Privacy and Integrity Preserving Multi-dimensional Range Queries for Cloud Computing

Fei Chen*, Alex X. Liu†
*BloomReach Inc.
†Department of Computer Science and Engineering, University of Michigan State, USA
Email: chenfei.ts@gmail.com, alexliu@cse.msu.edu

***Abstract*—In cloud computing, a cloud provider hosts the data of an organization and replies query results to the customers of the organization. Because organization's data are confidential and the cloud provider cannot be fully trusted, some schemes have been proposed to preserve data privacy and query result integrity. However, these schemes either include false positives in query results, or are too expensive. In this paper, we propose an effective and efficient privacy and integrity preserving scheme for multi-dimensional range queries. To preserve privacy, we propose an order-preserving hash-based function to encode both data and queries so that a cloud provider can correctly process encoded queries over encoded data without knowing their values. To preserve integrity, we propose a new data structure called local bit matrices that allows a customer to verify the integrity of a query result with a high probability. Experimental results show that our scheme can efficiently process a dataset with one million data items.**

I. Introduction

A. Motivation and Problem Statement

Cloud computing has become a new computing paradigm, where cloud providers host significant amounts of hardware, software, and network resources, to store organizations' data and perform computation over the data on demand of customers' requests. It has three major advantages. First, organizations can instantly open business and provide products or services to their customers without building and maintaining their computing infrastructure, which significantly reduces costs. Second, the data stored in a cloud are more reliable and can be accessed whenever a customer has internet connection. Third, cloud providers have powerful computation capacity, which provides better experience to customers. Many clouds have been successfully built, *e.g.*, Amazon EC2 and S3 [1], Google App Engine [2], and Microsoft Azure [3].

Supporting databases is a must for cloud providers. Databases have been introduced since 1960s and become a key component for most applications. When organizations move their applications to clouds, database operations, especially database queries, should be supported by these clouds.

The database-as-a-service (DAS) model, first introduced by Hacigumus *et al.* [13], is one of the most important works in cloud computing, where a cloud provider hosts the data of an organization and replies query results to the customers on behalf of the organization, However, the DAS model brings significant security and privacy challenges. As cloud providers cannot be fully trusted and the data of an organization are typically confidential, the organization needs to encrypt the data before storing it in a cloud to prevent the cloud provider from revealing the data. However, it is difficult to process queries over encrypted data. Furthermore, since cloud providers serve as an important role for answering queries from customers, they may return forged data for the query or may not return all the data items that satisfy the query.

Therefore, we want to design a protocol for the DAS model that supports multi-dimensional range queries while preserving the privacy of both data and queries and the integrity of query results. The importance of this problem is also pointed out in [20]. Range queries are one of the most important queries for various database systems and have wide applications. In terms of data privacy, cloud providers cannot reveal the organization data and customer queries. Note that the customer queries also need to be kept confidential from cloud providers because such queries may leak critical information about query results. In terms of query result integrity, customers need to detect whether a query result includes forged data or does not include all the data items that satisfy the query.

B. Technical Challenges

There are three key challenges in solving secure multi-dimensional range queries problem in the DAS model. First, a cloud provider needs to correctly process range queries over encrypted data without knowing the values of both data and queries. Second, customers need to verify whether a query result contains all the data items that satisfy the query and does not contain any forged data. Third, supporting multi-dimensional range queries is a difficult problem.

C. Limitations of Previous Work

Four main techniques have been proposed in the privacy-preserving schemes: bucket partitioning (*e.g.*, [13], [14]), order-preserving hash functions (*e.g.*, [11]), order-preserving encryptions (*e.g.*, [4], [22]), and public-key encryption (*e.g.*, [6], [19]). However, bucket partitioning leads to false positives in query results, *i.e.*, a query result includes data items that do not satisfy the query. Existing order-preserving hash functions and order-preserving encryptions require large amount of shared secret information between an organization and its customers. The public-key cryptography is too expensive to be applied in realistic applications.

ISBN 978-3-901882-58-6

Three main techniques have been proposed in the integrity-preserving schemes: Merkle hash trees (*e.g.*, [10], [18], [23]), signature aggregation and chaining (*e.g.*, [17], [16], [24], [25]), and spatial data structures (*e.g.*, [7]). However, Merkle hash trees is expensive to support multi-dimensional range queries. Signature aggregation and chaining requires a cloud provider to reply to the customer the boundary data items of the query that do not satisfy the query. However, for applications that require strict access control, leaking boundary data items may violate access control policies. The importance of enforcing access control policies for DAS has been discussed in [7]. Furthermore, the chaining technique is expensive when applying it to multi-dimensional case. Spatial data structures are not clear to be used for searching query results over such structures in a privacy preserving manner.

D. Our Approach

In this paper, we propose an effective and efficient privacy and integrity preserving scheme for the DAS model. To preserve privacy, we propose an order-preserving hash-based function to encode both data and queries such that a cloud provider can correctly process encoded queries over encoded data without knowing their values. To preserve integrity, we present the first probabilistic integrity-preserving scheme for multi-dimensional range queries. In this scheme, we propose a new data structure, called *local bit matrices*, to encode neighborhood information for each data item from an organization, such that a customer can verify the integrity of a query result with a high probability.

Comparing with the state-of-the-art of privacy and integrity preserving schemes, our scheme achieves both security and efficiency. In terms of security, our scheme not only enables a cloud provider to correctly process queries over encrypted data, but also leaks only the minimum privacy information as we will discuss in Section IV-C. In terms of efficiency, our scheme is much more efficient due to the use of the hash function and symmetric encryption.

E. Key Contributions

In this paper, we make three key contributions. First, we propose an efficient privacy-preserving scheme that can process multi-dimensional range queries without false positives. Second, we propose the first probabilistic scheme for verifying the integrity of range query results. This scheme employs a new data structure, *local bit matrices*, which enables customers to verify query result integrity with high probability. Third, we conduct extensive experiments on real and synthetic datasets to evaluate the effectiveness and efficiency of our scheme.

F. Summary of Experimental Results

We performed extensive experiments on synthetic datasets and the Adult dataset [12]. Our experimental results show that our scheme is efficient for preserving privacy and integrity of multi-dimensional range queries in cloud computing. We chose three attributes of the Adult dataset in the experiments. For a synthetic dataset with one million 1-dimensional data items, the one-time offline data processing time is about 50 minutes, the space cost is 33MB, and the query processing time is 2 milliseconds. For the Adult dataset with 45222 3-dimensional data items, the data processing time is 104 seconds, the space cost is 1.5MB, and the query processing time is 3.5 milliseconds.

II. Related Work

A. Privacy Preserving in Databases

Database privacy has been studied extensively in both database and security communities (*e.g.*, [13], [14], [4], [5], [22], [11], [19], [6], [20], [21]). Based on whether query results include false positives, *i.e.*, data items that do not satisfy the queries but are included in the query results, we can classify these schemes into the following two categories: approximate privacy-preserving schemes [13], [14] and precise privacy-preserving schemes [4], [5], [22], [11], [19], [6]. The *approximate privacy-preserving schemes* reply query results with false positives but they are more efficient, while the *precise privacy-preserving schemes* reply query results without false positives but they are more expensive. Next, we discuss these two categories of privacy-preserving schemes.

Approximate Privacy-Preserving Schemes

Hacigumus *et al.* first proposed the bucket partitioning idea for querying encrypted data in the DAS model [13]. The basic idea is to divide the attribute domains into multiple buckets and then map bucket ids to random numbers for preserving privacy. Later, Hore *et al.* explored the optimal partitioning of buckets [14]. However, as pointed out in [14], bucket partitioning incurs a tradeoff between privacy and efficiency. If the bucket sizes are large, less privacy information is leaked, but query results include more false positives; if the bucket sizes are small, more privacy information is leaked, but query results include less false positives. In contrast, our scheme in this paper is a precise privacy-preserving scheme, which returns query results without false positives. Furthermore, our scheme leaks only the minimum privacy information for any possible precise privacy-preserving scheme, which will be discussed in detail in Section IV-C.

Precise Privacy-Preserving Schemes

Previous order-preserving hash functions [11] and order-preserving encryptions [4], [5], [22] can be employed for constructing precise privacy-preserving schemes in cloud computing. Fox *et al.* proposed an order-preserving minimal prefect hash function (OPMPHF) for a domain with N possible values [11]. Agrawal *et al.* proposed an order-preserving encryption (OPES) [4]. The basic idea of OPES is to transform data items to different values such that the transformed values preserve the order of the data items without disclosing the privacy of the data items to cloud providers. Specifically, this scheme first divides the data domain into multiple buckets, *i.e.*, m buckets, computes a transformation polynomial function for each bucket, and then applies the corresponding transformation function to the data items in each bucket. However, for z-dimensional data items, the OPMPHF function requires $O(zN \log N)$ shared secret information between the

organization and its customers, and the OPES encryption requires $O(zm)$ shared secret information. In contrast, our order-preserving hash-based function only requires $O(z)$ shared secret information.

B. Integrity Preserving in Databases

Independent of database privacy, database integrity has also been explored in prior work [10], [18], [23], [17], [16], [24], [25], [8], [7]. Merkle hash trees have been used for the authentication of data items [15] and they were used for verifying the integrity of range queries in [10], [18], [23]. Pang *et al.* [17], Narasimha & Tsudik [16], and Hu *et al.* [24] proposed similar schemes for verifying the integrity of range query results using signature aggregation and chaining. For each data item, Pang *et al.* computed the signature of the data item by signing the concatenation of the digests of the data item itself as well as its left and right neighbors [17]. Narasimha & Tsudik computed the signature by signing the concatenation of the digests of the data item and its left neighbors along each dimension [16]. Chen & Liu proposed a chaining technique to verify the integrity of range query results in two-tiered sensor networks [25]. It directly concatenates the data item with its left neighbors along each dimension. However, signature aggregation and chaining has two major drawbacks. First, it does support access control because it requires a cloud provider to reply to the customer the boundary data items of the query that do not satisfy the query. The importance of enforcing access control policies for DAS has been discussed in [7]. For applications that require strict access control, leaking boundary data items violates access control policies. Second, chaining technique is expensive when applying it to the multi-dimensional case. Considering a z-dimensional data D_j, for each dimension, it has a left neighbor and a right neighbor. Overall, D_j has $2 \times z$ neighbors. The chaining data structure consists of D_j and its $2 \times z$ neighbors. Thus it takes $O(z)$ space to store a chaining structure.

Chen *et al.* proposed Canonical Range Trees (CRTs) to store the counting information for multi-dimensional data such that the counting information can be used for integrity verification without leaking boundary data items of the query [7]. However, the most important requirement in cloud computing is to preserve data privacy. CRTs contain a lot of privacy information. Thus, we need to preserve the privacy of CRTs. As we discussed before, preserving privacy of relational databases is already difficult. Preserving privacy of CRT trees is much more difficult and no scheme has been proposed.

III. Models and Problem Statement

A. System Model

We consider the DAS model in Figure 1. The DAS model consists of three parties: an organization, a cloud provider, and customers. The organization outsources their private data to a cloud provider. A cloud provider hosts outsourced data from the organizations and processes the customers' queries on behalf of the organization. Customers are the clients of the organization that query a cloud provider and retrieve query results from the outsourced data in the cloud provider.

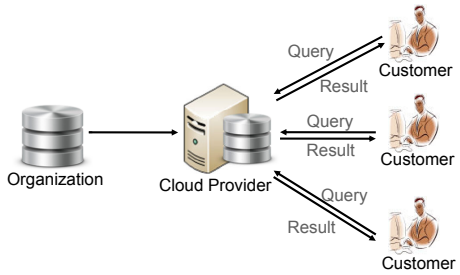

Fig. 1. The DAS model

B. Threat Model

We assume that organizations and their customers are trusted but the cloud provider is not. In a hostile environment, both customers and cloud providers may not be trusted. If a customer is malicious, it may retrieve all the organization's data and distribute to other unauthorized users. Such attack is very difficult to be prevented and is out of the scope of this paper. In this paper, we mainly focus on the scenario where the cloud provider is not trusted and it may try to reveal organizations' data and falsify the query results. In reality, cloud providers and organizations typically belong to different parties, *i.e.*, different companies. The organizations cannot share their private data with untrusted cloud providers. A malicious cloud provider may try to reveal the private data of organizations, and return falsified query results that include forged data or exclude legitimate data. We also assume that there are secure channels between the organization and the cloud provider, and between the cloud provider and each customer, which can be achieved using protocols such as SSL.

C. Problem Statement

The fundamental problem for the DAS model is: *how can we design the storage scheme and the query protocol in a privacy and integrity preserving manner*? A satisfactory solution should meet the following three requirements. (1) *Data and query privacy*: Data privacy means that a cloud provider cannot reveal any data item from organizations. Query privacy means that a cloud provider cannot reveal any query from customers. (2) *Data integrity*: If a cloud provider returns forged data or does not return all the data items that satisfy the query, such misbehavior should be detected by the customer. (3) *Range Query Processing*: The encoded data from organizations and encoded queries from customers should allow a cloud provider to correctly process range queries.

IV. Privacy Preserving for 1-dimensional Data

In this section, we present our privacy-preserving scheme for 1-dimensional data. To preserve privacy, it is natural to have an organization to encrypt its data items. Let $d_1, \cdots, d_n$ denote n data items, the encryption results can be denoted as $(d_1)_k, \cdots, (d_n)_k$, where k is the shared secret key between the organization and its customers. However, the key challenge is how a cloud provider can process queries over encrypted data without knowing the values of data items.

The basic idea of our scheme is to design an order-preserving hash-based function to encode the data items from the organization and the queries from its customers such that the cloud provider can use the encoded queries and encoded data items to find out the query results without knowing the actual values. More formally, let f_k denote the order-preserving hash-based function, where k is the shared secret key between the organization and its customers. To compute f_k, the organization and its customers also need to share the secret information, the domain of the data items $[x_1, x_N]$. This function f_k satisfies the following property: the condition $f_k(x_{i_1}) < f_k(x_{i_2})$ holds if and only if $x_1 \leq x_{i_1} < x_{i_2} \leq x_N$. To submit n data items $d_1, \cdots, d_n$ to a cloud provider, the organization first encrypts each data item with the secret key k, *i.e.*, $(d_1)_k, \cdots, (d_n)_k$. Second, the organization applies the function f_k to each data item, *i.e.*, $f_k(d_1), \cdots, f_k(d_n)$. Finally, the organization sends the encrypted data items $(d_1)_k, \cdots, (d_n)_k$ as well as the encoded data items $f_k(d_1), \cdots, f_k(d_n)$ to the cloud provider. To perform a range query $[a, b]$, the customer applies the order-preserving hash-based function f_k to the lower and upper bounds of the query, *i.e.*, $f_k(a)$ and $f_k(b)$, and then sends $[f_k(a), f_k(b)]$ as the query to the cloud provider. Finally, the cloud provider can find out whether the data item d_j $(1 \leq j \leq n)$ satisfies the query $[a, b]$ by checking whether the condition $f_k(a) \leq f_k(d_j) \leq f_k(b)$ holds. Fig. 2 shows the idea of our privacy-preserving scheme.

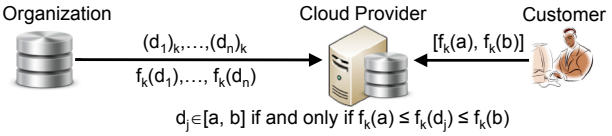

Fig. 2. Basic idea of privacy-preserving scheme

Next, we first present our order-preserving hash-based function and then discuss its properties. Second, we propose the privacy-preserving scheme by employing this function. Third, we propose an optimization technique to reduce the size of the results after applying the hash-based function. Fourth, we analyze the information leakage of our approach.

A. Order-Preserving Hash-based Function

In this paper, we propose an order-preserving hash-based function f_k. Let $\{x_1, \cdots, x_N\}$ denote the set that includes all possible data items and Δ denote a fixed value. Without loss of generalization, we assume that for any two adjacent data items x_i and x_{i+1} $(1 \leq i \leq N-1)$, $x_i < x_{i+1}$ and $x_{i+1} - x_i = \Delta$. Obviously, the number of all possible values is N and a value x_i $(1 \leq i \leq N)$ is equal to $x_1 + (i-1) \times \Delta$. For ease of presentation, in the rest of the paper, we use $[x_1, x_N]$ to denote the domain of these N data items and we consider only the case of $\Delta = 1$. The function f_k is

$$f_k(x_i) = \sum_{q=1}^{i} h_k(x_q) \tag{1}$$

where h_k is a keyed hash function, such as keyed HMAC-MD5 and keyed HMAC-SHA1. The intuition of designing such order-preserving hash-based function is two-fold. First, we leverage a normal hash function h_k as the basic building block such that the one-way property of h_k prevents the cloud provider from revealing the data items. Second, we consider the result of h_k as a positive integer and then calculate $f_k(x_i)$ by summing the hash results of all values that are less than or equal to x_i in the domain $[x_1, x_N]$ such that if $x_1 \leq x_{i_1} < x_{i_2} \leq x_N$, $f_k(x_{i_1})$ is less than $f_k(x_{i_2})$. In other words, f_k is an order-preserving function for the values in $[x_1, x_N]$. More formally, the order-preserving hash-based function f_k satisfies the following two properties:

Order Preserving: *Assume that $h_k(x_q)$ is a positive integer for any $x_q \in [x_1, x_N]$. For any $x_{i_1}, x_{i_2} \in [x_1, x_N]$, $f_k(x_{i_1}) < f_k(x_{i_2})$ if and only if $x_{i_1} < x_{i_2}$.*

Collision Resistance: *Assume that $h_k(x_q)$ is a positive integer for any $x_q \in [x_1, x_N]$. For any $x_{i_1}, x_{i_2} \in [x_1, x_N]$, it is impossible to find x_{i_1} and x_{i_2} where $x_{i_1} \neq x_{i_2}$ such that $f_k(x_{i_1}) = f_k(x_{i_2})$.*

These two properties and the one-way property of h_k allow the cloud provider to process the encoded range queries over the encoded data without revealing the values of the data and queries. Note that these two properties are holds for any value x_i in the domain $[x_1, x_N]$. If a value $x \notin [x_1, x_N]$, the two properties may not hold.

In fact, the hash-based function f_k can preserve any given *arbitrary order* of values in the domain $[x_1, x_N]$ no matter whether the condition $x_1 < \cdots < x_N$ holds. For example, if the order of 3 data items $3, 5, 7$ is defined as $5, 3, 7$, then $f_k(5) < f_k(3) < f_k(7)$. This property allows an organization to revise any data item x_i ($x_i \in [x_1, x_N]$) arbitrarily while still preserving the order. We will discuss how to leverage this property to prevent the statistical analysis of multi-dimensional data in Section VI-A.

B. The Privacy-Preserving Scheme

The privacy-preserving scheme includes three phases, *data submission*, *query submission*, *query processing*.

Data submission

The data submission phase concerns how an organization sends its data to a cloud provider. Let $d_1, \cdots, d_n$ denote the data items of an attribute in the private data of the organization. Recall that $[x_1, x_N]$ is the domain of the attribute and is the shared secret between the organization and its customers. Particularly, this shared secret consists of three values, the minimum value, x_1, the maximum value, x_N, and the difference of two adjacent values in the domain, Δ. For simplicity, we assume $d_1 < d_2 < \cdots < d_n$. If some data items have the same value, the organization can simply represent them as one data item annotated with the number of items that share this value.

To preserve data privacy, for each d_j $(1 \leq j \leq n)$, the organization first encrypts it with its secret key k, *i.e.*, $(d_j)_k$, and then applies the order-preserving hash-based function, *i.e.*, $f_k(d_j)$. Finally, the organization sends the encrypted data $(d_1)_k, \cdots, (d_n)_k$ as well as the hash results $f_k(d_1), \cdots, f_k(d_n)$ to the cloud provider.

Query submission

The query submission phase concerns how a customer sends a range query to the cloud provider. When a customer wants to perform a range query $[a, b]$ on the cloud provider, it first applies the order-preserving hash-based function to the lower and upper bounds of the query, *i.e.*, $f_k(a)$ and $f_k(b)$. Note that a and b are also two values in $[x_1, x_N]$. Finally, the customer sends $[f_k(a),\ f_k(b)]$ as a query to the cloud provider.

Query Processing

Upon receiving the query $[f_k(a),\ f_k(b)]$, the cloud provider processes this query on the n data items $(d_1)_k, \cdots, (d_n)_k$ by checking which $f_k(d_j)$ $(1 \leq j \leq n)$ satisfies the condition $f_k(a) \leq f_k(d_j) \leq f_k(b)$. Based on the order preserving property of the function f_k, $d_j \in [a, b]$ if and only if $f_k(a) \leq f_k(d_j) \leq f_k(b)$. Thus, the cloud provider only needs to return all encrypted data items whose hash values f_k are in the range $[f_k(a),\ f_k(b)]$.

C. Analysis of Information Leakage

Given n data items $d_1, \cdots, d_n$ and a range query $[a, b]$, any precise privacy-preserving scheme should enable the cloud provider to find out all the data items that satisfy the query $[a, b]$ without revealing the values of the data items from the organization and the query from its customer. According to this requirement, we have the following theorem.

Theorem 4.1: Given any precise privacy-preserving scheme, if all possible results of range queries can be found during the query processing phase, the cloud provider can reveal the order of the original items.

Based on Theorem 4.1, our privacy-preserving scheme prevents the cloud provider from revealing any other information except the order of data items. for two reasons. First, the cloud provider cannot reveal the values of the data and queries from the hash results due to the one-way property of h_k. Second, the cloud provider cannot reveal these values by launching statistical analysis because for any two data items d_{j_1} and d_{j_2} $(1 \leq j_1 < j_2 \leq n)$, $(d_{j_1})_k \neq (d_{j_2})_k$ and $f_k(d_{j_1}) \neq f_k(d_{j_2})$. Recall that if some data items have the same value, the organization represents them as one data item with the number of items that share this value.

V. Integrity Preserving for 1-dimensional Data

We present the first probabilistic integrity-preserving scheme for 1-dimensional data, which allows a customer to verify the integrity of a query result with a high probability. The meaning of integrity preserving is two-fold. First, a customer can verify whether the cloud provider forges some data items in a query result. Second, a customer can verify whether the provider deletes data items that satisfies the query.

The basic idea of the integrity-preserving scheme is to encrypt neighborhood information for each data item such that the neighborhood information of the data items in a query result can be used to verify the integrity of the query result. More formally, let $(M(d_j))_k$ denote the encrypted neighborhood information for each data item d_j $(1 \leq j \leq n)$. To submit n data items $d_1, \cdots, d_n$ to a cloud provider, the organization not only sends the encrypted data items $(d_1)_k, \cdots, (d_n)_k$ and the encoded data items $f_k(d_1), \cdots, f_k(d_n)$, but also sends the encrypted neighborhood information $(M(d_1))_k, \cdots, (M(d_n))_k$. Upon receiving a query $[f_k(a), f_k(b)]$ from a customer, the cloud provider first finds out the query result based on the privacy-preserving scheme. Suppose that the data items $d_{j_1}, \cdots, d_{j_2}$ $(1 \leq j_1 \leq j_2 \leq n)$ satisfy the query. The cloud provider not only replies to the customer the query result $(d_{j_1})_k, \cdots, (d_{j_2})_k$, but also replies the encrypted neighborhood information $(M(d_{j_1}))_k, \cdots, (M(d_{j_2}))_k$. For ease of presentation, let QR denote the *query result*, which includes all the encrypted data items that satisfy the query, *i.e.*, $QR = \{(d_{j_1})_k, \cdots, (d_{j_2})_k\}$, and VO denote the *verification object*, which includes the information for the customer to verify the integrity of QR, *i.e.*, $VO = \{(M(d_{j_1}))_k, \cdots, (M(d_{j_2}))_k\}$. To verify the integrity, the customer first decrypts the query result and verification object, *i.e.*, computes $d_{j_1}, \cdots, d_{j_2}$ and $M(d_{j_1}), \cdots, M(d_{j_2})$. Second, the organization checks whether $d_{j_1}, \cdots, d_{j_2}$ satisfy the query and the overlapping parts of the neighborhood information from every two adjacent data items exactly match. If so, the customer concludes that the query result includes all the data items that satisfy the query. Otherwise, the customer concludes that some data items in the query result are forged or deleted by the cloud provider. Fig. 3 shows the basic idea of our integrity-preserving scheme.

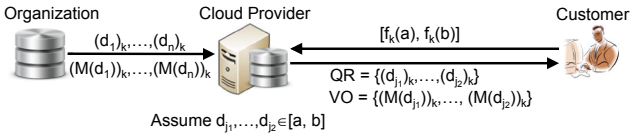

Fig. 3. Basic idea of integrity-preserving scheme

Our integrity-preserving scheme guarantees to detect the misbehavior of forging data items because the cloud provider cannot insert fake data items into a query result without knowing the secret key k. The scheme also allows a customer to detect the misbehavior of deleting data items in a query result with high probability. Furthermore, if the cloud provider conducts the deletion operation multiple times, the detection probability will increase significantly, *i.e.*, approach 100%.

Next we present new data structures, *bit matrices* and *local bit matrices*, and discuss their usage in integrity verification.

A. Bit Matrices and Local Bit Matrices

To define bit matrices and local bit matrices, we need to first partition the data domain into multiple non-overlapping buckets. For example in Fig. 4, we partition domain $[1, 15]$ to five buckets, $B_1 = [1, 3]$, $B_2 = [4, 6]$, $B_3 = [7, 10]$, $B_4 = [11, 12]$, and $B_5 = [13, 15]$. Second, we distribute the data items into the corresponding buckets. Third, we assign a bit value 1 to the buckets with data items, and assign a bit value 0 to the buckets without data items. Let $B(d_j)$ denote the bucket that includes d_j. A bucket is called the *left nonempty bucket* of data item d_j if the bucket is the left nearest bucket of $B(d_j)$ that includes data items. Similarly, a bucket is called the *right nonempty bucket* of data item d_j if the bucket is the right nearest bucket of $B(d_j)$ that includes data items. For

example, for data item 7 in Fig. 4, B_2 and B_5 are the left and right nonempty buckets of data item 7, respectively.

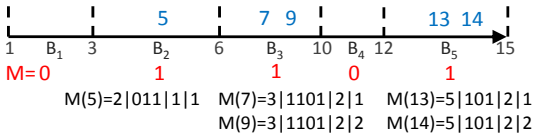

Fig. 4. Example bit matrix and local bit matrices

Based on the above concepts, we define bit matrices and local bit matrices as follows. The *bit matrix* of all data items, M, is formed by the bit values of all buckets. In Fig. 4, the bit matrix of the five data items is 01101, *i.e.*, $M = 01101$. The *local bit matrix* of a data item d_j, $M(d_j)$, consists of four parts: (1) the bucket id of $B(d_j)$; (2) a subset of the bit matrix, which is formed by the bit values from its left nonempty bucket to its right nonempty bucket; (3) the number of data items in bucket $B(d_j)$; (4) a distinct integer to distinguish the local bit matrix of d_j from other data items in bucket $B(d_j)$. In Fig. 4, the local bit matrix of data item 7 is $3|1101|2|1$, *i.e.*, $M(7) = 3|1101|2|1$, where 3 is the bucket id, 1101 is the subset of the bit matrix, 2 is the number of data items in bucket B_3, and 1 is the integer to distinguish $M(7)$ from $M(9)$. Intuitively, the bit matrix denotes the abstract information of all the data items, and the local bit matrix of a data item d_j denotes the abstract neighborhood information of d_j.

Note that the usage of bucket partition in this paper is different from that in previous work (*e.g.*, [13], [14], [4]). They leverage bucket partition to achieve privacy-preserving query processing, while we use the bit values of buckets for verifying the integrity of query results.

B. The Integrity-Preserving Scheme

This scheme includes four phases, *data submission*, *query submission*, *query processing*, and *query result verification*.

Data submission

Let $d_1, \cdots, d_n$ denote the data items of an attribute from the organization. The organization first partitions the data domain to m non-overlapping buckets $B_1, \cdots, B_m$, and then distributes $d_1, \cdots, d_n$ to these buckets. The bucket partition is a shared secret between the organization and its customers. Second, the organization computes the local bit matrix for each data item and then encrypts them with its secret key k, *i.e.*, computes $(M(d_1))_k, \cdots, (M(d_n))_k$. Third, the organization sends to the cloud provider the encrypted local bit matrices $(M(d_1))_k, \cdots, (M(d_n))_k$ as well as encrypted data items $(d_1)_k, \cdots, (d_n)_k$ and the encoded data items $f_k(d_1), \cdots, f_k(d_n)$.

Query submission

To perform a range query $[a, b]$, a customer sends $[f_k(a), f_k(b)]$ as the query to the cloud provider.

Query Processing

Upon receiving $[f_k(a), f_k(b)]$, the cloud provider computes QR as in Section IV-B. Here we consider how to compute VO. If $QR = \{(d_{j_1})_k, \cdots, (d_{j_2})_k\}$ $(1 \le j_1 \le j_2 \le n)$, $VO = \{(M(d_{j_1}))_k, \cdots, (M(d_{j_2}))_k\}$; if $QR = \emptyset$, which means that there is a data item d_{j_1} $(1 \le j_1 \le n)$ such that $d_{j_1} < a \le b < d_{j_1+1}$, then $VO = \{(M(d_{j_1}))_k, (M(d_{j_1+1}))_k\}$. Finally, the cloud provider replies QR and VO.

Query Result Verification

Upon receiving the query result QR and the verification object VO, the customer decrypts them, and then verifies the integrity of QR as follows. First, the customer verifies whether each item in QR satisfies the query $[a, b]$. Second, the customer verifies whether the cloud provider deletes any data item that satisfies the query. Let $\{(d_{j_1})_k, \cdots, (d_{j_2})_k\}$ be the correct query result and $B_{g_1}, \cdots, B_{g_t}$ be the buckets which include at least one data item in the query result. Let QR be the query result from the cloud provider. Suppose the cloud provider deletes a data item $(d_j)_k$ that satisfies the query, *i.e.*, $(d_j)_k \in \{(d_{j_1})_k, \cdots, (d_{j_2})_k\}$, and $d_j \in B_{g_s}$ $(1 \le s \le t)$. We consider the following four cases.

***Case* 1**: When $QR \neq \emptyset$, if $B_{g_s} \subseteq [a, b]$, the deletion of $(d_j)_k$ can be detected for two reasons. First, if B_{g_s} only includes one data item d_j, deleting $(d_j)_k$ can be detected because the local bit matrices of data items in $B_{g_{s-1}}$ or $B_{g_{s+1}}$ show that B_{g_s} should include at least one data item. Second, if B_{g_s} includes multiple data items, deleting $(d_j)_k$ can be detected because the local bit matrices of other data items in B_{g_s} have the number of data items in B_{g_s}. In Fig. 4, given a range query [4,11], the correct query result is $\{(5)_k, (7)_k, (9)_k\}$, and the verification object is $\{(M(5))_k, (M(7))_k, (M(9))_k\}$. Deleting $(7)_k$ in B_3 can be detected because based on $M(9)$, the customer knows that B_3 includes two data items.

***Case* 2**: When $QR \neq \emptyset$, if $B_{g_s} \not\subseteq [a, b]$, deleting $(d_j)_k$ cannot be detected because the customer does not know whether $B_{g_s} \cap [a, b]$ includes data items. Considering the example in Case 1, deleting $(5)_k$ cannot be detected because the customer does not know whether $B_2 \cap [4, 11]$ includes data items.

***Case* 3**: When $QR = \emptyset$, if $B(d_{j_1}) \cap [a, b] = \emptyset$ and $B(d_{j_1+1}) \cap [a, b] = \emptyset$, the deletion of $(d_j)_k$ can be detected because $M(d_{j_1})$ or $M(d_{j_1+1})$ shows that bucket B_{g_s} between $B(d_{j_1})$ and $B(d_{j_1+1})$ includes data items, and hence, condition $d_{j_1} < a \le b < d_{j_1+1}$ does not hold. In Fig. 4, given a range query [3,5], the correct query result is $\{(5)_k\}$. If the cloud provider replies $QR = \emptyset$ and $VO = \{(M(7))_k\}$, deleting $(5)_k$ can be detected because the customer knows that B_2 includes data items based on $M(7)$, and these data items are closer to the query [3,5] than 7.

***Case* 4**: When $QR = \emptyset$, if $B(d_{j_1}) \cap [a, b] \neq \emptyset$ or $B(d_{j_1+1}) \cap [a, b] \neq \emptyset$, the deletion of $(d_j)_k$ cannot be detected because the customer does not know whether $B(d_{j_1}) \cap [a, b]$ or $B(d_{j_1+1}) \cap [a, b]$ includes data items. In Fig. 4, given a range query [9,12], the correct query result is $\{(9)_k\}$. If the cloud provider replies $QR = \emptyset$ and $VO = \{(M(7))_k, (M(13))_k\}$, deleting $(9)_k$ cannot be detected because the customer does not know whether $B(3) \cap [9, 12]$ includes data items.

The cloud provider can break integrity verification if and only if it can distinguish Cases 2 and 4 from other two cases. Distinguishing these two cases is equivalent to knowing which data items belong to the same bucket. However, such information cannot be revealed by analyzing the encrypted data items $(d_1)_k, \cdots, (d_n)_k$, the encoded data items

$f_k(d_1), \cdots, f_k(d_n)$, and the encrypted local bit matrices $(M(d_1))_k, \cdots, (M(d_n))_k$. Because for any two data items d_{j_1} and d_{j_2} $(1 \leq j_1 < j_2 \leq n)$, $(d_{j_1})_k \neq (d_{j_2})_k$, $f_k(d_{j_1}) \neq f_k(d_{j_2})$, and $(M(d_{j_1}))_k \neq (M(d_{j_2}))_k$. Thus, the cloud provider can only randomly delete the data items in the query result and hope that such deletion will not be detected.

VI. Query Over Multi-dimensional Data

A. Privacy for Multi-dimensional Data

The data of organizations and the queries of customers are typically multi-dimensional. For example, a medical record typically includes patient's name, birthday, age, *etc.* A z-dimensional data item D is a z-tuple $(d^1, \cdots, d^z)$ where each d^r $(1 \leq r \leq z)$ is the value for the r-th dimension (*i.e.*, attribute). A z-dimensional range query consists of z sub-queries $[a^1, b^1], \cdots, [a^z, b^z]$ where each sub-query $[a^r, b^r]$ $(1 \leq r \leq z)$ is a range over the r-th dimension.

We extend our privacy-preserving scheme for one-dimensional data to multi-dimensional data as follows. Let $D_1, \cdots, D_n$ denote n z-dimensional data items, where $D_j = (d_j^1, \cdots, d_j^z)$ $(1 \leq j \leq n)$. First, the organization encrypts these data with its secret key k, *i.e.*, computes $(D_1)_k, \cdots, (D_n)_k$. Second, for each dimension r, it applies our order-preserving hash-based function f_{k_r}, *i.e.*, computes $f_{k_r}(d_1^r), \cdots, f_{k_r}(d_n^r)$, where k_r is the secret key of the order-preserving hash-based function for the r-th dimension. Last, it sends the encrypted data items $(D_1)_k, \cdots, (D_n)_k$, and $f_{k_1}(d_1^1), \cdots, f_{k_1}(d_n^1)$, $\cdots$, $f_{k_z}(d_1^z), \cdots, f_{k_z}(d_n^z)$ to the cloud provider. When a customer wants to perform a query $([a^1, b^1], \cdots, [a^z, b^z])$, it applies the order-preserving hash-based function f_{k_r} on the lower and upper bounds of each sub-query $[a^r, b^r]$ and sends $[f_{k_1}(a^1), f_{k_1}(b^1)], \cdots, [f_{k_z}(a^z), f_{k_z}(b^z)]$ to the cloud provider. The cloud provider then compares $f_{k_r}(d_1^r), \cdots, f_{k_r}(d_n^r)$ with $[f_{k_r}(a^r), f_{k_r}(b^r)]$ for each dimension r, to find out the query result QR. Considering 5 two-dimensional data items (1,11), (3,5), (6,8), (7,1) and (9,4), given a range query ([2,7],[3,8]), the query result QR is $\{(3,5)_k, (6,8)_k\}$.

To prevent the attack of statistical analysis, the data sent from the organization to the cloud provider should satisfy the following two conditions. First, for any $1 \leq j_1 \neq j_2 \leq n$, $(D_{j_1})_k \neq (D_{j_2})_k$. To satisfy this condition, if multiple data items have the same value for each dimension, the organization can simply represent them as one data item annotated with the number of these items. Second, along each dimension r, for any $1 \leq j_1 \neq j_2 \leq n$, $f_{k_r}(d_{j_1}^r) \neq f_{k_r}(d_{j_2}^r)$. To satisfy this condition, the organization needs to revise the data items with the same value for the dimension r. Recall the arbitrary order-preserving property of f_{k_r}. It allows the organization to arbitrarily revise data items while still preserving the order of these items in the hash results. In our context, if $d_{j_1}^r = d_{j_2}^r$, the organization can concatenate a distinct number for each of them, *i.e.*, $d_{j_1}^r|0$ and $d_{j_2}^r|1$, and then apply the hash-based function f_{k_r}.

B. Integrity for Multi-dimensional Data

To preserve the integrity of multi-dimensional data, the organization builds multi-dimensional local bit matrices.

We first present the data structures, multi-dimensional bit matrices and local bit matrices, and then discuss the usage in integrity verification for multi-dimensional data. Considering the example in Fig. 5(a), we partition the data domain into $4 \times 6 = 24$ buckets. Then, we distribute the data items, $D_1, \cdots, D_5$, into the corresponding buckets. We assign a bit value 1 or 0 to each bucket to indicate whether the bucket includes data items or not. Let $B(d_j)$ denote the bucket that includes d_j. A bucket is called the *r-th left nonempty bucket* of data item d_j $(1 \leq r \leq z)$ if the bucket is the left nearest bucket of $B(d_j)$ that includes data items for the r-th dimension. Similarly, a bucket is called the *r-th right nonempty bucket* of data item d_j if the bucket is the right nearest bucket of $B(d_j)$ that includes data items for the r-th dimension. In Fig. 5(a), $B(D_2)$ is the 1-th left nonempty bucket of data item D_3.

Based on the above concepts, we define bit matrices and local bit matrices as follows. The *bit matrix* of all data items, M, is formed by the bit values of all buckets. In Fig. 5(a), the bit matrix of the five data items

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The *local bit matrix* of a data item D_j, $M(D_j)$, consists of four parts: (1) the bucket id of $B(D_j)$; (2) a subset of the bit matrix, which is formed by the bit values of the buckets within a rectangle, which includes its left and right nonempty buckets for each dimension; (3) the number of data items in bucket $B(D_j)$; (4) a distinct integer to distinguish the local bit matrix of D_j from other data items in bucket $B(D_j)$. In Fig. 5(b), the local bit matrix of D_3 is

$$M(D_3) = ID| \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} |1|1$$

where ID is the bucket id of $B(D_3)$.

The integrity-preserving scheme for z-dimensional data $(z > 1)$ is similar as that for 1-dimensional data. Here we only show an example. Consider the five data items $D_1{:}(d_1^1, d_1^2), \cdots, D_5{:}(d_5^1, d_5^2)$ in Fig. 5. The organization sends to the cloud provider the encrypted data items $(D_1)_k, \cdots, (D_5)_k$, encrypted local bit matrices $(M(D_1))_k, \cdots, (M(D_5))_k$, and the encoded data items $f_{k_1}(d_1^1), \cdots, f_{k_1}(d_5^1)$, $f_{k_2}(d_1^2), \cdots, f_{k_2}(d_5^2)$. Given a range query that includes two data items D_2 and D_3 in Fig. 5(c), the cloud provider replies to the customer the query result $QR = \{(D_2)_k, (D_3)_k\}$ and the verification object $VO = \{(M(D_2))_k, (M(D_3))_k\}$.

Next, we analyze the detection probability for multi-dimensional data. Let $B_1, \cdots, B_m$ denote the multiple non-overlapping buckets, and $([l_i^1, h_i^1], \cdots, [l_i^z, h_i^z])$ denote a z-dimensional bucket B_i $(1 \leq i \leq m)$. A bucket B_i is called a

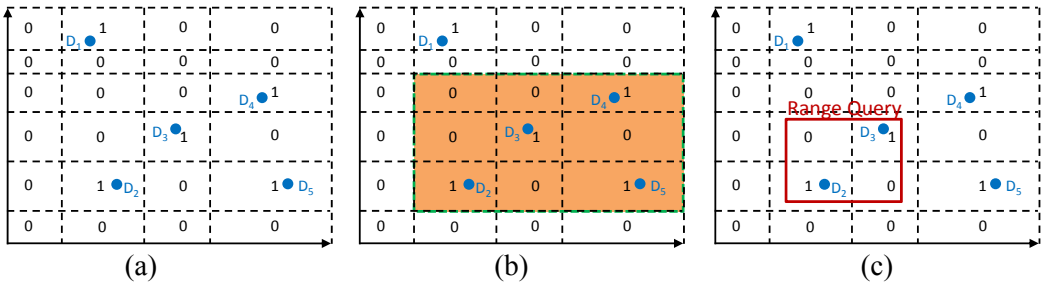

Fig. 5. The example 2-dimensional bit matrix and local bit matrices

single-value bucket if for each dimension r ($1 \leq r \leq z$), $l_i^r = h_i^r$. Let $[x_1^r, x_{N_r}^r]$ denote the domain for each dimension r. Let $e(X)$ denote the frequency of the data item $X : (x^1, \cdots, x^z)$. The detection probability of a deletion operation by cloud providers can be computed as

$$Pr = \frac{\sum_{i=1}^{m} \prod_{r=1}^{z} (l_i^r - x_1^r + 1)(x_{N_r}^r - h_i^r + 1) \sum_{X \in B_i} e(X)}{\sum_{j=1}^{n} \prod_{r=1}^{z} (d_j^r - x_1^r + 1)(x_{N_r}^r - d_j^r + 1)} \tag{2}$$

Theorem 6.1: Given any n z-dimensional data items $D_1, \cdots, D_n$, the maximum detection probability of a deletion operation is $Pr_{max} = 100\%$ if and only if each data item D_j ($1 \leq j \leq n$) forms a single-value bucket, *i.e.*, $([d_j^1, d_j^1], \cdots, [d_j^z, d_j^z])$.

Theorem 6.2: Given n z-dimensional data items $D_1, \cdots, D_n$, the minimum detection probability of a deletion operation is

$$Pr_{min} = \frac{n}{\sum_{j=1}^{n} \prod_{r=1}^{z} (d_j^r - x_1^r + 1)(x_{N_r}^r - d_j^r + 1)} \tag{3}$$

if and only if there is only one bucket $([x_1^1, x_{N_1}^1], \cdots, [x_1^z, x_{N_z}^z])$.

The calculation of the detection probability in Equation 2 and the proofs of Theorems 6.1 and 6.2 are similar to the 1-dimensional case. Finding optimal bucket partition for multi-dimensional data is an interesting yet difficult problem and will be discussed in further work.

VII. Evaluation

We evaluated the efficiency and effectiveness of our privacy and integrity preserving scheme for both 1-dimensional and multi-dimensional data. In terms of efficiency, we measured the data processing time for organizations, and the space cost and query processing time for cloud providers. In terms of effectiveness, we measured whether the experimental detection probability of deletion operations by cloud providers is consistent with the theoretical analysis in VI-B. Our experiments were implemented in Java 1.6.0 and carried out on a PC running Linux with 2 Intel Xeon cores and 16GB of memory.

A. Evaluation Setup

We conducted our experiments on a real data set, Adult, and five synthetic datasets. The Adult dataset is from the UCI Machine Learning Repository [12] and has been widely used in previous studies. It contains 45222 records. We chose three attributes in this dataset, Age, Education, and Hours-per-week. Note that Education is a categorical attribute and we mapped each Education value to a distinct integer. The domains of these three attributes are $[17, 90]$, $[1, 16]$, and $[1, 99]$, respectively. The five synthetic datasets are generated by randomly choosing $10^2, 10^3, \cdots, 10^6$ data items from five domains $[0, 10^3], [0, 10^4], \cdots, [0, 10^7]$, respectively. For our order-preserving hash-based function f_k, we used HMAC-MD5 with 128-bit keys as the basic hash function h_k. We used the DES encryption algorithm to encrypt both data items and local bit matrices.

B. Results for 1-dimensional Data

We employed the synthetic datasets to evaluate the efficiency and effectiveness of our scheme for 1-dimensional data. For each synthetic dataset, given different number of buckets, we first computed the optimal partition and the maximum detection probability, and then we implemented our scheme using the optimal partition. We also generated 1,000 random range queries to measure the total processing time for cloud providers and verify query result integrity for customers. To process the query, we used the binary search algorithm to find out the query result. Let n denote the number of data items in a dataset and m denote the given number of buckets. Note that if all data items form single-value buckets, the detection probability is 100%. The rest buckets are empty buckets and the number of these buckets is $n + 1$. Thus, the total number of buckets can be computed as $2n + 1$. In other words, given $m = 2n + 1$, the output of our optimal algorithm should be these $2n + 1$ buckets. Based on this observation, we define *partition ratio* as $m/(2n+1)$. The partition ratio helped us to normalize the results of our optimal partition algorithm for different datasets. Fig. 6 shows the normalized results for the five synthetic datasets. We observed that the detection probability increases with the partition ratio and if partition ratio is equal to 1, *i.e.*, $m = 2n + 1$, the probability becomes 1, which confirms the above discussion.

To check whether the experimental detection probability is consistent with the theoretical analysis, for each dataset, we randomly deleted a data item in each query result and then computed the percentage of query results that were detected by our integrity-preserving scheme. Note that this percentage is the experimental detection probability. Fig. 7 shows that the

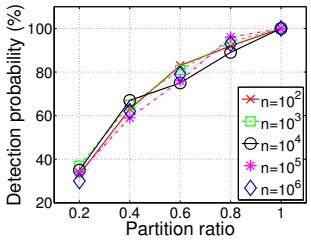

Fig. 6. Effectiveness of optimal partition algorithm

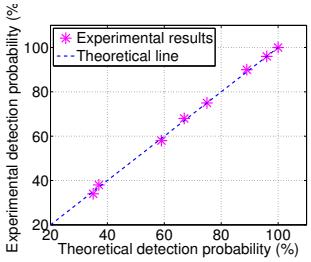

Fig. 7. Correctness of integrity-preserving scheme

experimental detection probability is close to the theoretical line, which demonstrates the correctness of our analysis.

Figures 8 and 9 show the data processing time and space cost for the five synthetic datasets, respectively. Note that the horizonal and vertical axes in these figures are in logarithmic scales. In Fig. 8, we observed that the data processing time is less than 300 seconds for 10^5 data items. Although for one million data items, the data processing time is about 50 minutes, which is reasonable for real applications because the data processing is a one-time offline procedure. In Fig. 9, we observed that the space cost grows linearly with the number of data items in a dataset. A cloud provider needs 33MB to store one million data items from an organization.

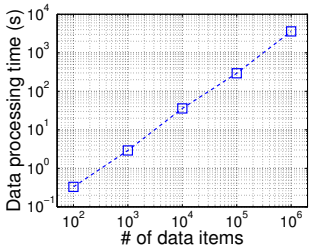

Fig. 8. Data processing time

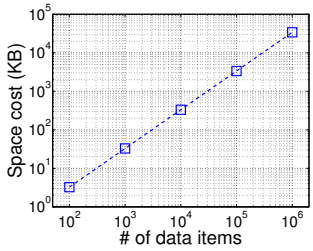

Fig. 9. Space cost

Fig. 10 shows the total processing time of 1,000 queries for the five synthetic datasets. Processing 1,000 queries over one million data items only takes 2 seconds.

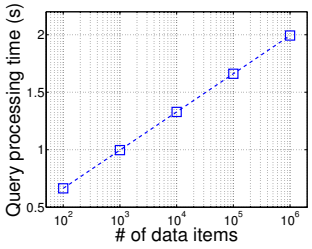

Fig. 10. Query processing time

C. Results for Multi-dimensional Data

We employed the Adult dataset to evaluate the efficiency and effectiveness of our scheme for multi-dimensional data. The experimental results show that the data processing time for this dataset is 104 seconds, the space cost is 1.5MB, and the total processing time of 1,000 random queries is 3.5 seconds. Due to the absence of the optimal partition algorithm for multi-dimensional data, we arbitrarily partitioned the Adult dataset to different sets of buckets. The results show that the experimental detection probability is consistent with the theoretical analysis for multi-dimensional range queries.

VIII. Conclusion

We propose a privacy and integrity preserving scheme for multi-dimensional range queries in cloud computing. To preserve privacy, we propose an order-preserving hash-based function to encode the data from an organization and the queries from its customers such that a cloud provider can process encoded queries over encoded data without knowing the actual values. To preserve integrity, we propose the first probabilistic integrity-preserving scheme for range queries. This scheme employs a new data structure, local bit matrices, which allows customers to verify query result integrity with high probability. We conducted extensive analysis and evaluation. The results demonstrate effectiveness and efficiency of our scheme. Our future work will consider data updating and optimal bucket partitioning for multi-dimensional data.

References

[1] Amazon web services, aws.amazon.com.
[2] Google app engine, code.google.com/appengine.
[3] Microsoft azure, www.microsoft.com/azure.
[4] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, Order preserving encryption for numeric data, in *Proc. SIGMOD*, 2004, pp. 563–574.
[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, Order-preserving symmetric encryption, in *Proc. EUROCRYPT*, 2009, pp. 224–241.
[6] D. Boneh and B. Waters, Conjunctive, subset, and range queries on encrypted data, in *Proc. TCC*, 2007, pp. 535–554.
[7] H. Chen, X. Man, W. Hsu, N. Li, and Q. Wang, Access control friendly query verification for outsourced data publishing, in *Proc. ESORICS*, 2008, pp. 177–191.
[8] W. Cheng, H. Pang, and K.-L. Tan, Authenticating multi-dimensional query results in data publishing, in *Proc. DBSec*, 2006, pp. 60–73.
[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press.
[10] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, Authentic data publication over the internet, *Journal of Computer Security*, vol. 11, no. 3, pp. 291–314, 2003.
[11] E. A. Fox, Q. F. Chen, A. M. Daoud, and L. S. Heath, Order-preserving minimal perfect hash functions and information retrieval, *ACM Transactions on Information Systems*, vol. 9, pp. 281–308, 1991.
[12] Frank, A., and Asuncion, A. UCI machine learning repository, 2010.
[13] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, Executing sql over encrypted data in the database-service-provider model, in *Proc. SIGMOD*, 2002, pp. 216–227.
[14] B. Hore, S. Mehrotra, and G. Tsudik, A privacy-preserving index for range queries, in *Proc. VLDB*, 2004, pp. 720–731.
[15] R. Merkle, Protocols for public key cryptosystems, in *Proc. IEEE S&P*, 1980, pp. 122–134.
[16] M. Narasimha and G. Tsudik, Authentication of outsourced databases using signature aggregation and chaining, in *Proc. DASFAA*, 2006.
[17] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan, Verifying completeness of relational query results in data publishing, in *Proc. SIGMOD*, 2005, pp. 407–418.
[18] H. Pang and K.-L. Tan, Authenticating query results in edge computing, in *Proc. ICDE*, 2004, pp. 560–571.
[19] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, Multi-dimensional range query over encrypted data, in *Proc. IEEE S&P*, 2007, pp. 350–364.
[20] C. Curino, E.-P. C. Jones, R.-A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, Relational Cloud: A Database-as-a-Service for the Cloud, in *Proc. CIDR*, 2011.
[21] R.-A. Popa, N. Zeldovich, and H. Balakrishnan. CryptDB: A practical encrypted relational DBMS, *Technical Report MIT-CSAIL-TR-2011-005*, 2011.
[22] A. Boldyreva, N. Chenette, and A. O'Neill, Order-preserving encryption revisited: improved security analysis and alternative solutions, in *Proc. CRYPTO*, 2011, pp. 578–595.
[23] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, Authenticated indexing for outsourced spatial databases, *VLDB Journal*, vol. 18, no. 3 pp. 631–648, 2009
[24] L. Hu, W. Ku, S. Bakiras, C. Shahabi, Verifying spatial queries using Voronoi neighbors, in *Proc. GIS*, 2010, pp. 350–359.
[25] F. Chen and A. Liu, SafeQ: secure and efficient query processing in sensor networks, in *Proc. INFOCOM*, 2010, pp. 1-9.