data point is averaged over 10 runs.

6.1 Evaluation of Index Construction

In this paper, the secure index generated for tuple t is $\mathcal{I}^S = \{\mathcal{I}_0^S, \ldots, \mathcal{I}_{M-1}^S\}$, where $M = n/m$ and n is the number of total levels while m is the number of levels in one bucket. The length of each sub-index $|\mathcal{I}_i^S|$ depends on m directly. Thus main impacts that influence the time for generating indexes include m and n. As shown in Fig. 2(a), given $n = 12$, we set $m = 3$ and $m = 4$ respectively. The time required for $m = 4$ is higher because $|\mathcal{I}_i^S| = 126 * 2$ when $m = 4$ while $|\mathcal{I}_i^S| = 35 * 2$ when $m = 3$. Figure 2(b) illustrates that the time is also evidently affected by the number of levels n. Given that each bucket has $m = 3$ levels, the time for constructing indexes increases as n becomes larger, since the number of sub-trapdoors increases. When the number of data items is fixed, the ratio of time required for $n_1 = 15$ to $n_2 = 12$ approximately equals to n_1/n_2. In addition, to ensure the index indistinguishability, our scheme constructs the secure index for each tuple and thus the total time is linearly affected by the total number of data items.

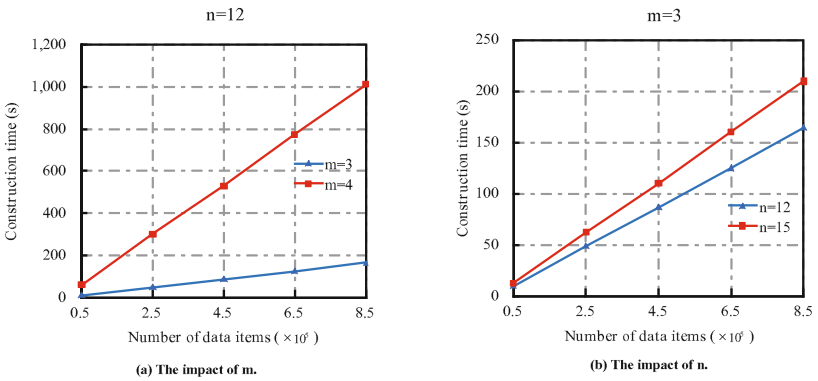

Fig. 2. Time for building indexes.

6.2 Evaluation of Query Processing

The time for query processing depends on the number of sub-trapdoors and the length of each sub-trapdoor. Figure 3(a) shows that if we put more levels in one bucket, the total time will increase since the length of each sub-trapdoor becomes longer. As illustrated in Fig. 3(b), as the number of sub-trapdoors in

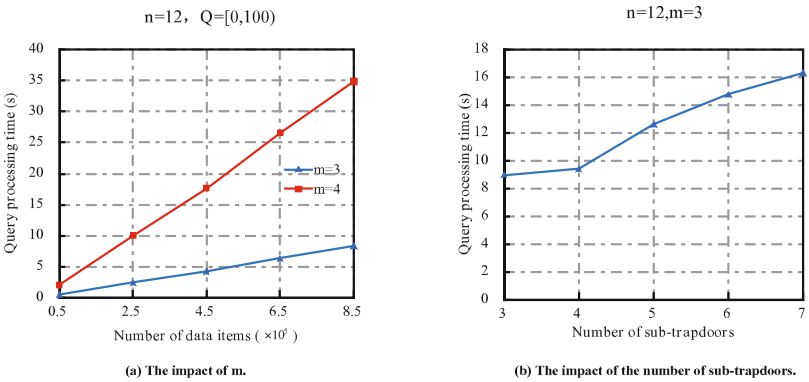

Fig. 3. Time for query processing.

a secure index grows, the total time will increase too. In addition, the time for query processing is also linearly related to the number of data items as shown in Fig. 3(a).

7 Conclusion

In this paper, we propose a privacy-enhancing scheme to realize range query processing over encrypted cloud databases. We first utilize the canonical ranges and polynomials to represent indexes and trapdoors, and then use kNN technique to encrypt them. Our scheme is secure under the widely adopted honest-but-curious model and the known background model. During the query processing, the data privacy can be protected even when the cloud server possesses the distribution of the attribute domain, since the indexes are indistinguishable and the trapdoors are unlinkable. In addition, by evaluating the performance of our scheme, we show that our system is also efficient.

References

1. Hacigumus, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proceedings of the 2002 ICDE International Conference on Data Engineering, pp. 29–38 (2002)
2. Amazon Web Services. http://aws.amazon.com/running_databases/?nc2=h_ls
3. Hospital Network Hacked, 4.5 Million Records Stolen. http://money.cnn.com/2014/08/18/technology/security/hospital-chs-hack
4. Google Fires Engineer for Privacy Breach. http://edition.cnn.com/2010/TECH/web/09/15/google.privacy.firing
5. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: Proceedings of the 2010 INFOCOM International Conference on Computer Communications, pp. 1–9 (2010)
6. Song, D., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 44–55 (2000)

7. Goh, E.J.: Secure indexes. Cryptology ePrint Archive: Report 2003/216 (2003)
8. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 2006 ACM CCS Conference on Computer and Communications Security, pp. 79–88 (2006)
9. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004)
10. Ballard, L., Kamara, S., Monrose, F.: Achieving efficient conjunctive keyword searches over encrypted data. In: Qing, S., Mao, W., López, J., Wang, G. (eds.) ICICS 2005. LNCS, vol. 3783, pp. 414–426. Springer, Heidelberg (2005)
11. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: Proceedings of the 2011 IEEE INFOCOM International Conference on Computer Communications, pp. 222–233 (2011)
12. Yu, J., Lu, P., Zhu, Y., Xue, G., Li, M.: Toward secure multi-keyword top-k retrieval over encrypted cloud data. In: Proceedings of the 2013 IEEE TDSC Transactions on Dependable and Secure Computing, pp. 239–250 (2013)
13. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: Proceedings of the 2010 IEEE INFOCOM International Conference on Computer Communications, pp. 1–5 (2010)
14. Chuah, M., Hu, W.: Privacy-aware bed-tree based solution for fuzzy multi-keyword search over encrypted data. In: Proceedings of the 2011 ICDCSW International Conference on Distributed Computing Systems Workshops, pp. 273–281 (2011)
15. Kuzu, M., Islam, M.S., Kantarcioglu, M.: Efficient similarity search over encrypted data. In: Proceedings of the 2012 IEEE ICDE International Conference on Data Engineering, pp. 1156–1167 (2012)
16. Wang, B., Yu, S., Lou, W., Hou, Y.T.: Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: Proceedings of the 2014 IEEE INFOCOM International Conference on Computer Communications, pp. 2112–2120 (2014)
17. Hacigumus, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD Special Interest Group on Management of Data, pp. 216–227 (2002)
18. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proceedings of the 2004 VLDB International Conference on Very Large Data Bases, pp. 720–731 (2004)
19. Wang, J., Du, X.: LOB: bucket based index for range queries. In: Proceedings of the 2008 WAIM International Conference on Web-Age Information Management, pp. 86–92 (2008)
20. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proceedings of the 2004 ACM SIGMOD Special Interest Group on Management of Data, pp. 563–574 (2004)
21. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009)
22. Damiani, E., Vimercati, S.D.C., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proceedings of the 2003 ACM CCS Conference on Computer and Communications Security, pp. 93–102 (2003)

23. Lu, Y.: Privacy-preserving logarithmic-time search on encrypted data in cloud. In: Proceedings of the 2012 NDSS Symposium Network and Distributed System Security Symposium (2012)
24. Pang, H., Zhang, J., Mouratidis, K.: Enhancing access privacy of range retrievals over B+-trees. IEEE TKDE Trans. Knowl. Data Eng. **25**, 1533–1547 (2013)
25. Li, J., Omiecinski, E.R.: Efficiency and security trade-off in supporting range queries on encrypted databases. In: Jajodia, S., Wijesekera, D. (eds.) Data and Applications Security 2005. LNCS, vol. 3654, pp. 69–83. Springer, Heidelberg (2005)
26. Li, R., Liu, A.X., Wang, A.L., Bruhadeshwar, B.: Fast range query processing with strong privacy protection for cloud computing. In: Proceedings of the 2014 VLDB International Conference on Very Large Data Bases, pp. 1953–1964 (2014)
27. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: Proceedings of the 2001 Annual ACM Symposium on Theory of Computing, pp. 639–648 (2001)
28. Wong, W.K., Cheung, D.W.L., Kao, B., Mamoulis, N.: Secure kNN computation on encrypted databases. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, pp. 139–152 (2009)
29. Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., et al.: Blind Seer: a scalable private DBMS. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP), pp. 359–374 (2014)