I. J. Computer Network and Information Security, **2017, 10, 13-23**
Published Online October 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2017.10.02

Multi-dimensional Range Query on Outsourced Database with Strong Privacy Guarantee

Do Hoang Giang
School of Computer Science and Engineering, Nanyang Technological University, Singapore
E-mail: do0004ng@e.ntu.edu.sg

Ng Wee Keong
School of Computer Science and Engineering, Nanyang Technological University, Singapore
E-mail: awkng@ntu.edu.sg

Received: 21 July 2017; Accepted: 07 September 2017; Published: 08 October 2017

***Abstract*—**Cloud services have provided important solutions for drastically reducing the cost of data management and maintenance. However, data outsourcing not only deprives clients of direct control over their data but also allows the server to gain direct access to the client data. Data encryption has been recognized as the solution to the privacy issue, but it also creates new challenges for both industry and academia. A naive question is whether the client still has the capability to query and obtain useful information when the data are encrypted and stored remotely. This paper investigates a solution to one of the most important types of query operations over encrypted data, namely multi-dimensional range queries. Our solution combines cryptographic techniques with the bucketization approach. We leverage a three-party architecture and secure multiparty computation to design and analyze the security of the protocols. Further, we discuss solutions for both static and dynamic datasets where new data records can be appended. First, we present the solutions for the case when the set of attributes in the query is pre-defined. Subsequently, we discuss the generalization.

Index Terms—Secure Computation, Multi-dimensional Range Query, Cryptography.

I. Introduction

The cloud computing paradigm has been recognized as an important solution for drastically reducing data storage and processing costs. Recent years have witnessed an increasing trend toward outsourcing data storage to cloud computing platforms such as Amazon EC2 [1], Google App Engine [2], and Microsoft Azure [3]. Companies and service providers in various fields, such as law enforcement, finance, and healthcare, are exploring solutions for outsourcing data. Outsourcing data enables clients to reduce not only infrastructure costs but also human costs.

Despite its advantages, data outsourcing has raised critical concerns over data privacy. It not only deprives clients of direct control over their data but also allows the server to gain direct access to the client data. Thus, information leakage can occur on the server side in various ways. The first threat is from corrupt employees who are able to access and reveal sensitive client information. Even if a cloud computing platform implements sufficiently robust policies against such privacy violations, it may be vulnerable to external malicious attacks. A successful cyberattack may lead to full exposure of client data. This threat is becoming increasingly apparent owing to the popularity of cyberattacks as services. Such threats are realistic, and a number of data breaches have been reported recently. Therefore, outsourcing plaintext datasets is strongly discouraged.

Data encryption is considered as the solution to the issue of data confidentiality. However, the context of data outsourcing introduces new challenges to managing encrypted data. The most critical issue is to preserve data usability in the encrypted dataset. This challenge has attracted considerable attention from both industry and academia in recent years. Searchable encryption techniques have been proposed to enable users to efficiently use their remote encrypted data without retrieving the entire data or exposing sensitive data to the data server. However, most existing methods focus on the problem of keyword matching and are inherently unsuitable for handling certain types of complex queries, such as multi-dimensional range queries, where the plaintexts are multi-numerical attributes.

In this paper, we study the problem of supporting an important class of complex search operations, namely multi-dimensional range queries. Multi-dimensional range queries are required for a wide range of practical applications, including network traffic log analysis, long-term health monitoring, and banking audits. There are essentially three broad categories of solutions: special data structure for range query evaluation, bucketization-based scheme, and order-preserving encryption-based techniques. However, these techniques fail to provide rigorous privacy protection owing to their statistical patterns, access patterns, or query leakage. Moreover, nearly all existing solutions support only static data (i.e., data that have been uploaded to the server only once) and previously known queries. To address the above-mentioned issues, this paper proposes a three-party architecture and investigates different protocols that support multi-dimensional range queries over encrypted remote data while rigorously guaranteeing privacy. We examine both static and dynamic cases in which data records can be appended to the existing data set. In addition, solutions for both fixed and unknown sets of queried attributes are studied.

The remainder of this paper is organized as follows. Section II reviews related studies on secure multi-dimensional range queries. Section III formulates the problem statement. Section IV describes our system model as well as the requirements for the designed protocols. Section V discusses the solution for the full-domain query in which the set of queried attributes is fixed and unchanged for all queries. This type of query is similar to most existing techniques for multi-dimensional range queries. Section VI presents solutions for a generalized version where the set of queried attributes can dynamically vary for different queries. The naive approach to this problem is to apply multiple single range queries. However, this approach leads to single-dimensional privacy leakage. In the proposed protocol, we leverage the concept of secure set intersection to resolve this issue. Section VII discusses practical considerations for the system implementation. Finally, Section VIII concludes the paper.

II. Related Works

This section reviews existing approaches toward to problem of secure multi-dimensional range query as well as oblivious RAM – a useful cryptographic technique we utilize to design the solutions.

A. *Secure Range Query*

There are essentially three general approaches for tackling multi-dimensional range queries on encrypted data: special data structure for range query evaluation, bucketization-based scheme, and order-preserving encryption-based techniques.

Special data structure. Boneh et al. [4] proposed a general public-key approach to support comparison, subset, and range queries on encrypted data by using hidden vector encryption, whose search complexity is O(mT), where m and T are the number of attributes and number of discrete values for each attribute, respectively. Moreover, the ciphertext size is relatively large owing to the use of composite-order bilinear map groups [5], which make it infeasible in many applications.

Shi et al. [6] proposed a scheme that supports a conjunction of range queries over multiple attributes (i.e., multi-dimensional range queries). The idea is to encrypt each data record as a point in multi-dimensional space. The query processing is equivalent to testing whether a point falls inside a hyper-rectangle for every data record. The authors used a segment tree data structure to represent the ranges for each dimension. Anonymous identity-based encryption is applied at each node of the tree with a different key. For a certain query, the client needs to reveal the keys corresponding to each range on each dimension. Hence, all the attributes on the range will be revealed after successful decryption at the server. Subsequently, different tree-based approaches have been proposed to tackle the problem of secure multi-dimensional range queries. Lu [7] proposed a range search scheme on encrypted data by leveraging predicate encryption and B+ trees. He extended the original scheme to support multi-dimensional range queries by replacing B+ trees with kd-trees in the implementation. Wang et al. [8] presented a scheme for evaluating multi-dimensional range queries by using asymmetric scalar-product preserving encryption and R-trees. However, these two methods lead to single-dimensional privacy leakage.

Bucketization-based scheme. Bucketization-based data representation for query processing in an untrusted environment was originally leveraged by Hacigumus et al. [9]. Their bucketization simply involves a data partitioning step based on equi-depth or equi-width partitioning to support single-dimensional range queries. The queries are mapped to a set of buckets that contain any value satisfying the range of the query. The original queries are translated into bucket-level queries, which request the encrypted buckets containing the desired values. Several studies have attempted to reduce bucket costs (i.e., false positives) while preserving the anonymity of the data set [10]. Hore et al. [11] extended the bucketization-based method to support secure multi-dimensional range queries. Their method tries to cluster d-dimensional space into various hyper-rectangles, and bucket indexing is performed on the clustered hyper-rectangles.

Order-preserving encryption based techniques. An order-preserving encryption scheme is a deterministic cryptosystem whose encryption algorithm produces ciphertexts that preserve the numerical ordering of the plaintexts. Roughly speaking, for any two ciphertexts c_1 and c_2 corresponding to plaintexts p_1 and p_2, respectively, if $p_1 \leq p_2$, then it is guaranteed that $c_1 \leq c_2$. Order-preserving encryption was first proposed by Agrawal et al. [12] to support range queries on encrypted data. When the dataset is encrypted by an order-preserving encryption scheme, the result of a single-dimensional range query is simply determined by the encryptions of the two end-points. The concept of order-preserving encryption was efficiently revised with formal security analysis by Boldyreva et al. [13] [14] and Mavroforakis et al. [15]. Although these techniques are highly efficient, they provide weak privacy. As such, traditional efficient indexes can be built directly on encrypted data and queried in the same manner as plaintexts. Nevertheless, order-preserving encryption is deterministic and hence suffers from inherent distribution leakage. In addition, it inevitably leaks the ordering of the data.

B. *Oblivious RAM*

Oblivious RAM (ORAM) is a cryptographic primitive that conceals memory access patterns. The idea of ORAM is to continuously reshuffle the memory and translate the address of each memory access to a set of randomized memory locations.

ORAM was first investigated by Goldreich [16] using the "square-root" solution. The objective was to address the problem of software protection, i.e., to prevent unauthorized copy. Goldreich's solution partitions the server memory into two regions: a main region of O(N) blocks and a shelter of $(O\sqrt{N}\,)$ blocks. Each access involves linear scan in the shelter region and one access to the main region. After $\left(O\sqrt{N}\right)$ operations, the data structure is reshuffled. Goldreich and Ostrosvky [17] presented the well-known "hierarchical" solution to achieve polylogarithmic memory bandwidth overhead. The key idea of this solution, in contrast to Goldreich's above-mentioned solution, is to organize the server memory into a hierarchy of buffers whose sizes grow at a geometric rate.

A recent breakthrough was made by Stefanov et al. [18], who mapped each data block to a random path of a binary tree. The information for the path of each data block is stored in a position map. To access a block, the client first looks up the position map and reads all the buckets on that path. Each data access requires $\mathrm{O}(\log^2 \mathrm{N})$ memory bandwidth overhead. The solution requires O(N) local data storage for the path of each data block. Shi et al. [19] proposed ORAM recursion to reduce the client storage to $O(1)$. The key idea is to store the position map on the server side as a second ORAM. This may be performed recursively until the client storage is O(1). Clearly, the access operation now includes looking up all the position map ORAMs in addition to the main data ORAM, which increases the bandwidth overhead to $O(log^3 N)$. Stefanov et al. [20] proposed an extremely simple and efficient ORAM protocol called Path-ORAM. Path-ORAM is the first technique to achieve $O(log\ N)$ memory bandwidth overhead under small client storage. In this paper, we refer to the Path-ORAM design when the ORAM technique is used.

III. Problem Formulation

We consider a scenario involving a data owner Alice. Alice possesses a multi-dimensional dataset D of n records. Each data record has m numerical attributes. Let a_i denote the identifier of the i-th attribute and x_j^i denote the value of the j-th attribute of the i-th record.

Alice wishes to outsource her dataset (i.e., the set of data records) to a cloud server to save local storage costs. A typical example is the data of body measurement recorded by sensors or mobile devices. Because mobile devices are normally limited in terms of storage capability, outsourcing such information to cloud services offers easy and low-cost solutions for long-term continuous health monitoring. On the other hand, such data include confidential personal information; hence, they should be stored in an encrypted form. At the same time, to facilitate health monitoring and treatment, the data owner needs to use the outsourced data efficiently.

To query the outsourced dataset, Alice performs a multi-dimensional range query. A multi-dimensional range query is a process of retrieval of data records that satisfy query values from the relative attribute domains:

$$Select\ *\ from\ D\ where\ a_{x_i} \in\ [s_{x_i}\ , t_{x_i}],$$
$$x_i \in\ S_q \in\ \{1, \cdots, m\}.$$

Let [a,b] represent an interval of integers from a to b, inclusively. The set of attribute identifiers $S_q\ =\{x_i\}$ is called the *set of queried attributes*. A multi-dimensional range query requests data records such as each attribute v_j in the *set of queried attributes*. Note that S_q takes values in the interval $[s_{x_i}, t_{x_i}]$. Roughly speaking, each data record can be considered as a point in m-dimensional space, and the multi-dimensional range query is defined by a hyper-rectangle. All the points inside this hyper-rectangle are considered as the results of the query.

When the *set of queried attributes* covers all m dimensions of the database D: $S_q\ =\{x_i\}\ =\{1, \cdots, m\}$, we say that the query is a full-domain query. Section V discusses the solution for this scenario. Section IV discusses viable solutions where the query's dimensions change dynamically from one query to another, with no fixed set of the concerned attribute domain for the query. Although studies have been conducted to address such a multi-dimensional query, they require the data owner to share his/her private key with a semi-trusted third party, which is not practical owing to privacy threats. In Sections IV and V, we will discuss the solutions in the settings of static and dynamic data storage. In the static database settings, the data owner uploads the entire data only once and is later able to query for the necessary records. On the other hand, in the dynamic database settings, new data records can be appended to the previously uploaded data, and expired data can be transferred to archive data storage and removed from the query results. We assume that the three parties loosely synchronize the time of archiving the data.

IV. System Models

A. System Model

We adopt a three-party architecture described by Boneh et al. [21] and Cristofaro et al. [22]. The system consists of three parties: the data server, the client, and the index server (see Fig.1).

1. *Client*: possesses a numerical multi-dimensional dataset and wishes to outsource his/her dataset in encrypted form to the data server. At some point, he/she should have the capability to securely construct a multi-dimensional range query to obtain the requested data.
2. *Data server*: provides storage services to the client. The cloud server is trusted to provide reliable services but it may also be curious about the content of the

data records stored in its database or the content of the queries submitted by the data owner.

3. *Index server*: stores meta-data to support answering of the query. As with the data server, it is trusted to store the meta-data as well as correctly process the query initiated by the data owner. In addition, it may be curious about the client's data as well as the query content.

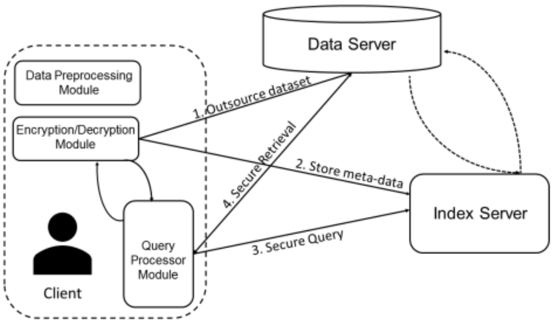

Fig.1. System Architecture

Workflow. Based on the notation in Fig.1, the workflow of our system can be described as follows. The data are first preprocessed by the data preparation module on the client side. The dataset and the meta-data (i.e., output of the preprocess phase) are encrypted by the encryption/decryption module and sent to the data server and the index server. The entire dataset should be encrypted by a symmetric encryption scheme. It involves low overhead for data storage and bandwidth, as well as efficient encryption/decryption operations. On the other hand, the meta-data should be encrypted by a specific public key cryptosystem and embedded in a special data structure to enable secure query processing.

To construct a multi-dimensional range query, the client constructs the query content using the query processor module, which has an interface with the encryption/decryption module. The encrypted query content is sent to the index server. The index server will output the corresponding indices of the satisfied data records so that the client is able to efficiently retrieve them from the data server. Further, the query processor module may be invoked to filter the results returned from the data server.

B. Security Assumptions and Requirements

In this paper, we assume that the user, cloud server, and index server are semi-honest. The servers will correctly follow the protocol specification; however, at the same time, they are also curious about the user's plaintext data. In addition, we assume that there is no collusion between two participating parties to access the user's confidential data. We emphasize that this assumption is realistic under the cloud environment. At present, with the development of cloud services, it is difficult to consider the possibility of collusion by well-established cloud service companies, which would damage their reputations and consequently have a negative impact on their revenues. In this work, we attempt to design a secure multi-dimensional range query while preserving the privacy of the database content and query values. Ideally, a secure protocol should only reveal what is leaked by the system parameters known to all parties and by the intended functional output. More specifically, we consider the following information leakage:

1. *Data Privacy.* A passive attacker who gets a snapshot of the encrypted database and encrypted index data should not be able to obtain any information about the user's private data. This implies that these two types of data should be encrypted by a probabilistic cryptosystem.
2. *Query Privacy.* The cloud and index servers should not be able to determine whether two queries are the same.
3. *Access Pattern.* Access to satisfied data records should not be revealed to the cloud server. Islam et al. [23] showed that data access pattern leakage could lead to the disclosure of a significant amount of sensitive information.
4. *One-dimensional Privacy.* Informally, single-dimensional privacy means that given a search token of a multi-dimensional range query, a computationally bounded adversary is not able to independently obtain the exact search results for any single-dimensional query.

While allowing the user to download the entire database and performing the query locally will achieve all the above-mentioned privacy requirements, this solution is infeasible. The proposed solution should be efficient in terms of time complexity, communication, and round complexity.

V. Full-Domain Multi-Dimensional Range Query

First, we describe the construction of full-domain multi-dimensional range queries. The dataset contains multiple data records that are points $\{x_j\}_{j=1}^{n}$ in a m-dimensional lattices. A full-domain query is defined by a hyper-rectangle in the m-dimensional space:

$$B = \{[s_1, t_1], [s_2, t_2], \dots, [s_m, t_m]\}$$

where $[s_i, t_i]$ represents a single-dimensional range. A data record $\{x_j\}_{j=1}^{n}$ satisfies a query represented by the hyper-rectangle B if and only if $x_j \in [s_j, t_j]$ for $\forall j$: $1 \leq j \leq m$.

As mentioned above, each record of the dataset is encrypted individually by a symmetric encryption scheme such as AES. This allows for efficient retrieval of specific records from data server. Hence, this work focuses on designing the data structure and algorithms to find the indices of data records that satisfy the query criteria. Specifically, we present a method for constructing the meta-data, the data structure to store the metadata at the index server, and the algorithms to process the query. Our solution for the full-domain multi-dimensional range query problem is inspired by the multi-dimensional bucketization approach. First, we partition the data space into M disjoint buckets. The number of buckets M is smaller than the number of data points n. Each data record belongs to exactly one bucket. The query processing translates into the problem of retrieving the bucket content. Hore et al. [11] claimed that nondeterministic encryption of the bucket labels does not raise the level of security, because simply encrypting bucket labels cannot protect the query privacy or access pattern from an adversary. In this paper, we leverage the ORAM technique to hide the access pattern and thus provide a stronger security guarantee.

A. Solution Overview

The dataset is first pre-processed by the data preprocessing module. It is partitioned into non-overlapping buckets, and the bucket label is set as the tag for each data record in the bucket. The data owner encrypts each original data record (i.e., a d-dimensional data point) and uploads it to the data server. Furthermore, an encrypted inverted index table is stored at the index server in the following form:

$$\langle\, Bucket\ Label, encrypted\ set\ of\ record\ indices\ \rangle$$

The idea of the solution is to leverage ORAM data access for each intersecting bucket without leaking any information about the data or the query content. The satisfied records are determined by buckets intersecting the query. Many existing studies have indicated that ORAM is impractical owing to its high computational cost. However, this claim is not necessarily true, especially because nearly all existing studies focus only on designing various methods to retrieve the indices of the satisfied records while leaving the actual data retrieval process to a black box. The black box is assumed to be a standard ORAM protocol for data records with known indices. It contradicts the above-mentioned concerns with regard to practical application. Recent results on Path-ORAM have shown that it is a practical tool for data access. When the record indices are known to the client, a constant number of communication rounds of ORAM data access should be required to retrieve the data.

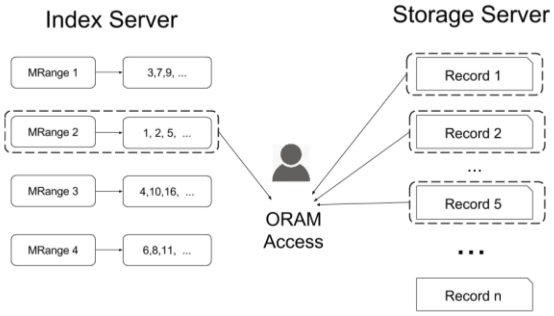

Fig.2. Secure Multi-dimensional Range Query from ORAM

The two main challenges faced by this approach are how to partition the space into buckets and how to embed the record indices into the ORAM data structure stored at the index server.

Bucketization algorithms. We retrieve a bucket if and only if it overlaps the query rectangle. We partition the space into M non-overlapping hyper-rectangles. Each rectangle contains approximately the same number of data points. The parameter M is determined by the tradeoff between the cost of false positives and the cost of communication between the client and the index server. When the number of buckets increases, each bucket contains fewer points, and the false positive rate decreases. However, the number of buckets intersecting with the query increases, which means that the number of ORAM accesses increases. On the other hand, when the number of buckets decreases to one, fewer ORAM accesses are required for each query. Further, when the false positive rate increases, additional bandwidth overhead is incurred to retrieve the actual data records from the server. With a certain parameter M, we use the Mondrian multi-dimensional partition algorithm presented by LeFerve et al. [24] (see Algorithm 1). In this protocol, we assume that $M = 2^k$ for simplicity.

```
Algorithm 1: $Partition(D, M)$
  Input: Dataset of multidimnensional points D, number of
         bucket $M = 2^k$
  Output: M buckets
1 $attr \to 1, \cdots, m$ ;
2 $splitVal \to find_m edian(D, attr)$ ;
3 $left\_half \to \{r \in D : r.attr \leq splitVal\}$ ;
4 $right\_half \to \{r \in D : r.attr > splitVal\}$;
5 return
    $Partition(left\_half, M/2) \bigcup Partition(right\_half, M/2)$;
```

At each call, the algorithm partitions the space according to the value of attribute *attr*, which is chosen at step 1. The data space is partitioned into two parts. Each part has approximately the same number of elements. The complexity of the preprocessing phase is $O(log\ n)$.

ORAM storage. Each data label is associated with a list of record indices. Our solution stores these indices in the ORAM data structure so that we can secure access to the content of a particular bucket. A naive solution is to use a sufficiently large ORAM data block to store the entire set of encrypted record indices. The user can retrieve the entire set of indices for a block label at once. However, because the size of the index set of each bucket label may vary, it is wasteful to use a large-sized data block to store a small piece of information. Moreover, it is computationally expensive to read from and write to a large-sized data block. Our approach is to represent the index sets in a compact form so that the data ORAM block can be determined beforehand with reasonable size.

B. Static Dataset

First, we consider the simplest scenario where the dataset is intact after uploading to the server. Archiving data is an example of this scenario.

The dataset is preprocessed and encrypted once before being uploaded to the data server. In the preprocessing stage, the bucketization algorithm (Algorithm 1) is applied to partition the space into buckets. Each record (i.e., a data point) belongs to exactly one bucket. Data are rearranged so that the identifiers of data records in the same bucket label form a consecutive numerical counter. Hence, the necessary information to reconstruct an index set is the starting counter and ending counter. The meta-data stored at the index server are ORAM data structures such that each block has an encrypted bucket label as the key for the encryption of the starting counter and ending counter as the values. Hence, to query for a data bucket, the data owner performs an ORAM data access with the index server; the access key is the bucket label. Then, the client obtains the range of data identifiers. The last step is to perform one more round of data access for each data identifier with the data server.

Analysis. Because the data and their indices are encrypted on the client side by a standard secure cryptosystem, the data server and index server obtain no information from the view of the stored data. Furthermore, the data owner accesses the index data at the index server and the data content at the data server by means of ORAM. The two servers are not able to obtain additional information of the query content or the access pattern. Because the data are obtained by the bucket label, one-dimensional privacy is satisfied for the query.

The bandwidth overhead of the solution is *O(log n)* owing to the overhead of Path ORAM. The client is also required to maintain a local storage of *O(log n)*. Finally, the complexity of each data access incurred at the server is *O(log n)*.

We also note that in the case of the static dataset, private information retrieval (PIR) can be used as an alternative to ORAM. In the PIR approach, for reading data, the computational complexity for the data owner is linearly related to the size of the retrieved data; in terms of the server size, the computational cost is linearly related to the size of the entire data set.

C. Dynamic Dataset

We consider the case where the update operation is allowed. More specifically, new data records may be appended to the existing dataset. Log file is an example of this scenario.

Because the new data records can be in any data bucket, we cannot use the previous approach. In the above-mentioned approach, the indices of data records in the same bucket label are required to form consecutive integers. Because the new records are not necessarily in the bucket containing the previous records, this requirement does not hold. We adopt another approach to index the records. We make use of the collision-resistance hash function. The index of a new record is determined by the hash of the previous record index that is in the same bucket. More specifically, we consider a collision-resistant hash function $h: \{0,1\}^n \mapsto \{0,1\}^m$ and define a sequence generated by h as follows:

$$a_0 = label - id, \qquad a_i = h(a_{i-1}).$$

To obtain the index set of the bucket label for the i-th bucket, we need to know the values of a_0^i and a_n^i, where a_n^i is the last element in the sequence generated for data records belonging to the i-th bucket. In this case, when the data are appended, the data owner is required to modify the corresponding ORAM data block to update the last element of the current index set. On the other hand, to obtain the index sets of buckets intersecting the query, the data owner retrieves the corresponding block to get $Enc(a_0^i)$ and $Enc(a_n^i)$. Further, the data owner decrypts the encrypted content and iteratively generates the sequence starting from the first element until the last element using the formula $a_i = h(a_{i-1})$. A collision-resistant hash function is required for this construction so that different data records are not mapped to the same index.

To insert a new record into the data set, the data owner first determines the bucket label of the new data record. Next, he/she performs a read and an update operation with the index server by means of ORAM. The new content of the ORAM block now contains the encrypted next sequential number of the last record index in this bucket.

Analysis. We use a collision-resistant hash function to index the records because each data record belongs to exactly on data bucket. Moreover, it is difficult to find a collision; each data record has a unique identifier determined by the hash functions.

The security of the dynamic case construction is analyzed in the same was as that of the static case. In both cases, the user accesses the indices and the real data records using ORAM. Hence, the only information leakage to the cloud server is the result size, which is the lower bound of the number of satisfied records. The reason is that the user is required to make at least the minimum number of satisfied record ORAM data accesses to the cloud server data.

The complexity and bandwidth overhead of the solution remain polylogarithmic owing to the use of Path ORAM. The update operation as an ORAM write operation also requires *O(log n)* memory bandwidth and computational complexity.

VI. Dynamic Multi-dimensional Range query

The aforementioned approach can be generalized to address the scenario in which the *set of queried attributes* is fixed. Roughly speaking, we should be able to efficiently represent the query space by a small number of fixed hyper-rectangles. In this section, we examine a more general case where the *set of queried attributes* varies for different queries. In other words, the number of dimensions in each query can vary from one to m - the number attributes in each data record. Clearly, we can treat the query as a full-domain query by considering each missing dimension as the full range. However, this approach normally leads to an excessive number of communication rounds with the index server. Instead of using the multi-dimensional bucketization technique, we apply the single-dimensional bucketization approach as a solution to this generalized problem.

A general multi-dimensional range query is defined by an unordered set:

$$B = \{[s_{i_1}, t_{i_1}], [s_{i_2}, t_{i_2}], \dots, [s_{i_k}, t_{i_k}]\}$$

$1 \leq i_j \leq m$, and $[s_{i_j}, t_{i_j}]$ represents a single-dimensional range. The size of the *set of queried attributes* is $k, 1 \leq k \leq m$.

A. Solution Overview

Our solution for the generalized problem employs private set intersection techniques. First, we partition each single-dimensional domain into buckets. A data record of m attributes falls into m buckets. For each single-dimensional range in the query, the requested data records must belong to the buckets intersecting with the range. A naive solution is to perform multiple queries on each queried dimension. The results are the intersection of the results. This approach has been used by Zhang et al. [25] to design multi-dimensional range queries in sensor environments. However, the naive approach fails to provide single-dimensional privacy because the server gains the results for each queried dimension. In this paper, instead of performing the intersection operation on the client side, we conduct it on the server side in a privacy preserving manner.

The query is firstly decomposed to each corresponding dimension. For each queried dimension, the intersecting buckets are enumerated. The query now can be translated into the one or multiple conjunctive queries. The variables in the conjunctive query are intersection buckets in each dimension. Finally, secure set intersection algorithm should be applied to answer the conjunctive query.

To illustrate our idea, we consider the following example. Alice has a dataset where each data record is a three-dimensional point (x, y, z). The attributes receive integer values in the domain [1,10]. Alice decomposes the data space with eight buckets: $[0,5]_x, [6,10]_x, [0,5]_y, [6,10]_y, [0,5]_z, [6,10]_z$. Each record is in exactly three buckets. Alice wishes to perform a simple multi-dimensional range query: $3 \leq x \leq 6, 2 \leq y \leq 4$. The query can be transformed into two conjunctive queries: $[0,5]_x \wedge [0,5]_y$ and $[6,10]_x \wedge [0,5]_y$. Query $[0,5]_x \wedge [0,5]_y$ (similar for the other) is translated as retrieving all the data records belonging to both $[0,5]_x$ and $[0,5]_y$. The answer for it is the intersection between two buckets.

Bucketization algorithms. We apply a simple equi-depth bucketization approach to partition each dimension for the data space. The number of buckets M in each dimension is determined by the tradeoff between the number of conjunctive queries and the false-positive rate. If we partition the dimension into many small buckets, we are able to reduce the false positive rate of the results. On the other hand, if M in each dimension is small, we are required to perform fewer conjunctive queries but additional work is required for the post-processing.

We now describe the method for securely answering each transformed conjunctive query in two different settings.

B. Static Dataset

For the case of a static dataset, the entire dataset is processed and uploaded only once to the servers. We leverage the Kissner-Song private set intersection [26] to perform conjunctive queries.

The idea underlying the Kissner-Song protocol is to fix a large field F and represent a set $S \subset F$ by a polynomial A_S that has zeros in all the elements of S, i.e., $A_S(x) = \prod_{s \in S}(x - s)$. To compute the intersection of many sets S_i, we construct a polynomial B whose zeros are the intersection of these sets. Clearly, if a point $s \in F$ *is* contained in all the sets S_i, then $A_{S_i}(s) = 0 \; \forall \, i$, and therefore, if we compute B as a linear combination of A_{S_i}'s, then $B(s) = 0$. On the other hand, if $A_{S_i}(s) \neq 0$ for some i and B is a random linear combination of A_{S_i}'s, then $B(s) \neq 0$ with high probability. Roughly speaking, instead of storing the record indices for each bucket, we store the coefficients of the polynomial that represents the index set. However, the coefficients should be encrypted so that the index server is not able to trace the indices.

In this paper, we use the BGN encryption technique [5] to encrypt the polynomial coefficients. The BGN cryptosystem, proposed by Boneh, Goh, and Nissim, allows both additions and multiplications with a constant size ciphertext. However, there is a catch: while the addition can be performed multiple times, only one instance of multiplication is permitted. Nevertheless, this protocol is considered to be much more practical than fully homomorphic encryption schemes. The homomorphism allows us to compute a linear combination of the polynomial in encrypted form so that set intersection operations can be securely performed on the index server.

Let Gen, Enc, Dec denote three BGN cryptosystem algorithms: key generation, encryption, and decryption. We have the following properties:

- Gen: generates a key pair (pk, sk) where pk and sk are public and private keys, respectively.
- $Dec(Enc(m)) = m$ for m in the message space.
- $Add(pk, Enc(x), Enc(y)) = Enc(x + y)$ can be performed multiple times on two encrypted values $Enc(x)$ and $Enc(y)$.
- $Mul(pk, Enc(x), Enc(y)) = Enc(x \cdot y)$ can be done only once.

Consider a bucket consisting of ℓ data records with indices $i_1, \cdots, i_\ell$. It can be represented by a polynomial of the form $A(x) = (x - i_1) \cdots (x - i_\ell) = \sum c_i \times x^i$. The coefficients c_i of the polynomial are encrypted by a BGN cryptosystem and stored at the index server. The index server stores $m \times M$ encrypted polynomials, where M is the number of buckets for each dimension and m is the number of attributes in each data record.

To issue a multi-dimensional range query (i.e.,

$Q = \{[s_{i_1}, t_{i_1}], \cdots, [s_{i_k}, t_{i_k}]\}$), the client first decomposes the query into k dimensions: $i_1, i_2, \cdots, i_k$. The query is transformed to multiple conjunctive queries; each one is associated with k buckets corresponding to k dimensions. To obtain the answer for each conjunctive query, the client does the following:

1. We consider a bit $b_i = 1$ if the $i-th$ bucket intersects with the query (0 otherwise). There are exactly k encryptions of bit 1 for each conjunctive query.
2. The client generates appropriate tags, $\sigma_i = Enc(b_i)$, and sends σ_i to the index server.
3. The index server generates random non-zero numbers p_i and computes $B(x) = \sum_i \sigma_i \times p_i \times A_i$ (in encrypted form), and sends the corresponding result to the user.
4. The client decrypts and factors the polynomial $B(x)$ and finds its roots, which are the indices of the records that are of interest to the user.
5. The client performs ORAM data access to obtain the necessary data records from the cloud server.

Analysis. To analyze the security of the proposed protocol, we need to examine the data view of the two servers. Because the server stores securely encrypted data, and the data are accessed only by ORAM, it gains no knowledge of the user sensitive data, query content, or access pattern. Moreover, the index server receives only BGN-encrypted bits; it is not able to obtain any information about the query. Hence, the proposed protocol leaks no information of the data content, query content, or query result to the server or the index server, except for the upper bound of the number of matching records (owing to the number of ORAM accesses).

The proposed solution requires the client to send $O(m \times M)$ encrypted bits to the index server and receive approximately $O(n/M)$ bits from the index server. At the same time, the index server is also required to perform up to $O(n)$ multiplication and exponentiation operations in the ciphertext space. The computation and communication costs for the server are the same as those of the previous methods.

C. Dynamic Dataset

In the dynamic dataset case, the new data records may be dynamically appended to the existing dataset. The aforementioned approach, which represents the index sets by polynomials, requires the entire encrypted coefficients for each corresponding bucket of the new record to be recomputed. We propose another approach that requires interaction between the server and the index server. We note that the abovementioned methods for both full-domain queries and dynamic multi-dimensional range queries do not require any communication between the two servers. The proposed protocol is inspired by a conjunctive keyword searchable encryption protocol proposed by Cash et al. [27]. We use it with a major modification to adapt our privacy requirements.

We start the protocol description by reviewing a few concepts related to bilinear maps. We will use the following notation:

1) G_1 and G_2 are two (multiplicative) cyclic groups of prime order p.
2) g_1 is a generator of $G_1$and g_2 is a generator of G_2.

<u>*INSERT (Record id:: $x = \{x_1, \cdots, x_m\}$)*</u>

- For each $i-th$ dimension ($1 \leq i \leq m$), the client does:
  - ▪ Let $j-th$ bucket of $i-th$ dimension: bucket l_i^j contains attribute value x_i.
  - ▪ Compute $xind \leftarrow F_{K_I}(id),\ z \leftarrow F_{K_Z}(l_i^j)$ and $y \leftarrow g_1^{\frac{xind}{z}}$
  - ▪ Set $eid \leftarrow Enc(id)$, append (eid, y) to a list for bucket l_i^j at index server.
  - ▪ Set $xtag \leftarrow e(g_1, g_2)^{(F_{K_X}(l_i^j) \cdot\ xind)}$ and append to a set S at cloud server.
- Encrypt the data record and upload the cloud server.

<u>*SEARCH ($Q = \{[s_{i_1}, t_{i_1}], \dots, [s_{i_k}, t_{i_k}]\}$)*</u>

- Client decomposes the query into k dimensions: $i_1, i_2, \dots, i_k$. The query is transformed into conjunctive query of buckets. For each conjunctive query q, we denote the buckets of interest for these k dimensions as $L_1, \dots, L_k$.
- For $i = 2, \dots, k$:
  - Client computes $xtoken_i \leftarrow g_2^{F_{K_Z}(L_1) \cdot F_{K_X}(L_i)}$
- Client sends L_1 , $\{xtoken_2, \cdots, xtoken_n\}$ to the index server.
- For each item (eid, y) in L_1 list in random order:
  - ▪ Index server sends eid to the cloud server.
  - ▪ Securely compute the cardinality $size$ of the intersection between set S at the server and $\{e(y, xtoken_2), \dots, e(y, xtoken_k)\}$
  - ▪ If $size = k - 1$, the cloud server sends eid to the client.
- Client decrypts and obtains the indices.
- Client performs ORAM data access to obtain the necessary data records from the cloud server.

A bilinear map is a map $e : G_1 \times G_2 \rightarrow G_T$ with the following two properties:

1) Bilinear: for all $u \in G_1, v \in G_2$, and $a, b \in Z$, $e(u^a, v^b) = e(u, v)^{ab}$.
2) Non-degenerate: $e(g_1, g_2) \neq 1$.

The proposed solution consists of two algorithms: INSERT and SEARCH for appending new records and performing multi-dimensional range queries on the encrypted data set respectively.

Let F denote a keyed pseudo-random function $f: Z_p \times K \to Z_p$, and select keys K_S, K_X, K_I, K_Z for F.

The correctness of the search protocol relies on the following fact:

$$\begin{aligned} e(y, xtoken\_i) &= e(g_1^{xind/z}, g_2^{F_{K_Z}(L_1)\cdot F_{K_x}(L_i)}) \\ &= e(g_1^{\frac{F_{K_I}(id)}{F_{K_Z}(L_1)}}, g_2^{F_{K_Z}(L_1)\cdot F_{K_X}(L_i)}) \\ &= e(g_1, g_2)^{F_{K_I}(id)\cdot F_{K_X}(L_i)} \end{aligned}$$

Hence, if the set $\{e(y, xtoken_2), \ldots, e(y, xtoken_k)\}$ is a subset of set S, the data record belongs to exactly all the requested buckets.

Our construction of the index list for the bucket and for the set S are similar to the construction of $TSet$ and $XSet$ proposed by Cash et al. [27]. In their protocol, $TSet$ and $XSet$ are stored in the same place in the cloud server. The correctness and the privacy of the two sets follow the proofs of the authors, given in [27]. However, because the set intersection is performed locally by the cloud server, that construction leads to unnecessary information leakage (i.e., access pattern). In the proposed construction, the storage of the bucket list and set S is separated into two parts, and the set intersection cardinality is obtained by a secure two-party computation protocol. We apply the protocol proposed by Cristofaro et al. [28]. Fig. 3 shows the workflow of the protocol. The protocol is secure under semi-honest model assumptions. The complexity is linearly related to the sizes of the two sets.

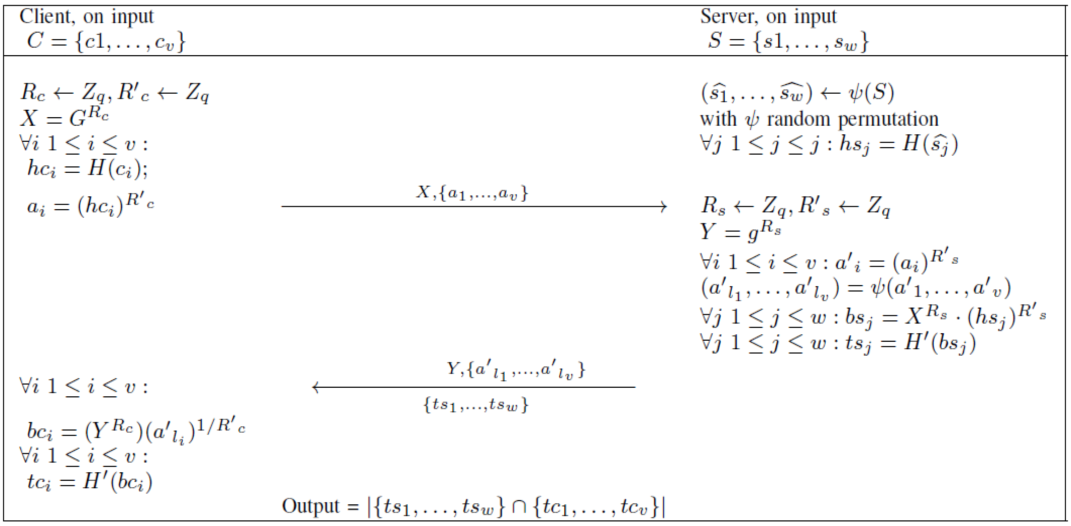
Client, on input
$C = \{c1, \ldots, c_v\}$
Server, on input
$S = \{s1, \ldots, s_w\}$
$R_c \leftarrow Z_q, R'_c \leftarrow Z_q$
$X = G^{R_c}$
$\forall i \; 1 \leq i \leq v:$
$hc_i = H(c_i);$
$a_i = (hc_i)^{R'_c}$
$(\widehat{s1}, \ldots, \widehat{s_w}) \leftarrow \psi(S)$
with ψ random permutation
$\forall j \; 1 \leq j \leq j : hs_j = H(\widehat{s_j})$
$X, \{a_1, \ldots, a_v\}$
$R_s \leftarrow Z_q, R'_s \leftarrow Z_q$
$Y = g^{R_s}$
$\forall i \; 1 \leq i \leq v : a'_i = (a_i)^{R'_s}$
$(a'_{l_1}, \ldots, a'_{l_v}) = \psi(a'_1, \ldots, a'_v)$
$\forall j \; 1 \leq j \leq w : bs_j = X^{R_s} \cdot (hs_j)^{R'_s}$
$\forall j \; 1 \leq j \leq w : ts_j = H'(bs_j)$
$Y, \{a'_{l_1}, \ldots, a'_{l_v}\}$
$\{ts_1, \ldots, ts_w\}$
$\forall i \; 1 \leq i \leq v:$
$bc_i = (Y^{R_c})(a'_{l_i})^{1/R'_c}$
$\forall i \; 1 \leq i \leq v:$
$tc_i = H'(bc_i)$
Output = $|\{ts_1, \ldots, ts_w\} \cap \{tc_1, \ldots, tc_v\}|$

Fig.3. Secure set Intersection Cardinality

Analysis. The insertion algorithm only appends the semantically secure encryptions to the server and index server. It leaks no information to the two servers; for details of the proof, readers may refer to [27].

During the search phase, the server only receives the encryption of the record index eid and knows whether the encrypted index belongs to the results. By observing the value of eid, the server is able to determine whether there are records that satisfy multiple queries. This information leakage can be eliminated by applying one more encryption layer on eid.

On the other hand, the newly proposed protocol does not provide query privacy to the index server. The index server is able to determine whether two queries are the same by observing the $xtoken$ sets received from the client. However, this is the only information leakage of the client's private data to the index server. Because the cardinality of private set intersection is only revealed to the cloud server, the index server does not obtain any information about the satisfied records.

The computational complexity and communication cost of the two servers are linearly related to the number of data records. The communication cost and computational complexity for the client are $O(max(log|D|, n))$, where $|D|$ is the number of data records and n is the dimension of the query Q.

VII. Experimental Results

We conducted a number of experiments to verify the practicality of the proposed solution. The experiments were performed on a synthetic dataset. We created the synthetic dataset by sampling 20,000 data points, each having 4 integral attributes from the domain [0,999].

Our implementation of Path ORAM requires up to 2 minutes to construct the ORAM data structures of the dataset of 10,000 data records. It also only takes only 0.02 s per access, including the time for decryption. This execution time is considered as a criterion for the tradeoff between the accuracy and the number of buckets, as discussed in Section V. For full-domain multi-dimensional range queries, we randomly generate 10,000 queries where each dimension is selected randomly from a uniform distribution from the same domain with the dataset. We took the average results of these queries and reported them.

Consider the number of buckets M. The accuracy of the intersection bucket approach is $0.1, 0.15, 0.3, 0.5$ for $M = 128,256,512,1028$, respectively. On the other hand, the numbers of buckets intersecting with the query are $22, 33, 53, 107$ for each configuration of M. Thus, when the parameter M increases, we are able to reduce the false-positive rate; however, we need to perform additional ORAM queries with the index server.

To test dynamic multi-dimensional range queries, we considered 5000 random queries. T*he set of queried attributes* contains 1–3 attributes. The two end points of each query are uniformly generated from the attribute domain.

Fig.4 shows the relation between the accuracy of the proposed approach and the number of buckets in each dimension (M). We conducted experiments with M in the range of 5–20. When we used only 5 buckets per dimension, the reported false positive rate was high, but we were only required to perform 5 conjunctive queries on average for each query. On the other hand, when we increased M to 20, the accuracy increased to 70%. However, we also needed to answer 60 conjunctive queries (i.e., depicted Fig.5) to obtain the complete set of requested records.

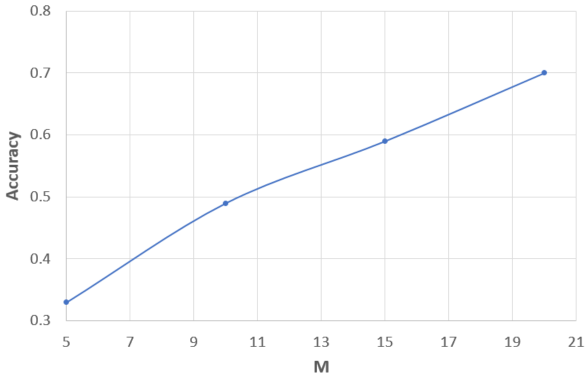

Fig.4. Accuracy for Different Numbers of Buckets (M)

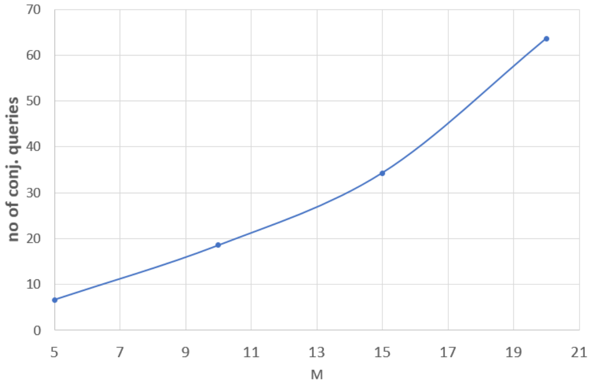

Fig.5. Number of Conjunctive Queries for Different Numbers of Buckets (M)

VIII. Conclusion

In this paper, we investigated different protocols to securely evaluate multi-dimensional range queries over encrypted data in cloud platforms. Our main idea is to leverage the bucketization algorithm to label the numerical range. The server stores the encrypted data, while the index server stores the meta-data output by the bucketization algorithm. When the multi-dimensional range queries are fixed beforehand, we perform two rounds of ORAM data access to answer the queries. The data records are labeled before encryption and sent to the server. On the other hand, to support more general queries, we presented novel solutions that allow multi-dimensional range queries to be answered, where the query constraints are not required to be fixed. The idea is to bucketize each attribute of a multi-dimensional data record and perform set intersection to answer conjunctive queries for multiple data labels. While the latter approach is able to support more general requirements, it is costly in terms of both computational complexity and bandwidth communication. In the most general case, where the dataset can be dynamically appended and the set of queried attributes is not fixed, our approach leaks a small amount of information to the index server.

References

[1] *Amazon Web Services (AWS) - Cloud Computing Services.*

[2] *Google App Engine.*

[3] *Microsoft Azure.*

[4] D. Boneh and B. Waters, "Conjunctive, Subset, and Range Queries on Encrypted Data," in *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007*.

[5] D. Boneh, E.-J. Goh and K. Nissim, "Evaluating 2-DNF Formulas on Ciphertexts," in *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005*.

[6] E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song and A. Perrig, "Multi-Dimensional Range Query over Encrypted Data," in *2007 IEEE Symposium on Security and Privacy S&P 2007, Oakland, California, USA, 20-23 May 2007*.

[7] Y. Lu, "Privacy-preserving Logarithmic-time Search on Encrypted Data in Cloud," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*.

[8] B. Wang, Y. Hou, M. Li, H. Wang and H. Li, "Maple: scalable multi-dimensional range search over encrypted cloud data with tree-based index," in *9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014*.

[9] H. Hacigümüs, B. R. Iyer, C. Li and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*.

[10] B. Hore, S. Mehrotra and G. Tsudik, "A Privacy-Preserving Index for Range Queries," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3, 2004*.

[11] B. Hore, S. Mehrotra, M. Canim and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *{VLDB} J.*, vol. 21, pp. 333-358, 2012.

[12] R. Agrawal, J. Kiernan, R. Srikant and Y. Xu, "Order-preserving encryption for numeric data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004,* 2004.

[13] A. Boldyreva, N. Chenette, Y. Lee and A. OŃeill, "Order-Preserving Symmetric Encryption," in *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009*.

[14] A. Boldyreva, N. Chenette and A. OŃeill, "Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, CA, USA, August 14-18, 2011*.

[15] C. Mavroforakis, N. Chenette, A. OŃeill, G. Kollios and R. Canetti, "Modular Order-Preserving Encryption, Revisited," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Australia, May 31 - June 4, 2015*.

[16] O. Goldreich, "Towards a Theory of Software Protection and Simulation by Oblivious RAMs," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York, New York, USA, 1987*.

[17] O. Goldreich, "Software Protection and Simulation on Oblivious RAMs," *J. ACM,* vol. 43, no. 3, pp. 431-473, 1996.

[18] E. Stefanov, E. Shi and D. X. Song, "Towards Practical Oblivious RAM," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*.

[19] E. Shi, T.-H. H. Chan, E. Stefanov and M. Li, "Oblivious RAM with O((logN)3) Worst-Case Cost," in *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011*.

[20] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," in *2013 {ACM} {SIGSAC} Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*.

[21] D. Boneh, C. Gentry, S. Halevi, F. Wang and D. J. Wu, "Private Database Queries Using Somewhat Homomorphic Encryption," in *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Canada, June 25-28, 2013*.

[22] E. D. Cristofaro, Y. Lu and G. Tsudik, "Efficient Techniques for Privacy-Preserving Sharing of Sensitive Information," in *Trust and Trustworthy Computing - 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011*.

[23] M. S. Islam, M. Kuzu and M. Kantarcioglu, "Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*.

[24] LeFevre, Kristen, D. J. DeWitt and R. Ramakrishnan, "Mondrian Multidimensional K-Anonymity," in *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA, 2016*.

[25] R. Zhang, J. Shi and Y. Zhang, "Secure multidimensional range queries in sensor networks," in *Proceedings of the 10th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2009, New Orleans, LA, USA, May 18-21, 2009*.

[26] L. Kissner and D. X. Song, "Privacy-Preserving Set Operations," in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005*.

[27] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu and M. Steiner, "Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries," in *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013*.

[28] E. D. Cristofaro, P. Gasti and G. Tsudik, "Fast and Private Computation of Cardinality of Set Intersection and Union," in *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012*.

Authors' Profiles

Hoang Giang Do received the B.Eng. degree in computer science and engineering from Nanyang Technological University, Singapore, in 2013, where he is currently working toward the Ph.D. degree. His current research interests include applied cryptography, privacy preserving query processing and applications of cryptocurrency.

Dr. Wee Keong Ng is Associate Professor in the School of Computer Science & Engineering, Nanyang Technological University, Singapore. He received his Ph.D. from the University of Michigan at Ann Arbor, USA. His research areas are secure data analytics, secure data storage, data monetization, and data security, and has published more than 200 technical papers in these areas. Dr. Ng has served in program/organizing committees of international conferences. In recent years, he is General Co-chair of the International Conference on Information and Communications Security 2016; Jury Member of the Second Dutch Cyber Security Research Award in March 2016; Senior PC Member of the 20th, 19th, 18th, 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining. Dr. Ng has advised and graduated more than 20 Ph.D. students and 20 Master students.

How to cite this paper: Do Hoang Giang, Ng Wee Keong,"Multi-dimensional Range Query on Outsourced Database with Strong Privacy Guarantee", International Journal of Computer Network and Information Security(IJCNIS), Vol.9, No.10, pp.13-23, 2017.DOI: 10.5815/ijcnis.2017.10.02