Verifiable Range Query Processing for Cloud Computing

Yanling Li[1], Junzuo Lai[1,2], Chuansheng Wang[1(✉)], Jianghe Zhang[1], and Jie Xiong[1]

[1] Department of Computer Science, Jinan University, Guangzhou, China
chueng0828@126.com
[2] State Key Laboratory of Cryptology, Beijing, China

Abstract. With the popularity of cloud computing technology, the clients usually store a mass of data in the cloud server. Because of the untrusted cloud servers, the massive data query raises privacy concerns. To prevent sensitive data on the cloud from hostile attacking, and obtain the query result timely, users usually use the searchable encryption technology to store encrypted data on the cloud. In the prior work, there are many privacy-preserving schemes for cloud computing, but the verification of these schemes cannot be ensured. Due to software errors, communication transmission failure or the dishonest features of the public cloud servers, only part of the data set was searched. So the integrity is also an urgent problem to be solved. In this paper, we propose a verifiable range query processing scheme with the ability to verify the correctness of query result. The key idea of this paper is to add additional information to a complete binary tree, which is used to organize indexing elements. The result returned by the cloud server will be accompanied by validation information so that the user can verify whether the result is complete. Finally, we confirm that the storage overhead of the verifiable scheme is $O(n \log n)$, where n is the total number of data items, and implement our scheme to testify to its practicability.

Keywords: Cloud computing · Range query · Verification

1 Introduction

1.1 Background

In recent years, as the Internet developed at a high rate of speed, our life and work affected by the Internet have become more convenient. Following the prevalent, the cloud computing is being integrated into our life and work. Instead of storing data in the hardware devices, increasing popularity, data and computing are outsourced to clouds for many factors. First, it does not require spending money on purchasing equipment, and it is wise to delegate the heavy computation workloads into the powerful servers. Obviously, outsourcing can reduce the cost effectively. Additionally, since the most resources that may be used in our work are existing in the cloud, we can transfer what we need from the cloud. It greatly improves the efficiency of our work. Because of these advantages, the cloud servers are favored by many businesses. At present, there are many companies with outsourcing computation, such as Google App Engine [10], IBM Blue Cloud Computing Platform [15], Amazon Web Services [1], and Microsoft Azure [19]. These service providers bring convenience to the cloud users.

Meanwhile, there is an obvious weakness for outsourcing computation [8]. In some special scenarios, secure outsourcing computation is significant. Yet, the data on the public cloud takes a high risk due to many causes. For instance, provided that data users try to request our data in the cloud, and our information would be leaked. Of course, that is not what we are willing to face. For example, last year, Apples iCloud leaked private photos uploaded by users, this issue given rise to people to consider whether the cloud storage is secure, especially for the confidential institutions, such as the national governments, securities traders, investment banks and others. Privacy becomes an urgent issue to be solved [20]. Beyond that, it is possible that the cloud may intercept data between users' transaction or return erroneous results to users. Therefore, we should strengthen data privacy protection at the same time to enhance the verification of computing and other security technology.

1.2 Motivation

Cloud server becomes more popular for people to store data, one person's data may be used by others. So, in this paper, we adopt a model as following: a data owner stores data on the cloud, and multiple data users could query the interested data on the cloud. For the most simple example, a data user stores his own data on the cloud and queries what he is interested from these data in the cloud. Figure 1 shows the three parties in our model: a data owner, multiple data users and a cloud. Data users usually protect sensitive data by encryption. Before uploading data on the cloud, the data owner encrypts data in order to data security. This operation ensures the confidentiality of the data, but all of these come at a price. For example, it becomes hard to query data on the ciphertext. When user queries data, first, he should download all data he stored on the cloud, decrypt these data, after that search out the required data. Obviously, it is infeasible when the data size is extremely large. Our motivation is to achieve verification of the query results in the case of ensuring data security.

1.3 Related Work

Prior works have made many contributions to data security. Here we just talk about the security for range query. The existing privacy-preserving query schemes are divided into two categories according to their query types: range queries [13] and key-word queries [5,7]. Rang queries which query all data items that fall into the given range, can also be called range searchable symmetric encryption schemes. Prior range searchable symmetric encryption schemes can be divided into two kinds: bucketing schemes [12–14] and order-preserving

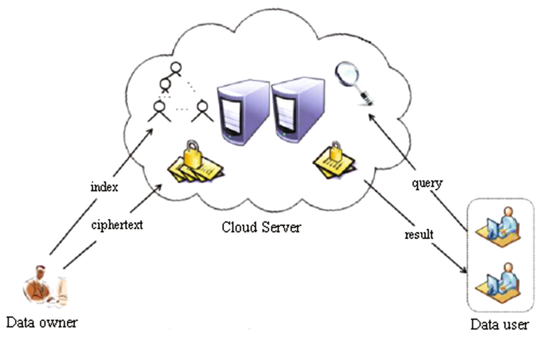


Fig. 1. Storing computing model

schemes [3,4,17]. In bucketing schemes, data owner partitions the data domain into various sizes. For example, the range $[0, 150]$ represents the age of human, we divide it into many ranges like that $[0, 12], [13, 22], [23, 60], [61, 150]$. Data owner constructs index by the ID of a bucket and all encrypted data items in this field. The trapdoor of a range query consists of the IDs of the buckets that overlap a query range. For instance, for query range $[10, 20]$, the corresponding trapdoor consists of ID1 and ID2. All data items in the buckets will be returned to data user on condition that the buckets overlap the query ranges. In this example, the cloud will return ranges $[0, 12], [13, 22]$ to the data user. From the above, we can get a conclusion that the prior encryption schemes still have many shortcomings. The weakness of private-preserving is the most significant. The Cloud could estimate the actual values according to historical query results. In addition to this, the communication cost of this scheme is very high, as there are many data items which are not gratified the query. Reducing the size of every bucket leads to lower cost, but weakens privacy at the same time. Because, in this case, the number of buckets approximates to the number of data items. It is easy to estimate the size of our data set.

In order-preserving schemes, data order keeps consistent after encryption. For example, for any two data items a and b, as well as a function f which is used to keep the order unchanged, called order-preserving encryption function. $a < b$ if and only if $f(a) < f(b)$. In order-preserving schemes, the index for data items $d_1, \ldots, d_n$ is $f(d_1), \ldots, f(d_n)$, and the trapdoor of range $[a, b]$ is $[f(a), f(b)]$. It is obvious that order-preserving is also weak for privacy, since they allow the cloud to estimate the actual values of the data items and the query in a statistical way.

The above mentioned schemes prove that the fundamental cause of the weak privacy preserving is that these data have different distributions when they have the same number of encrypted data items, in other words, they have index distinguished. In bucketing schemes, for different numbers of data items, different distributions in the data values will result in the regions to have different size distributions as to they require the number of data items in the equilibrium area. In order-preserving schemes, in the case of the same number of data items, the different distributions in the data values will lead to the ciphertexts to have different distributions in the space. Using the domain knowledge about the data distribution, the bucketing schemes and the order-preserving schemes can be used by the cloud to statistically estimate the values of the data and queries.

In view of the weak privacy protection caused by index distinguished, Li et al. [18] proposed a range query processing scheme that achieves index indistinguishability under the indistinguishability against chosen keyword attack (IND-CKA). They achieve index indistinguishability by complete binary tree, that is to say, when the number of data items is equal, they have the same data structure that can not be distinguished. And the nodes are indistinguishable, thanks to the randomness. They proved their scheme is privacy preserving under the widely adopted IND-CKA security model, but there are also existing many uncertain factors. Because the cloud is not credible, it may not try its best to query what the users interested, the results returned by the cloud may be wrong or incomplete. Nevertheless, for users, they can not judge what they have got is good or bad. Thus it needs operation operated by data users to verify the correctness of the results.

1.4 Our Contribution

At present, according to the study of the searchable encryption scheme, they are not exceedingly convenient for range query. For instance, users can not verify the integrality of the returned results in range query. Unreliable server may take incomplete data to users, and this problem would bring annoyance to the following work. To solve this question is an urgent issue for us.

In this paper, we proposed a leveled verifiable range queries scheme based on a private-preserving scheme which is proposed by Li et al. in [18]. We reserve the security and high efficiency of the original scheme, and obtain verification by storing additional information in the leaf nodes. Our main works as follows: firstly, analyzing [18], pointing out deficiency in the original scheme, its main defect is that users can not verify the correct and integrity of the results. This paper proposed a modified scheme aiming at these shortcomings. Not only do we analyze the security of our scheme, but we have done a comparison with the original scheme, and shown the advantage of our scheme by theoretical analysis and experiments. In our scheme, we need space is $O(n \log n)$ as before, but it has verification at the same time.

Next, we give a brief overview of the searchable encryption technology, hash function and Bloom filter that we utilize in Sects. 2 and 3, we describe our verifiable scheme in detail; In Sects. 4 and 5, we analyse security of our scheme and implement it respectively.

2 Preliminaries

2.1 Searchable Encryption Technology

Searchable encryption technology allows the client to store, on an untrusted server, message encrypted by a private or public key. The client could query related information from the untrusted server by a trapdoor, which is constructed by some key words, while the trapdoor does not reveal keywords or ciphertexts anymore. Searchable encryption technology is divided into two categories: symmetric searchable encryption [22] and asymmetric searchable encryption [5]. The searchable encryption technology can be divided into four steps:

(1) Encryption: A user encrypts messages with the private key, afterward uploads the ciphertext on an external server.
(2) Trapdoor Construction: Users with search permission construct trapdoor by encrypting query keywords, while the trapdoor does not leak any information about the keywords.
(3) Query: External server queries according to the trapdoor, after that returns the result to the data user. While the server only knows whether the files contain these keywords, but does not know other additional information.
(4) Decryption: The user receives the query results returned by the server, then decrypts ciphertext with private key to obtain related information.

2.2 Hash Function

Hash functions, also called compression functions, have many applications in cryptography and computer security. In general, hash functions are just functions that take arbitrary-length strings and compress them into shorter strings. Hash functions have many useful properties. Hash functions have security, since the rival can not restore the original data according to the output value in any polynomial time, namely, unipolarity. For example, the rival knows $H(x)$, but it is unlikely for the rival to compute x in any polynomial time. Besides that the adversary can not find two different input values and the output values are the same in any polynomial time. For instance, there is a pair of values x and x', and no polynomial-time adversary can compute $H(x) = H(x')$, that is to say that hash functions have Collision-Resistant. Typical hash functions are such as CR32, MD5, SHA1 [21] and so on. The hash functions exert a great influence on integrity and digital signatures.

2.3 Bloom Filter

Bloom filters are usually used to retrieve whether an element is in a collection [2]. It is actually a very long binary vector (each bit is set to be 0) and a series of random mapping functions. We compute every data item using hash functions to get the corresponding position in the hash table, secondarily we set this position to be 1. When judging one element whether in the collection, we just need to compute the hash functions, and find the corresponding position in the hash table, and check the value in the position. If the value is 1, it reveals that the element is in the collection, else we get the opposite consequence. We can affirm that the element is included by the collection while the corresponding positions are all set to be 1.

Comparing with other data structure, Bloom filter has a huge advantage in space and time. The most prominent advantage is that storage space and insert query time are all constant. Beyond that, hash function is independent, and it brings convenience to hardware to achieve parallel processing. Bloom filter does not store data items, hence it has great advantages in the occasion that has confidentiality requirements. Meanwhile, the shortcomings of Bloom filter are apparent as its advantages. False positive is one of them. As the number of deposited elements increases, the rate of false positive increases. When the number of data items more than a certain number, the element that is not included in set will obtain the same result as it is in the collection. The solution to the false positive is to create a small list which is called white lists that store elements that may be misjudged. In addition, it is unable to delete elements in Bloom filter, for the reason that it must ensure that the deleted element is indeed inside the Bloom filer, but it is not easily guaranteed. Bloom filter is generally used to query and filter spam.

2.4 Adversary Model

In this paper, we assume that the cloud is semi-honest (also called honest-but-curious) [6] as original scheme. That is to say that the cloud could execute our protocol and compute algorithm correctly to help us obtain the result. But at the same time, the cloud may try to analysis information obtained by the distribution or result before acquiring many useful messages. For example, in bucketing schemes, the cloud may according to the number of buckets to evaluate the number of data items when reducing communication. For the data owners and users, we assume that they are all trusted.

3 Verifiable Scheme

3.1 Prefix Encoding

As proposed in [18], we should first encode prefix as described in [5]. Through prefix encoding, we could check whether the data sets have the same elements instead of judging whether a data belong to a range. Next, we explain how to encode the data properly to submit it to the server. Given a number x, let the binary representation of x is $x_1x_2 \dots x_w$, where x_w is the least significant bit. Each number corresponds to a prefix family, denoted as $F(x)$, including $w+1$ prefixes: $\{x_1x_2 \dots x_w, x_1x_2 \dots x_{w-1}*, \dots, x_1 * \dots *, ** \dots *\}$, where the ith prefix is $x_1x_2 \dots x_{w-i+1} * \dots *$. For example, the prefix set of number 6 of 5 bits is $F(6) = F(00110) = \{00110, 0011*, 001**, 00***, 0****, *****\}$. Given a range $[a, b]$, firstly, we transfer it into a smallest prefix encoding set, represented as $S([a, b])$. In this way, the range represented by prefix encoding is same as range $[a, b]$. For example, $S([0, 8]) = \{00***, 01000\}$. In the given range $[a, b]$, a and b are two numbers of w bits, respectively, so the number of prefixes in $S([a, b])$ is at most $2w-2$ [11]. For any x and range $[a, b]$, when $x \in [a, b]$, $x \in p$ if and only if the prefix $p \in S([a, b])$. For any x and prefix p, $x \in p$ is same as $p \in F(x)$. So, for any x and range $[a, b]$, $x \in [a, b]$ if and only if $F(x) \in S([a, b])$. According to the example above, $6 \in [0, 8]$ and $F(6) \cap S([0, 8]) = \{00{*}{*}{*}\}$. In this paper, for n data $d_1, d_2, \ldots, d_n$, the data owner computes prefix families $F(d_1), \ldots, F(d_n)$, and the data user can compute prefix family $S([a, b])$ of range $[a, b]$.

Before uploading data to the cloud server, the user should sort the data and record the values before and after the data. For example, if the uploaded data set is $S = \{1, 9, 4, 8, 14, 11, 16, 21, 26, 10\}$. After sorting, the set S is transferred into $S' = \{1, 4, 8, 9, 10, 11, 14, 16, 21, 26\}$. We denote $P(x)$ as the value x and the values of its before and after. For example, $P(4) = \{1, 4, 8\}$. When the value is the head or tail of the sequential queue, we denote as ∞ or $-\infty$, such as $P(1) = \{-\infty, 1, 4\}$, $P(26) = \{21, 26, \infty\}$.

3.2 PBtree Construction

In order to achieve the efficient query, we store $F(d_1) \ldots F(d_n)$ in a complete binary tree, called PBtree. Here, "P" means privacy and "B" means Bloom filter. We do not use existing database indexing structures (such as $b+$ tree) for two reasons as follows: when the two numbers are relatively large, query in the $b+$ tree also need do some testing work; $b+$ tree storing different data items has different structures, even they have the same number of data items. While two different data sets that have equal size are stored on the PBtree respectively, then two PBtree have same data structures, that is to say the two PBtrees are indistinguishable.

Definition 1 (PBtree). *The PBtree used to store n data items is a full binary tree, which has n terminal nodes and $n - 1$ non-terminal nodes. In PBtree, n terminal nodes form a linked list from left to right, and every node is represented by a Bloom filter. Each leaf node stores a data item, and each non-terminal node stores the union set of its left and right subtrees. For any non-terminal node, the number of data items in its left subtree either equals that of its right subtree or exceeds only by one.*

According to this definition, we can easily know that PBtree is a highly balanced binary research tree. The height of the PBtree storing n data items is $\lfloor \log n \rfloor + 1$. We construct a PBtree adopting a top-down fashion. Firstly, we construct the root node. The root node contains the set of prefix $\{F(d_1), \ldots, F(d_n)\}$. Then, we divide the prefix set $\{F(d_1), \ldots, F(d_n)\}$ into two subsets S_{left} and S_{right}. If n is even, $|S_{left}| = |S_{right}|$, else $|S_{left}| = |S_{right}| - 1$. The two subsets are the root nodes of the left and right childtree respectively. For any left subtree and right subtree, we recursively apply the above steps until the terminal node. Each terminal node contains prefix set of one data item. Figure 2 shows the PBtree for prefix set $S = \{F(d_1), F(d_2), F(d_3), F(d_4), F(d_5), F(d_6), F(d_7), F(d_8), F(d_9)\}$.

In Theorem 1, the key properties of PBtree are simply described according to their construction algorithm. Constraint $0 \leq |S_{left}| - |S_{right}| \leq 1$ makes the structure of PBtree completely dependent on the number of data contained.

Theorem 1 (Structure indistinguishable). *For any two data sets S_1 and S_2, they have the same constructions of PBtrees if and only if $|S_1| = |S_2|$.*

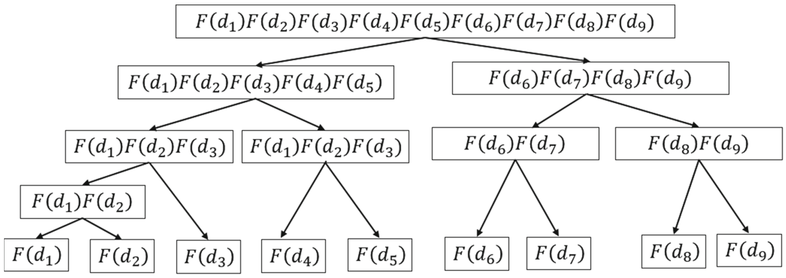


Fig. 2. PBtree example

3.3 Data Encryption

In this paper, we have two parts to be encrypted, data items and prefixes.

For data items, we adopt asymmetric encryption. Here our encryption is based on a n bit RSA modulus. The encryption process as follows.

(1) Generate a n bit RSA modulus $n = pq$ for primes p, q;
(2) Choose an integer e satisfying $gcd(g(n), e) = 1$ and $1 < e < g(n)$, where $g(n) = (p-1)(q-1)$;
(3) Compute $d \equiv e^{-1} mod\ g(n)$;
(4) The public key is now $pk = (e, n)$, and the secret key is $sk = (d, n)$. For all ordered data items $d_1, \ldots, d_n$, the encryption term of the ith data item d_i is $C_i = (d_{i-1}||d_i||d_{i+1})^e (mod\ n)$.

The encryption of prefixes is still implemented by secure hash function and Bloom filter. For each node v, the prefix family of node v is stored by Bloom filter, represented as $v.B$. Assuming r secret keys $k_1, \ldots, k_r$ have been shared between the data owner and the data user. $L(v)$ is a label of node v, which contains prefix sets. $U(v)$ represents a union set of prefix sets in $L(v)$. For example, if the two prefix families $F(x)$ and $F(x')$ are in the node v, then the set $L(v) = \{F(x), F(x')\}$, and the set $U(v) = \{F(x) \cup F(x')\}$. Each data is a w-bits binary data.

For prefix p_i, we compute p_i with r keys using hash functions: $HMAC(k_1, p_i), \ldots, HMAC(k_r, p_i)$. This step is to achieve one-wayness. That is to say that we can easily compute $HMAC(k_j, p_i)$ with r keys and p_i, but it is hard to obtain p_i and r keys even the adversary knows $HMAC(k_j, p_i)$, where $1 \le j \le r$. For any node v, generating a random number $v.R$ which has the same size with keys. Then using $v.R$ to compute r hash functions: $HMAC(v.R, HMAC(k_1, p_i)), \ldots,\ HMAC(v.R, HMAC(k_r, p_i))$. For each prefix p_i and for each key k_j, we set $v.B[HMAC(v.R, HMAC(k_j, p_i)) \bmod M] := 1$, where M is the length of the Bloom filter.

So far, the PBtree has been constructed by the data owner, then the data owner sends encrypted data and PBtree to the cloud server.

3.4 Trapdoor Computation

Before querying data from the cloud server, it is necessary for the data user to computing trapdoor. Given a range $[a, b]$ that used to be queried. Suppose $S[a, b]$ contains z prefixes $p_1, \ldots, p_z$. For any prefix, the data user computes r results of hash functions $HMAC(k_1, p_i), \ldots, HMAC(k_r, p_i)$. The trapdoor of the range $[a, b]$ is represented as a matrix $M_{[a,b]}$ that is consist of $z * r$ hashes.

$$\begin{pmatrix} HMAC(k_1, p_1) & \cdots & HMAC(k_r, p_1) \\ \cdots & \ddots & \cdots \\ HMAC(k_1, p_z) & \cdots & HMAC(k_r, p_z) \end{pmatrix}$$

The ith prefix p_i corresponds to the ith row of the matrix of the trapdoor. Then data user sends the matrix $M_{[a,b]}$ to the cloud server (Fig. 3).

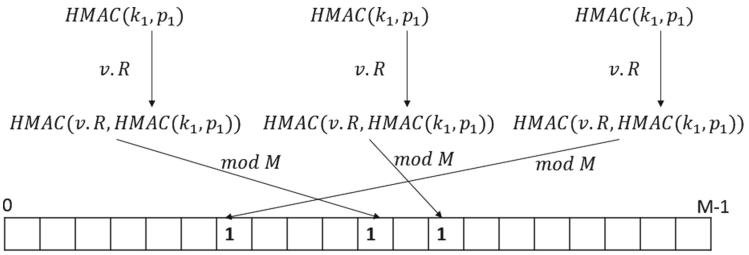


Fig. 3. Bloom filter

3.5 Query Processing

After receiving the trapdoor sent by the data user, the cloud server uses the trapdoor to search on the PBtree. Firstly, the cloud checks whether $v.B[HMAC(v.R, HMAC(k_j, p_i)) \bmod M] := 1$ for every j $(1 \leq j \leq r)$ in ith row in the matrix $M_{[a,b]}$. If there exists a row i $(1 \leq i \leq z)$ in $M_{[a,b]}$ satisfying $v.B[HMAC(v.R, HMAC(k_j, p_i)) \bmod M] := 1$, then it indicates that there may exists p_i in the PBtree. If there has at least one equation as $v.B[HMAC(v.R, HMAC(k_j, p_i)) \bmod M] := 0$, then we can infer that $U(v) \cap p_i = \phi$. For any subtree node v' of node v, there exists $U(v') \cap p_i = \phi$, because $U(v') \subset U(v)$. Then we can remove ith row of the matrix $M_{[a,b]}$ from $M_{[a,b]}$. We take new matrix to search on the PBtree. We continue that operation on the PBtree, until the matrix $M_{[a,b]}$ becomes empty or we finish searching terminal nodes.

Now, we analyze the time complexity of this algorithm. The number of PBtree index items is n, the query range is $[a, b]$, and the number of query result is R.

The average runtime of query algorithm depends on the size of query result $|R|$, if $|R| = 0$, then it will only need check the root node of PBtree, so the time complex is $O(1)$ in this case. While n is usually much larger than $|R|$ in the real word. So as to querying every item in the result set R, we need to traverse at least $2(\log n) - 1$ nodes. Therefore, the time complex of this algorithm above is $O(|R| \log n)$ generally (Fig. 4).

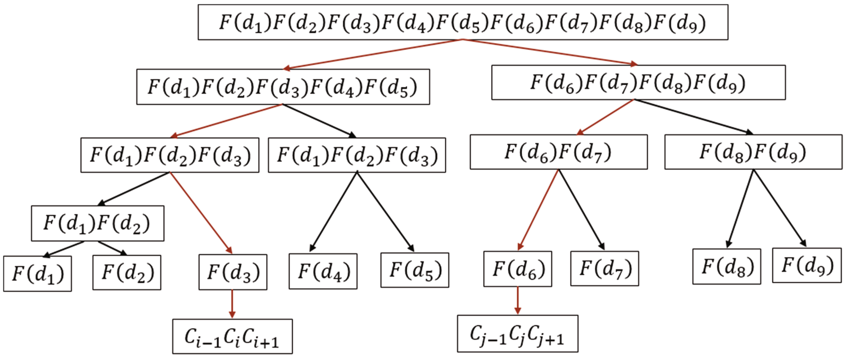


Fig. 4. Query for cloud server

3.6 Decryption

If the finishing condition is searching on terminal node, there exists ith row of $M_{[a,b]}$ for every p_i satisfying $v.B[HMAC(v.R, HMAC(k_j, p_i)) \ mod \ M] := 1$. It shows that the number in this terminal node falls into the range $[a, b]$ that the data user queries. The ciphertext of this number in the terminal node is $C_i = (d_{i-1}, d_i, d_{i+1})^e(mod \ n)$. Here, the secret key is $sk = (d, n)$. The data user computes C_i with secret key sk to obtain plaintext, the plaintext is $(d_{i-1}, d_i, d_{i+1}) = C_i^d(mod \ n)$. Then, the data user obtains interested number d_i. So far, the query operation is implemented.

3.7 Verification

Prior work is completed at the end of the query, they can not ensure the integrity of the query results. But, in our scheme, we have added additional information into PBtree. The data user gets the result which not only contains what the data user wants, but adjacent data items, which would be used to verify whether the result is really integrated or not.

If the cloud server did not query the matching results, it will return the whole PBtree to the data user for verification. On the other hand, if the cloud server has got the related data set, it will return the result to the data user. Supposing that the range of data user queried is $[a, b]$, after querying in the cloud server, the result is $\{d_i, d_j, d_k\}$. But the data set returned to the data user includes other adjacent data items, it is $\{d_{i-1}, d_i, d_{i+1}, d_{j-1}, d_j, d_{j+1}, d_{k-1}, d_k, d_{k+1}\}$. The data user could use these additional data items to judge whether the result returned by the cloud server is complete.

3.8 False Positive Analysis

Because of the property of the Bloom filter, there always exists false positive when we use Bloom filter to judge whether a prefix is concluded in a set. In order to improve the accuracy of the query results, we need to estimate the rate of false positive to make the query optimal. We always set the number of hash functions is $(m/n) \times \ln 2$, and it will minimize the false positive rate on this condition. As analysed in [18], we can easily get the relationship between a and M_a as follows:

$$M_a = af \times \frac{1-(2f)^{h-\lceil log\ a\rceil}}{1-2f} + (2^{\lceil \log\ a\rceil} - a)f(2f)^{h-\lceil log\ a\rceil}$$

where a is the size of all possible query result sets and M_a is denoted as the maximum expected number of false positives. When $f = 0.05$ and $h = 13$, the relationship between M_a and a is as shown in Fig. 5.

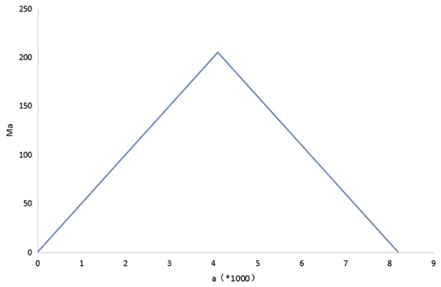


Fig. 5. Relationship of M_a and a

4 Security Analysis

4.1 Security Model

PBtree achieves IND-CKA security by pseudo-random function. There is no unavoidable advantage to distinguish it from the random function [16]. This pseudo-random function is: $g : \{0,1\}^n \times \{0,1\}^s \rightarrow \{0,1\}^m$. It means that inputting a string of n bit and a string of s bit maps a m bit string. And the random function is $G : \{0,1\}^n \rightarrow \{0,1\}^m$. This function is used to map a n bit string to a m bit string. For the pseudo-random function g, selecting a fixed scalar $k \in \{0,1\}^s$, and it can efficiently compute $g(x,k)$ for any $x \in \{0,1\}^n$. In polynomial time, the rival has no negligible probability to distinguish $g\{x,k\}$ and the output of the random function G. We use $HMAC()$ as the pseudo-random function. When the adversary does not have negligible probability in polynomial time to distinguish between the actual index generated using the pseudo-random function and the simulated index generated by the random function, then it demonstrates that this symmetric searchable encryption scheme is secure.

4.2 Security Proof

We treat PBtree as a series of Bloom filters, each of them storing a set of different prefixes that respond to user queries. Therefore, it can be observed that the safety proof of PBtree can be equivalent to proving that the Bloom filter is compliant with IND-CKA and satisfies the following conditions: (1) It can not be leaked any information about data items stored in Bloom filters; (2) the adversaries can not distinguish two Bloom filters storing different size of data sets. We consider a non-adaptive adversary who has finite original query results, including a set of security trapdoors and their corresponding query results. To help proving, we assume a probabilistic polynomial time simulator S, it can simulate the creation of a security index, which retains only a small number of history search query traces. The rival using S as using a real index to query, the challenge of the adversary is whether there is a negligible probability to make a distinction between the results returned by two different indexes. In the following definition, let the security parameter s be the length of the secret key.

Records of historical query H_q. The set $D = \{D_1, D_2, \ldots, D_n\}$ represents a set of data, and D_i is the ith data item. The set $R_{1:q} = \{R_1, R_2, \ldots, R_q\}$ represents the range query for q times, and the format of each query is $R_i = \{a_i, b_i\}(a_i, b_i, q \in N)$. Historical records is defined as $H_q = \{D, R_{1:q}\}$, where D contains at least one query that satisfies $R_{1:q}$. In order to limit the adversary to be solvable in the polynomial time, q must be a polynomial of the safe parameter s.

Advantage of the adversary A_v. For each range query $R_i = \{a_i, b_i\}$, there will be a generation of r_i trapdoors $T_i = \{t_{i,1}, t_{i,2}, \ldots, t_{i,r_i}\}$, then we encrypt them with secret key K. The advantages of the adversary include the trapdoor that satisfies the range query, security index I of the data set D, and the set of encrypted data items $Enc_K(D) = \{Enc_K(D_1), Enc_K(D_2), \ldots, Enc_K(D_n)\}$. Here, $A_v(H_q) = \{T; I; Enc_K(D)\}$. In addition, the adversary may also know the approximate amount of encrypted data.

Trace of the query. Defined as an adversary to match in index I after using T access and search model. The data items matching the access pattern are $M(T) = \{m(t_1), m(t_2), \ldots, m(t_q)\}$. $m(t_i)$ represents the data item that matches the trapdoor t_i. Search model is an asymmetric binary matrix $\prod_T$ defined on T, and when $t_p = t_q$, $\prod_T [p, q] = 1$. The trace $M_{(Hq)} = \{M(R_{1:q}), \prod_T [p, q]\}$ is defined on H_q. In the two modes, the adversary obtains only one set of matching data for each trap. Thus, each Bloom filter can be treated as a different match(which may be the same in PBtree). Each range query can not match to multiple different trapdoors.

Theorem 2. *PBtree scheme is IND-CKA security base on the pseudo-random function f and the encryption algorithm Enc.*

Proof. The adversary can construct a polynomial time simulator $S = \{S_0, S_q\}$ with the advantage $A_v(H_q)$ and a real result query trace M_{Hq}. $A_v^*a(H_q)$ denotes the advantage of rival simulation, I^* denotes the index of the simulation, $Enc_K(D^*)$ denotes the simulated encrypted data, and T^* denotes the trapdoor. According to definition, each Bloom filter matches a different trapdoor, and the query results are visible. ID_j represents a unique identifier for a Bloom filter. The final output of the simulator is a trapdoor created by the historical traces of the query range that the adversary selected, assuming that the adversary can not know the index and the trapdoor before selecting range.

First: simulate index. It is known that the length and number of Bloom filters are related to I, and generate a string B^* with the same length as $I^*$to simulate the index I^*, set to 1 in the random bit, and ensure that the number of position set to 1 is similar in each Bloom filter of each layer. Then, we generate random $Enc_K(D^*)$, each of which has the same length as the original encrypted data $Enc_K(D)$, $|Enc_K(D^*)| = |Enc_K(D)|$.

In the index I^*, we store the entire set of $Enc_K(D^*)$ in the first Bloom filter representing the PBtree root node. In the next two Bloom filters store two subsets of $Enc_K(D^*)$, and for each data, it is assigned to one of the Bloom filters through throwing coins. We take this operation in turn, so that the number of data for each subset is differ by no more than one.

Second: Simulator state S_0. In h_q, when $q = 0$, it represents that the simulator state is S_0. We define the adversary's advantage is $A_v^*(H_0) = \{T^*; I^*; Enc_K(D^*)\}$. In the trapdoor set T^*, each data item in $Enc_K(D^*)$ corresponds to a matching trapdoor. The length of each trapdoor is calculated by the random function g, and the maximum length of the trapdoor may depend on the length of the data in the prefix set(when the length of data is n, the length of the trapdoor is $n + 1$). Therefore, we generate $(n + 1) * |Enc_K(D^*)|$ trapdoors with the length of $|g(.)|$, and each data in $Enc_K(D^*)$ is associated with no more than $n + 1$ trapdoors. The distribution of each trapdoor in index I^* is the same as in the original index I, and the structure of the index generated by the simulation is exactly as same as the index structure generated in PBtree. Since g is a random function and the distribution probability of the trapdoor is uniform, this distribution is indistinguishable for the adversary in probability polynomial time.

Third: simulator state S_q. In h_q, when $q \geq 1$, it represents that the simulator state is S_q. We define the advantage of the adversary is $A_v^*(H_q) = \{T^*; T_q; Enc_K(D^*)\}$. T_q is the historical query of the corresponding trapdoor. Considering that data set in each trapdoor is $M(T) = m(t_1), m(t_2), \ldots, m(t_q)$, $M(R_{1:q})$ contains p unique data. In each data $Enc_K(D_p)$, simulator combines the trapdoors and the corresponding data items of $M(T_i)$. Because of $p < |D|$, simulator generates i random strings$(1 \leq i \leq |D| - q + 1)$. And we associate $Enc_K^*(D_i)$ with $n + 1$ trapdoors as the second step to ensure that the strings do not match the strings in $M(T_i)$. The simulator state ST_q records trapdoors and the matching data. For the first Bloom filter, we map identifiers of all data: $Enc_K(D^*) = M(R_{1:q}) \cup Enc^*_K(D_i)$, the Bloom filter of child node are operated in the way that we have described above. The output of simulator is $\{T^*; T_q; Enc_K(D^*)\}$. All steps are made by the simulator in polynomial time.

If data queried by the adversary is matched with the set $M(R_{1:q})$ in the probability polynomial time, the simulator will provide the correct trap. For other data, because of the random function g, the trapdoors provided by the simulator are indistinguishable. Since each Bloom filter contains random bits that are set to 1, our scheme is proved safe under the IND-CKA model.

5 Experiment Evaluation

5.1 Experiment

In this section, we implement our verifiable scheme, and evaluate it in terms of computational cost, query cost, security and verification. Specifically, in our experiments, we develop our scheme on Ubuntu 16.04 with 8 GB memory and an intel core i7-6700 processor. We use HAMC-SHAI as the pseudo-random function in the Bloom filter. For the Bloom filter, its length m and the stored data amount n satisfy such a relationship: $m/n = 10$. We use the virtual machine to simulate the operation of the server. The data used for presentation and performance test are randomly generated by the $random()$ function.

First, data owner transfers the data to cloud server by a client. The client reads and orders these data, then records every data item as a triplet. Its duties also include encryption, and it encrypts data items and prefixes with AES algorithm. After constructing PBtree, the client sends encrypted data and PBtree to the cloud server. We chose random datasets which consist of 10000 to 100000 records. Next, when someone wants to query data from the server, he should input the query range to the client, then the trapdoor will be computed and sent to server by the client. Last, when the server receives the trapdoor, it will use the method we mentioned before to match the data on the server. When the matching is success, the server returns the result data set, and when it fails to match the corresponding data, the server returns all the data stored on PBtree to the client for users verifying.

5.2 Evaluation

To evaluate our work, we compared our work with existing range query scheme on ciphertext including the private-preserving range query scheme, bucketing schemes and order-preserving schemes. In evaluation of the performance of our scheme, we consider five factors: local computing overhead, server query computing overhead, server storing overhead, security and verification. The table shows the result. In this comparing work, we set the data size is n, and the query size is R.

According to the table, our scheme increases a little computing and storing overhead, but compared to the paper [5,9], it has better private-preserving.

Besides, our scheme not only has the same algorithm complexity as the original project, but also has the significate property of verification (Table 1).

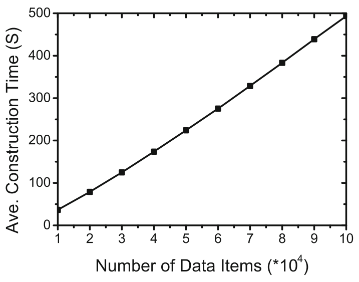
Table 1. Performance comparison

Schemes	Local computation	Query computation	Storage	Security	Verification
Rang query	$O(n \log n)$	$O(\|R\| \log n)$	$O(n \log n)$	Strong	No
Bucketing	$O(n^2)$	$O(\|R\| \cdot n)$	$O(n)$	Weak	No
Order-preserving	$O(n \log n)$	$O(n)$	$O(n)$	Weak	No
Our scheme	$O(n \log n)$	$O(\|R\| \log n)$	$O(n \log n)$	Strong	Yes

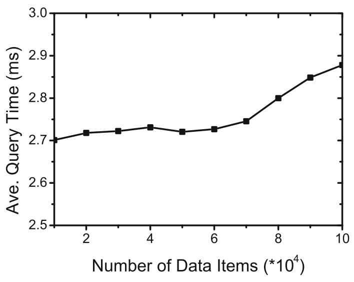
In this section, we answer the running time of our verifiable scheme. For the time of the assessment, we mainly take into account two phases: a construction phase and a query phase.

In the construction phase, this process includes three interactive steps, which are data ordering, prefix encoding and PBtree construction. The results are as Fig. 6(a).

In the second phase, query process consists of generating and transmitting the trapdoor, matching the prefix, and decrypting the data. The results of this phase are shown in Fig. 6(b).



(a) Construction time



(b) Query processing time

Fig. 6. Performance evaluation

6 Conclusion

In this paper, we present that although the private-preserving range query scheme proposed by Li et al. is security under the IND-CKA model, but it is not satisfied verification. Data users receive the result that the cloud server returns, but it is trouble for data users that they cannot be sure whether the query result is completely correct. Our scheme is based on the private-preserving range query scheme, and achieves the property of verification by adding additional information into the query result, and data users utilize additional information to verify the query result.

References

1. Amazon: Amazon Web Services. http://aws.amazon.com
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970). http://doi.acm.org/10.1145/362686.362692
3. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_13
4. Boldyreva, A., Chenette, N., O'Neill, A.: Order-preserving encryption revisited: improved security analysis and alternative solutions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 578–595. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_33
5. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_30
6. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, 22–24 May 1996, pp. 639–648 (1996). http://doi.acm.org/10.1145/237814.238015
7. Chang, Y., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. IACR Cryptology ePrint Archive 2004, 51 (2004). http://eprint.iacr.org/2004/051
8. Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., Molina, J.: Controlling data in the cloud: outsourcing computation without outsourcing control. In: Proceedings of the First ACM Cloud Computing Security Workshop, CCSW 2009, Chicago, IL, USA, 13 November 2009, pp. 85–90 (2009). http://doi.acm.org/10.1145/1655008.1655020
9. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2
10. Google: Google App Engine. https://en.softonic.com/
11. Gupta, P., Mckeown, N.: Algorithms for packet classification. IEEE Netw. **15**(2), 24–32 (2002)
12. Hacigümüs, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, 3-6 June 2002, pp. 216–227 (2002). http://doi.acm.org/10.1145/564691.564717
13. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure multidimensional range queries over outsourced data. VLDB J. **21**(3), 333–358 (2012). https://doi.org/10.1007/s00778-011-0245-7
14. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004, pp. 720–731 (2004). http://www.vldb.org/conf/2004/RS19P2.PDF
15. IBM: IBM Blue Cloud Computing Platform. https://www.ibm.com/cloud-computing/
16. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press, Boca Raton (2007)

17. Li, J., Omiecinski, E.R.: Efficiency and security trade-off in supporting range queries on encrypted databases. In: Jajodia, S., Wijesekera, D. (eds.) DBSec 2005. LNCS, vol. 3654, pp. 69–83. Springer, Heidelberg (2005). https://doi.org/10.1007/11535706_6
18. Li, R., Liu, A.X., Wang, A.L., Bruhadeshwar, B.: Fast and scalable range query processing with strong privacy protection for cloud computing. IEEE/ACM Trans. Netw. **24**(4), 2305–2318 (2016). https://doi.org/10.1109/TNET.2015.2457493
19. Microsoft: Microsoft Azure. `http://.microsoft.com/azure`
20. Ren, K., Wang, C., Wang, Q.: Security challenges for the public cloud. IEEE Internet Comput. **16**(1), 69–73 (2012). https://doi.org/10.1109/MIC.2012.14
21. Rivest, R.: The MD5 Message-Digest Algorithm. RFC Editor (1992)
22. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000, pp. 44–55 (2000). https://doi.org/10.1109/SECPRI.2000.848445