**Noname manuscript No.**
(will be inserted by the editor)

# Authentication of Outsourced Linear Function Query with Efficient Updates

**Gang Sheng · Chunming Tang · Wei Gao · Yunlu Cai · Xing Hu**

Received: date / Accepted: date

**Abstract** Storing the large-scale data on the cloud server side becomes nowadays an alternative for the data owner with the popularity and maturity of the cloud computing technique, where the data owner can manage the data with limited resources, and the user issues the query request to the cloud server instead of the data owner. As the server is not completely trusted, it is necessary for the user to perform results authentication to check whether or not the returned results from the cloud server are correct. We investigate in this paper how to perform efficient data update for the result authentication of the outsourced univariate linear function query. We seek to outsource almost

This work was supported in part by the National Natural Science Foundation of China (No.11271003, 11271226), Guangdong Province Natural Science Foundation of major basic research and Cultivation Project (No.2015A030308016), Basic Research Major Projects of Department of Education Guangdong Province (No.2014KZDXM044), Innovation Team Construction Project of Guangdong Province Regular Universities (No.2015KCXTD014), Project of Department of Education of Hunan Province(No. 15C0536), and Collaborative Innovation Major Projects of Bureau of Education of Guangzhou City (No.1201610005).

Gang Sheng
College of Mathematics and Information Science, Guangzhou University, Guangzhou, China
E-mail: shenggang@neusoft.edu.cn

Chunming Tang
College of Mathematics and Information Science, Guangzhou University, Guangzhou, China
E-mail: ctang@gzhu.edu.cn

Wei Gao
School of Mathematics and Statistics Science, Ludong University, Yantai, China
E-mail: mygaowei@163.com

Yunlu Cai
College of Mathematics and Information Science, Guangzhou University, Guangzhou, China
E-mail: yunlucai@gzhu.edu.cn

Xing Hu
School of Mathematics and Computational Science, Hunan University of Science and Technology, Changsha, China
E-mail: huxing@gzhu.edu.cn

all the data and computing to the server, and as few data and computations as possible are stored and performed on the data owner side, respectively. We present a novel scheme to achieve the security goal, which is divided into two parts. The first part is a verification algorithm for the outsourced computing of line intersections, which enables the data owner to store most of the data on the server side, and to execute less of the computing of the line intersections. The second part is an authentication data structure Two Level Merkle B Tree for the outsourced univariate linear function query, where the top level is used to index the user input and authenticate the query results, and the bottom level is used to index the query condition and authenticate the query results. The authentication data structure enables the data owner to update the data efficiently, and to implement the query on the server side. The theoretic analysis shows that our proposed scheme works with higher efficiency.

**Keywords** Cloud Computing · Function Query · Correctness Verification

## 1 Introduction

Function query is a kind of widely used query with simple form, which is often performed on the large-scale data in the current era of the big data. Large high-end resources are needed for massive computing and storage, which is a great burden if the resources are employed completely by the data owner himself. The cloud computing technique is designed to provide the user with pay-as-you-go services, by which the data owner can manage the large-scale data with the equipment with limited resources, such as a computer, even a mobile terminal. Thus, the data owner can focus on the core business, and does not need to pay high costs for configuring the high performance hardware.

As a result of the loss of direct physical control of the data, some serious security issues arise with the adoption of the cloud service by the data owner, such as data leakage, data privacy disclosure, incorrect results returned from the server, and so on. The data security problem is one of the key problems in the application and research field of the cloud computing. Only when the security issues are properly solved can the data owner eliminate their concerns. The data owner then would like to adopt the disclosure services of the cloud computing, which will in turn promote the development and popularity of the cloud computing.

Yang et al. proposed to outsource the function query on large-scale dataset to the cloud server, and mainly aimed at the security issue of results authentication [20]. As the results of function query are sorted by the calculation results with the user input, then the calculation results appear in different order with different user input. Such traditional authentication techniques as the Merkle Hash Tree [10] and the signature chain [11] are based on the numerical order of the authenticated data, which cannot be applied directly to authentication the results of function query. Thus, Yang et al. presented a novel segment-sort strategy, where the range of the user input is segmented into many intervals, the functions are sorted by the calculation results, and

the signature mesh technique is applied for every interval. The data and the signatures are stored together on the server side for user query request. The server executes the query issued by the user, and returns the obtained query results and the relative verification objects to the user. The user can verify the correctness of the results with the verification objects. No matter the returned results are fabricated or omitted can be detected by the user with public key.

However, in light of our observation, only the security issue of the result verification of the outsourced function query is studied in [20], and the proposed scheme has three limitations. The first is introduced by the adoption of the digital signature in the signature mesh method. When the dataset is updated, such as adding and/or deleting data, a large number of digital signature operations are performed by the data owner. The data owner is generally equipped with limited resources and the digital signature itself is a computationally expensive operation, which leads to the inefficiency in the practical running time. The second is that the dataset and the obtained signatures are stored together on the data owner side although the query on the dataset is delegated to the cloud server. The data owner must pay a certain price to manage the dataset and the signatures locally as usual, which implies the outsourcing is not complete. The third is that the query execution on the server side is not taken into consideration. The focus of [20] is the result authentication and no algorithm for the function query is given. The cloud server has to select or design the corresponding algorithm to execute the user query.

The goal of this paper is to overcome the limitations mentioned above, where as few data as possible are stored and as little computing as possible is performed on the data owner side. The data owner can carry out efficient update for the outsourced function query. The security problems then arise, that is, how the data owner can authenticate the data update performed on the cloud server side, and how the user can authenticate the query results returned from the cloud server. We propose a novel scheme to resolve the two problems, which is divided into two steps. In the first step, we use the random check method to verify the correctness of the intersections computed by the cloud server. In the second step, we then present an authentication data structure for the query results authentication. The main contributions of this paper are summarized as follows.

- We introduce the problem of executing efficient data update by the data owner for the outsourced function query.
- We present a random check algorithm for the data owner to verify the correctness the returned line intersections, by which only small amount of data are stored on the data owner side.
- We introduce a novel authentication data structure, by which both the query execution and the efficient data update are implemented on the server side simultaneously.
- We give the performance analysis, which show the running efficiency of our proposed scheme.

The rest of the paper is organized as follows. We give the preliminaries in Section 2, including the system model, the definition of the function query and the authentication data structure. In Section 3, we present a novel authentication data structure Two Level Merkle B+ Tree for results verification of the outsourced query function. We propose a scheme for efficient data update of outsourced query function in Section 4. We give the performance analysis in Section 5. We review the related works in Section 6. Finally, we give the conclusions in Section 7.

## 2 Preliminaries

### 2.1 System model

The adopted framework in this paper consists of three parties, the data owner (DO), the cloud server (CS) and the user, which is shown in Fig.1.

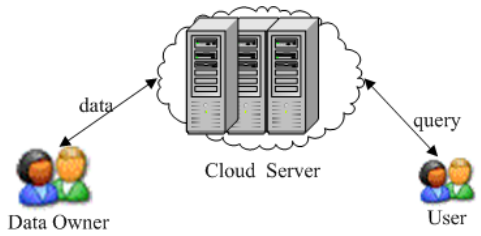


**Fig. 1** System Model.

The CS possesses abundant resources for computing and data storage, and performs the request issued by the DO and/or the user. The DO has a large-scale dataset $F$ for function query. But the DO is equipped with limited resources and has no ability to manage the dataset properly. Then the DO outsources $F$ to the CS, and performs data update on the outsourced dataset through the Internet. The user issues query on the outsourced dataset stored on the CS side. We assume that the data owner and the user are both honest, and the cloud server is semi-honest. The CS may return incorrect query results to the user. Thus, the user should have the ability to check whether or not the returned results are correct. We study the problem of results authentication of outsourced function query with efficient data update.

### 2.2 Function Query

The function query falls into three types according to the number and the degree of the involved variables, i.e., the univariate linear function, the multi-

variate linear function and the multivariate high degree function. We mainly study the univariate linear function in this paper, which has only a single variable with degree as one. So the function query means just the univariate linear function in the rest of the paper when there is no confusion.

We now explain how the function query works. Let

$$F = \{f_i = (a_i, b_i) |\ a_i, b_i \in R,\ i = 1, \ldots, |F|\} \tag{1}$$

be a dataset, where the tuple $(a_i, b_i)$ is the necessary parameters of the univariate linear function, and $|F|$ is the size of the dataset. Let $x \in R$ be the user input. With $x$ and $F$, we here define a dataset

$$C = \{c_i = a_i * x + b_i | i = 1, \ldots, |F|\}, \tag{2}$$

where $c_i$ is computed by the user input $x$ and every tuple $(a_i, b_i)$. The dataset $C$ is just given as a symbol here for easy understanding of the function query definition, and not every element $c_i$ of $C$ is to be computed in practice for the purpose of improving the running efficiency when performing the function query. Function query is divided into three classes according to the given different query condition by the user, which are shown as follows.

- $Top - K(x, k)$ returns the maximum $k$ results from the dataset $C$, where $k$ is a positive integer.
- $kNN(x, k, y_0)$ returns the $k$ nearest results with $y_0$ from the dataset $C$, where $k$ is a positive integer and $y_0 \in R$.
- $Range(x, l, u)$ returns all the results that is between $l$ and $u$ from the dataset $C$, where $l, u \in R$.

### 2.3 Authentication Data Structure

Li et al. proposed an authentication data structure, named MB Tree, to authenticate the query results of the outsourced relational database [7]. MB Tree is combined by B+ Tree and the kernel of Merkle Hash Tree(MT Tree) [10], where an item is inserted into every entry of the B+ tree node to store the hash value, which is shown in Fig.2, where $k_i$ is the key, $p_i$ is the pointer, and $h_i$ is just the newly added hash value. If the node is a leaf node, the item $p_i$ refers to a record $r_i$, and the item $h_i$ is the hash of the referred record, i.e., $h_i = H(r_i)$. If the node is a non-leaf node, the item $p_i$ refers to a child node, and the item $h_i$ is the hash of the catenation of the referred child nodes. The function $H$ here is a one way hash function, which is usually adopted as MD5 or SHA. The hash value $h_i$ of every entry of every node is computed recursively until the tree root is met. DO uses the secret key to sign the root of the MB Tree with a digital signature scheme to get $sig(root)$. The update of the MB Tree obeys the rules of updating the B+ Tree. When a new record is inserted into the MB Tree, a record is modified or deleted, the items key, pointer and hash value in the corresponding leaf node are updated accordingly. Then the related items in the parent node needs to be updated. The update proceeds

recursively until the root of the MB tree is updated. DO signs the root of MB Tree to get the current $sig(root)$.

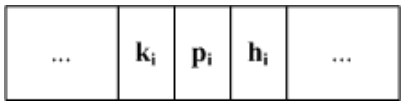


**Fig. 2** MB Tree node

DO sends the data and $sig(root)$ to CS, which constructs or updates the MB Tree for the data according to the rules mentioned above. It only differs in that CS doesn't sign the root of the MB Tree on the CS side. On the receipt of the query request from the user, DO performs the query on the MB Tree to get the query results $R$, and records the related data and the hash values, which are called the Verification Objects ($VO$). $R$, $VO$ and $sig(root)$ are returned together to the user. With $R$ and $VO$, the user reconstructs the MB Tree to get the root $root'$, and apply the digital signature technique with the public key to check if $root'$ matches $sig(root)$. If the result is yes, the results $R$ are deemed to be correct; otherwise, $R$ are incorrect. Thus, the security goal of result correctness verification can be achieved.

## 3 Two Level Merkle B+ Tree

In this section, we present a novel authentication data structure, named Two Level Merkle B+ Tree (TLMB Tree), to achieve the goals of query execution and query results authentication simultaneously, by which the amount of digital signature is reduced significantly, and the running efficiency of data update is increased accordingly. We now mainly introduce the details of the TLMB Tree.

### 3.1 Structure Details

The TLMB Tree is divided into two levels, i.e., the bottom level and the top level. The bottom level is a set of MB Trees, denoted as $MB_B$, which consists of a lot of MB Trees. Then, we have $MB_B = \{MB_i | i = 1, \ldots, |MB_B|\}$, where $MB_i$ is the $i$th MB Tree, and $|MB_B|$ is the size of the set $MB_B$. The entry value of every leaf node of $MB_i$ is the stored data and the hash value, whose construction and data update are performed according to the rules of MB Tree in 2.3. It only differs in that the data owner does not sign the root of $MB_i$.

The top level is also a MB Tree, denoted as $MB_T$, whose entry value of every leaf node is just the hash value of the root of the bottom level MB Tree. $MB_T$ is then constructed by the rules of MB Tree. The root $MB_T$ is signed by the data owner. We give a diagram in Fig.3, where the fan-out of the trees $MB_T$ and $MB_i$ are all set to be 3.

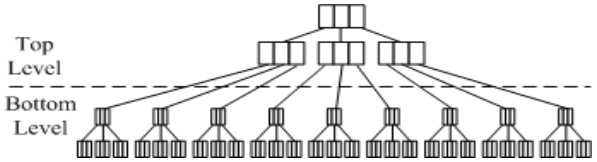


**Fig. 3** Two Level Merkle B+ Tree.

We utilize the TLMB Tree to execute the user query and verify the result correctness of the outsourced function query. We adopt the segment-sort strategy by Yang et al. [20]. Every tuple $(a_i, b_i) \in F$ is modeled as a line, denoted as $l_i$, in the two-dimensional space. Then there exist also $|F|$ lines for the dataset $F$. The linear equation for $l_i$ with the user input $x$ is given as

$$l_i \ : \ y_i = a_i * x + b_i. \tag{3}$$

Suppose $I$ be the set of the line intersections, where

$$I = \{x_i | i = 1, \ldots, |I|\}, \tag{4}$$

where $I$ is the size of the set $I$, and there exist $k$ lines $l_1, \ldots, l_k$, such that the equations $y_1 = \ldots = y_k$ holds with $k > 1$ as a positive integer.

The domain of the query input $x$ is divided into a lot of intervals according to the intersections of the lines, i.e., $(-\infty, x_1], \ldots, (x_i, x_{i+1}], \ldots, (x_I, +\infty)$. The lines are sorted according to $y_i$ in each interval. The interval $(-\infty, x_1]$ can be rewritten as $(x_0, x_1]$ and $(x_I, +\infty)$ as $(x_I, x_{I+1})$ for symbol consistency, where $-\infty$ is replaced with $x_0$ and $+\infty$ with $x_{I+1}$.

We construct a MB Tree for every interval in the bottom level, and a lot of MB Trees are thus constructed. A single MB Tree is constructed in the top level, where every entry in the leaf node refers to a MB Tree in the bottom level. The TLMB Tree is null in the initial, and is constructed by updating data repeatedly.

### 3.2 Data Update

We mainly introduce the data update of TLMB Tree, including the data adding and deleting, and the data modifying can be achieved by deleting first and adding then, which will not be discussed here.

*3.2.1 Data Adding*

A tuple is modeled as a line for the outsourced function query, then we call a tuple also as a line in the rest of the paper when there is no confusion. As a line may intersect with some other lines, there happen two cases when a new tuple $(a_k, b_k)$ (or a line $l_k$) is added into the dataset $F$, where $k > 0$ be a positive integer. One is that $l_k$ intersects with some lines of $F$ at new intersection points, and the other is that $l_k$ intersects with the other lines of $F$ at some existing intersection points.

In the first case, many new intersection points are generated, into which $l_k$ is inserted. Let $x_j$ be a newly generated intersection point, $MB_{i+1}$ be the MB Tree constructed for the interval $(x_i, x_{i+1}]$, which is an existing interval that covers $x_j$. The interval $(x_i, x_{i+1}]$ is then divided into two intervals $(x_i, x_j]$ and $(x_j, x_{i+1}]$ by $x_j$. A MB tree is constructed for each of the new intervals, which can be implemented by inserting $l_j$ into $MB_{i+1}$ respectively. In the second case, $l_k$ is inserted into the existing intervals, that is, $l_k$ is inserted into some of the existing MB Trees in the bottom level $MB_B$ according to the rules mentioned in 2.3, respectively.

In the above two cases, the root of every MB Tree in $MB_B$ is new when adding a new tuple, whether the MB tree is updated or newly generated. This means that the value of every leaf node of $MB_T$ in the top level is newly generated, and many newly generated leaf nodes are inserted into the tree. Thus, $MB_T$ is re-constructed completely. Finally, the data owner signs the root of $MB_T$ with digital signature, and gets $sig(root)$.

*3.2.2 Data Deleting*

When a tuple is deleted from the dataset, some intervals are be merged because some intersection points are to be deleted. Let $(a_k, b_k)$ (or $l_k$) be the tuple (or line) to be deleted. Suppose $x_j$ be a intersection point that $l_k$ intersects with other line. Then the intervals $(x_{j-1}, x_j]$ and $(x_j, x_{j+1}]$ are merged into a single interval $(x_{j-1}, x_{j+1}]$. One of the two MB Trees $MB_j$ and $MB_{j+1}$ needs to be deleted. We assume that $MB_{j+1}$ is deleted and $MB_j$ is retained, and $(a_k, b_k)$ is deleted from $MB_j$. Then, $MB_T$ is re-constructed. The data owner accordingly signs the root of $MB_T$ with a digital signature scheme, and gets the current $sig(root)$.

## 3.3 Query

On receiving the query request issued by the user, the CS executes the query to get the query results $R$, and extracts the associated verification objects $VO$. The query process is divided into to two steps.

First, CS executes the query on $MB_T$ with the user input $x$. An interval $(x_{j-1}, x_j]$ is found that $(x_{j-1}, x_j]$ covers $x$, that is, $x_{j-1} \leq x \leq x_j$, where $j$ is a positive integer. This also indicates that the MB Tree $MB_j$ is the target in

the bottom level. The interval is just the query results of the top level, then $R_T = (x_{j-1}, x_j]$, and the verification objects $VO_T$ are also extracted.

Second, CS executes the query on $MB_j$ with the query condition. The user may provide different query condition according to different query type. The MB Trees in the bottom level are indexed by the value of $c_i \in C$. CS finds the results $R_B$ and extracts the verification objects $VO_B$ in the bottom.

Thus, the final results $R = R_T \cup R_B$ and verification objects $VO = VO_T \cup VO_B$ are obtained. $R$, $VO$ and $sig(root)$ are then returned to the user. The execution of the query is just the same as that of B+ Tree, which will not be introduced here. We mainly introduce the extraction of the verification objects, which is shown in Fig.4.

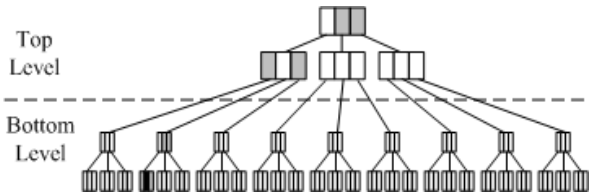


**Fig. 4** Verification Object of TLMB.

Suppose that the query results lie in the second entry of the first child node of the second MB Tree of $MB_B$, which is marked dark. Then, the entries marked gray are the verification objects.

Upon receiving $R$ and $VO$, the user re-constructs the authentication tree, which is shown in Fig.5. The root $root'$ of the MB Tree in the top Level is obtained in the end. The user utilizes the digital signature technique to check whether $root'$ matches $sig(root)$. If $root'$ matches $sig(root)$, then the result is correct; otherwise, reject it.

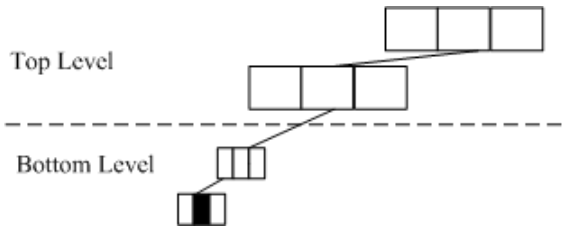


**Fig. 5** Restruction of TLMB.

## 4 Dynamic Authentication

In this section, we propose a scheme for efficient update of outsourced function query. We mainly introduce how to use the TLMB Tree for the results authentication. We adopt the signature mesh method [20] as the benchmark method.

### 4.1 The Benchmark Method

The benchmark method is to apply the signature mesh method in [20] directly. All the function data and the signatures are stored on the data owner side and on the server side, respectively. When a new function is added, the intersections of the newly added function with the existing functions are computed on the data owner side, and some new intervals will be added. The data owner then signs the newly added function and its successor. The newly added function and signatures are sent to the server. The server computes the intersections of the newly added function with the existing functions, and some new intervals will be added, which is the same as that on the data owner side. Then, the signatures are stored on the corresponding position of the signature mesh.

### 4.2 Our Proposed Method

Our goal is to store as few data as possible and perform as little computing as possible on the data owner side locally. The data and the computing are outsourced to the cloud server as much as possible. Because the most data are stored on the server side, the data owner needs to authenticate the update result when performing data update.

We now give the details of adding a new tuple. The update process can be divided into two steps. In the first step, DO requests to add a new function, CS computes the intersections, and the newly generated intersections and the related authentication data are returned to DO. In the second step, DO verifies the newly generated intersections. If the verification result is OK, then DO updates the authentication data structure, and release the new authentication data.

The update process is a five-tuple (**KeyGen**, **Insert**, **Compute**, **Verify**, **Update**). Owing to the usage of digital signature in our proposed authentication data structure, then generation of the keys is done by the algorithm **KeyGen**. The first step consists of two algorithms of **Insert** and **Compute**. The second step consists of two algorithms of **Verify** and **Update**. The details are given as follows.

**KeyGen.** DO generates the secret key $key$, a key pair $(pk, sk)$ for digital signature, where $sk$ is the secret key for signing and $pk$ is the pubic key for the signature verification.

**Insert.** Let $(a_k, b_k)$ be the tuple to be added. The DO sends the request $Insert(k, a_k, b_k)$ to the CS, which runs the algorithm, where $k$ is the number of the tuple.

We propose to store the whole dataset on the cloud server side to reduce the burden of data storage on the data owner side. When adding the tuple $(a_k, b_k)$, the CS computes the intersection points of $l_k$ with each line $l_j \in F$ to get a set $I_s$. Because the set $I_s$ is used to update the TLMB, we then need to verify the correctness $I_s$. We adopt the random check method to verify the correctness of $I_s$, that is, the whole dataset $F$ is stored on CS, and a part of $F$ is chosen randomly to store the copy locally on the DO side. We adopt the following equation to determine if the tuple is stored locally

$$H(k|key) \ mod \ m == 0, \tag{5}$$

where, $H$ here denotes a one-way hash function, $key$ is a secret key holden by the DO. If the equation holds, then the copy of the $k$th tuple is stored locally. The variable $m$ is a positive integer, which decides the portion of the copy stored locally. The larger the value of $m$ is chosen, the less data are stored locally. Suppose that the stored dataset copy locally on the DO side is $F_l = \{f_{l,j} = (j, k_j, a_{k_j}, b_{k_j}) | j = 1, \ldots, |F_l|\}$, where $|F_l|$ is the size of $F_l$.

**Compute.** On the receipt of $Insert(k, a_k, b_k)$ from DO, the CS computes the intersection points of $l_k$ and the existing tuples. The CS then updates the authentication data structure and extracts the corresponding verification objects.

Let the dataset stored on the cloud server side be $F_s = \{f_j = (j, a_j, b_j) | j = 1, \ldots, |F_s|\}$, where $|F_s|$ is the size of $F_s$. The cloud server computes the intersection points of the tuple $(k, a_k, b_k)$ and each tuple $f_j \in F_s$, and then a intersection set is obtained, denoted as $I_s = \{(j, x_{k,j}) | x_{k,j} = (b_j - b_k)/(a_k - a_j), j = 1, \ldots, |I_s|\}$, where $x_{k,j}$ is the intersection point of the tuple $f_j$ stored on CS and the newly added tuple $f_k$, where $|I_s|$ is the size of $I_s$. Note that $|I_s| = |F_s|$ because $f_k$ intersects with $f_j \in F$ $(i = 1, \ldots, |F|)$.

Adding a new tuple leads to two aspects of impact. On the one hand, a new tuple is inserted into the existing intervals, where new entries will be inserted into the authentication data structure. On the other hand, new intervals are added because new intersections are generated, where new authentication data structures will be constructed. The newly added intervals are split from the existing intervals, thus, the construction of new authentication data structures can be done by inserting entries into existing authentication data structures. Therefore, the two cases can be done by inserting entries into existing authentication data structures. When the server inserts new entries into the authentication data structures, the involved nodes that need to be updated are recorded and the verification object set is obtained, denoted as $VO = \{(x_j, vo_j) |, j = 0, 1, \ldots, n(n+1)/2\}$, where $vo_j$ is the verification objects of inserting new tuple $f_j$ into the authentication data structure of the interval $(x_j, x_{j+1})$, $n(n+1)/2$ is the interval count in the worst case. Finally, the cloud server sends the intersection set $I_s$ and the verification object set $VO$ back to the data owner.

**Verify.** The data owner computes the intersections of tuple $f_i$ and each tuple in $F_l$, denoted as $I_l$. If $I_l \subseteq I_s$, then $I_s$ can be considered correct; otherwise, reject it.

**Update.** If the intersections $I_s$ is considered correct, then the data owner further updates the authentication data structure and signs the root. The data owner inserts the newly added tuple into each authentication data structure in each interval one by one, and then constructs a new authentication data structure for the roots of the authentication data structure in the bottom level. Finally, the root is signed by the data owner.

In the new authentication data structure, the amount of hash is increased, and the amount of signature is decreased significantly to only one time, i.e., the signature of the root.

## 5 Performance Analysis

We give the performance analysis of the two methods in this section, i.e., the benchmark method in [20] and our proposed method. We first discuss the performance of adding a new tuple.

Suppose there exist already $n-1$ tuples. When a new tuple $f_n = (a_n, b_n)$ is added, $f_n$ will intersect with each of the $n-1$ tuples, and at most $n-1$ intersections and intervals will be added. Thus, when there are $n-1$ tuples, there are $n(n-1)/2$ intersections and $1+n(n-1)/2$ intervals. The $1+n(n-1)/2$ intervals can be grouped into two classes, the newly added and the existing. The tuple $f_n(x)$ will be inserted into the existing intervals. The newly added intervals are split from the existing intervals, and the tuple $f_n(x)$ will also be inserted into the newly added intervals. Then, the tuple $f_n(x)$ will be inserted into each of the $1 + n(n-1)/2$ intervals.

Therefore, when utilizing the signature chain technique, the signature chain in each interval will be updated. Specifically, the signatures of the newly added tuple and its predecessor must be updated. Two times of signature are performed in each interval, and $2 + n(n-1)$ times of signature are needed in total.

We then analyze the performance of our proposed scheme. The data owner computes the intersections of the newly added tuple and the tuples stored locally, and then performs update on the authentication data structure TLMB. The data owner inserts the newly added tuple into each MB Tree in the bottom level, the related nodes in each layer will be updated, and $2 * (1 + \lceil log_k n \rceil)$ hashes are involved for each MB Tree at most. There are $n(n+1)/2$ MB Trees, then $(1+\lceil log_k n \rceil) * n(n+1)$ hashes are involved in total. There are $n(n+1)/2$ leaf nodes for the MB Tree in the top level, whose height is $1+\lceil log_k n(n+1)/2 \rceil$ with $k$ as the fan-out of the non-leaf node. $1 + k^2 + \ldots + k^{\lceil log_k n \rceil} = (1 - k^{\lceil log_k n \rceil})/(1-k)$ hashes are involved for constructing the MB Tree in the top level. 1 signature is needed on the root in the end.

The theoretic comparisons of the two methods are shown in Table 1.

**Table 1** Comparisons

| Algorithm | running times in [20] | running times of our method |
|---|---|---|
| Hash | 0 | $(1+\lceil log_k n\rceil)*n(n+1)$ $+(1-k^{\lceil log_k n\rceil})/(1-k)$ |
| Signature | $2+n(n-1)$ | 1 |

The difference cannot be seen easily by the above table. We now give a specific example. Let $n = 1,000,000$. Then the method in [20] executes the signature operation $2 + n(n-1) = 999,999,000,002$ times. Our method executes the hash operation $(1 + \lceil log_k n\rceil) * n(n+1) + (1 - k^{\lceil log_k n\rceil})/(1 - k) = 7,000,007,111,111$ times for $k = 10$, and $4,000,004,010,101$ times for $k = 100$.

As can be seen that, the data owner executes more hash than digital signature. However, in our computer with an Intel Core i5 CPU running at 2.50 GHz with 4GB RAM, the wasted time of computing hash for a string is 0.024 milliseconds, and 4.83 milliseconds of signing the same string. The wasted time of digital signature is about 200 times as much as that of hash, and $7,000,007,111,111/999,999,000,002 = 7.000014$ and $4,000,004,010,101/999,999,000,002 = 4.00000801$. Therefore, the computing of hash is higher if considering the running time. The performance of deleting a tuple is similar as that of adding a new tuple, which will not be discussed here.

We then discuss the data amount stored on the data owner side. The copy of all the tuples are stored locally on the data owner side, i.e. $n$ tuples, in the benchmark method, whereas $n/m$ tuples are stored on the data owner side in our proposed method.

## 6 Related Works

The related works of result correctness verification in the research field of secure outsourcing can be grouped into two categories, i.e., query result correctness verification and computing result correctness verification.

The first category is the query result correctness verification of outsourced database.

Devanbu et al. first proposed to utilize Merkle Hash Tree(MHT) to verify the correctness of query result of outsourced database [3]. MHT is a binary tree, where the hash value is stored in the nodes, and root is signed by the data owner using digital signature. Li et al. combined MHT and B+ Tree to present another authentication tree Merkle B+ Tree to verify the correctness of query result of outsourced database, which works with higher efficiency [7]. Li et al. further presented another authentication tree for correctness verification of outsourced aggregate query [8]. As far as the spatial query is concerned, Yang et al. introduces Merkle R Tree (MR Tree) for correctness verification of KNN query on outsourced spatial data [21].

Xian et al. combined Chinese remainder theorem and B+ Tree to propose an authentication with mask MABTree, which reduces both the computation-

al overhead in the integrity check process and the verification time of the query results [18]. Wen et al. proposed an authentication data structure Min-Max Hash tree for results correctness verification of outsourced append-only database [17]. Pang et al. introduced the signature chain technique for the correctness verification of outsourced database query [11]. The adjacent three tuples are signed, which forms a signature chain. The adversary cannot tamper with and/or miss the query results, by which the soundness and completeness are achieved simultaneously.

Xie et al. pointed that the authentication data structure method and the signature chain method need to modify the back-end database management system, and and proposed to insert fake tuples secretly by the data owner, which is transparent to the server [19]. No modification needs to be done on the server side, and the server performs normal query. The user checks whether or not the fake tuples that should be included in the result really exist, if yes, the query result is correct. Wang et al. proposed the dual encryption method [16]. The data owner selects port of the data randomly, and encrypted the selected data and the original data with different keys, which are both stored on the server side. The server performs query on the two datasets respectively. The user can determine the correctness of the result by the relationship of the two datasets.

The second category is the result correctness verification of outsourced computing.

Fiore et al. adopted the discrete logarithm and bilinear map to present the schemes for publicly verifiable matrix multiplication, which enables the third party to verify the correctness of the computing result with public data [4]. Many other works follow the framework [9],[6],[13]. Li et al proposed efficient pseudorandom functions [9], and Sheng et al. proposed the matrix digest technique to reduce the verification-related computing from $O(n^2)$ to $O(n)$ [13]. Hohenberger et al. proposed the outsourcing of the modular exponentiation, and apply it for encryption [5]. The dual server method is adopted to verify the correctness of the computing result. The request is sent to two different servers, and the result correctness can be judged by the relationship of the return results. Many other works follow the framework for outsourced modular exponentiation [1],[12]. Chevalier et al. proposed a verification scheme for outsourced modular exponentiation with a single server [2].

Wang et al. proposed a secure scheme for outsourced linear programming, and the dual programming is used to verifying the correctness of the computing result [14]. Wang et al. proposed a secure scheme for outsourced linear system of equations, and adopt the iterative computing method [15].

## 7 Conclusions

In this paper, we propose a new scheme for outsourced univariate linear function query that support efficient data update, where as much data storage and computing as possible are delegated to the cloud server. We utilize the

random check method for correctness verification of line intersections, which enables the data owner to store small amount of data locally. We propose a novel authentication data structure TLMB Tree for query result verification of outsourced function query, which works efficiently for data update owing to the usage of hash function. The efficiency can be shown by the theoretic analysis, and the amount of data and computing is reduced significantly. In the future work, we plan to propose a more secure algorithm for intersection verification of outsourced linear function query.

## References


1. Chen, X., Li, J., Ma, J., Tang, Q., Lou, W.: New algorithms for secure outsourcing of modular exponentiations. IEEE Transactions on Parallel and Distributed Systems **25**(9), 2386–2396 (2014)
2. Chevalier, C., Laguillaumie, F., Vergnaud, D.: Privately outsourcing exponentiation to a single server: Cryptanalysis and optimal constructions. In: I. Askoxylakis, S. Ioannidis, S. Katsikas, C. Meadows (eds.) Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS 2016), pp. 261–278. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-45744-4$_1$3
3. Devanbu, P.T., Gertz, M., Martel, C.U., Stubblebine, S.G.: Authentic third-party data publication. In: Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security: Data and Application Security, Development and Directions, pp. 101–112. Kluwer, B.V., Deventer, The Netherlands, The Netherlands (2001)
4. Fiore, D., Gennaro, R.: Publicly verifiable delegation of large polynomials and matrix computations, with applications. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12, pp. 501–512. ACM, New York, NY, USA (2012). DOI 10.1145/2382196.2382250
5. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: J. Kilian (ed.) Proceedings Second Theory of Cryptography Conference (TCC 2005), Cambridge, MA, USA., pp. 264–282. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
6. Jia, K., Li, H., Liu, D., Yu, S.: Enabling efficient and secure outsourcing of large matrix multiplications. In: Proceedings of 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1–6 (2015). DOI 10.1109/GLOCOM.2015.7417184
7. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD '06, pp. 121–132. ACM, New York, NY, USA (2006). DOI 10.1145/1142473.1142488
8. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Authenticated index structures for aggregation queries. Acm Transactions on Information and System Security **13**(4), 747–759 (2010)
9. Li, H., Zhang, S., Luan, T.H., Ren, H., Dai, Y., Zhou, L.: Enabling efficient publicly verifiable outsourcing computation for matrix multiplication. In: Proceedings of 2015 International Telecommunication Networks and Applications Conference (ITNAC), pp. 44–50. IEEE (2015)
10. Merkle, R.C.: A certified digital signature. In: Proceedings of CRYPTO '89, Santa Barbara, California, Usa, August 20-24, 1989, pp. 218–238 (1989)
11. Pang, H., Jain, A., Ramamritham, K., Tan, K.L.: Verifying completeness of relational query results in data publishing. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, SIGMOD '05, pp. 407–418. ACM, New York, NY, USA (2005). DOI 10.1145/1066157.1066204
12. Ren, Y., Ding, N., Zhang, X., Lu, H., Gu, D.: Verifiable outsourcing algorithms for modular exponentiations with improved checkability. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ASIA CCS '16, pp. 293–303. ACM, New York, NY, USA (2016). DOI 10.1145/2897845.2897881

13. Sheng, G., Tang, C., Gao, W., Yin, Y.: Md-$\mathcal{VC}_{Matrix}$: An efficient scheme for publicly verifiable computation of outsourced matrix multiplication. In: J. Chen, V. Piuri, C. Su, M. Yung (eds.) Proceedings of the 10th International Conference on Network and System Security (NSS 2016), pp. 349–362. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-46298-$1_2$3
14. Wang, C., Ren, K., Wang, J.: Secure and practical outsourcing of linear programming in cloud computing. In: Proceedings of the IEEE INFOCOM 2011, pp. 820–828 (2011). DOI 10.1109/INFCOM.2011.5935305
15. Wang, C., Ren, K., Wang, J., Wang, Q.: Harnessing the cloud for securely outsourcing large-scale systems of linear equations. IEEE Transactions on Parallel and Distributed Systems **24**(6), 1172–1181 (2013)
16. Wang, H., Yin, J., Perng, C.s., Yu, P.S.: Dual encryption for query integrity assurance. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM '08, pp. 863–872. ACM, New York, NY, USA (2008). DOI 10.1145/1458082.1458196
17. Wen, T., Sheng, G., Guo, Q., Sheng, G.J.: Query results authentication of outsourced append only databases. Journal of Computer Research and Development **49**(10), 2077–2085 (2012)
18. Xian, H., Feng, D.: An integrity checking scheme in outsourced database model. Journal of Computer Research and Development **47**(6), 1107–1115 (2010)
19. Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07, pp. 782–793. VLDB Endowment (2007)
20. Yang, G., Cai, Y., Hu, Z.: Authentication of function queries. In: Proceedings of IEEE 32nd International Conference on Data Engineering (ICDE 2016), pp. 337–348 (2016). DOI 10.1109/ICDE.2016.7498252
21. Yang, Y., Papadopoulos, S., Papadias, D., Kollios, G.: Authenticated indexing for outsourced spatial databases. The VLDB Journal **18**(3), 631–648 (2009)