Information Sciences 498 (2019) 91–105

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Efficient privacy-preserving data merging and skyline computation over multi-source encrypted data

Yandong Zheng[a], Rongxing Lu[a,*], Beibei Li[b], Jun Shao[c], Haomiao Yang[d], Kim-Kwang Raymond Choo[e]

[a] *Faculty of Computer Science, University of New Brunswick, Fredericton, Canada*
[b] *School of Electrical and Electronic Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore*
[c] *School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, China*
[d] *School of Computer Science and Engineering and Center for Cyber Security, University of Electronic Science and Technology of China, Chengdu, China*
[e] *Department of Information Systems and Cyber Security and Department of Electrical and Computer Engineering, The University of Texas at San Antonio, San Antonio, TX 78249, USA*

A R T I C L E I N F O

Article history:
Received 29 September 2018
Revised 13 April 2019
Accepted 19 May 2019
Available online 20 May 2019

Keywords:
Leftist tree
Data comparison
Data merging
Skyline computation
Multi-source data

A B S T R A C T

Efficient data merging from the significant amount of data routinely collected from various data sources is crucial in the uncovering of relevant and key information of interest (e.g., skyline). There are, however, privacy considerations during data merging and skyline operations, particularly when dealing with sensitive data (e.g., healthcare data). Existing focuses on data merging and skyline computation either do not (fully) consider data privacy or have low efficiency. Thus, in this paper, we aim to address both privacy and efficiency during data merging and skyline computations over multi-source encrypted data. Specifically, we integrate the leftist tree with public key encryption and index based skyline computation to achieve data merging and skyline computation over encrypted data. First, we design a non-interactive data comparison protocol using public key encryption technique. This allows us to compare encrypted and outsourced data under a single cloud server instead of two non-colluding cloud servers in previous studies. Then, we combine the leftist tree with public key encryption to achieve privacy-preserving data merging with high efficiency, namely, $O(\log_2(n_1 + n_2))$ computational complexity for merging two leftist trees of sizes n_1 and n_2. Third, we present an index and leftist tree based skyline computation algorithm, which can efficiently perform skyline query over the merged encrypted data. Then, detailed security analysis and performance evaluation demonstrate that our scheme is both secure and efficient for data merging and skyline computation.

1. Introduction

It was estimated that approximately 2.5 quintillion bytes of data were produced each day [9], and such data play a vital role in decision making, business planning, knowledge discovery, and so on [20]. This has resulted in the continuing interest in big data analysis in applications ranging from medical diagnosis to personalized recommendation service, etc

* Corresponding author.
E-mail address: rlu1@unb.ca (R. Lu).

https://doi.org/10.1016/j.ins.2019.05.055
0020-0255/© 2019 Elsevier Inc. All rights reserved.

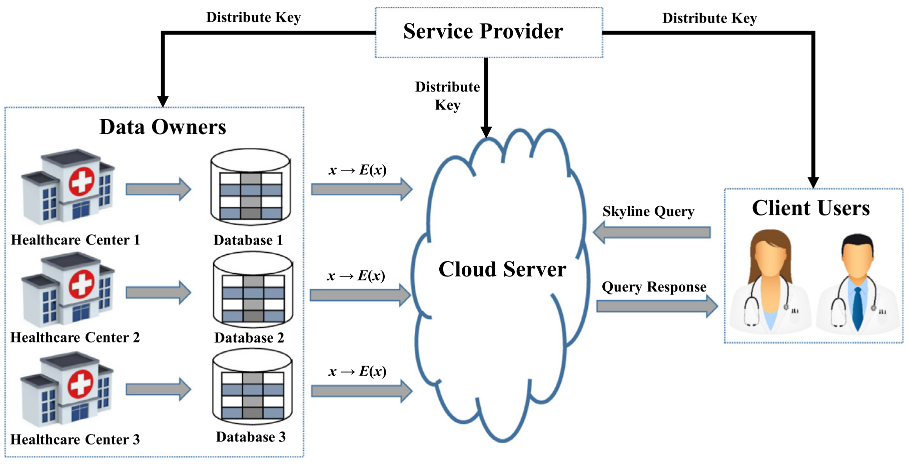


Fig. 1. System model under consideration.

[8,13,14,31]. However, the accuracy and reliability of such applications depend heavily on the volume of data. For instance, skyline computation over a small dataset is not as accurate as that over a large dataset [18,26].

The volume of data may be increasing (significantly) in general, data source scarcity case is still an issue. For example, the number of breast cancer or leukemia cases in a single hospital is usually insufficient to support accurate and reliable diagnosis. This complicates data analysis. A straightforward method is to merge data from various data owners (e.g., the cancer treatment centers throughout the state of Tex as such as UT Health San Antonio, MD Anderson Cancer Center, and Texas Breast Specialists-Arlington), where merged data have the same attributes and format in order to facilitate data analysis over the merged data. Researchers have proposed mergeable heaps such as leftist tree [6], binomial heaps [4] and Fibonacci heaps [10] to achieve efficient data merging. However, such approaches will leak sensitive information (e.g., identity and other identifying information of patients) to the entity that conducts the data merging. Despite the importance of designing a privacy-preserving data merging scheme, this is seldom considered in previous studies.

Skyline query is one of the most important queries as it has potential application in various areas such as target decision, data mining, and data visualization [13,21,32]. Computing skyline is to extract a collection of data points that are not dominated by any other points. A number of algorithms to address skyline computation have been proposed in the literature, and these examples include block-nest loop (BNL) [3], sort-filter skyline [5], divide-and-conquer [3], bitmap [24], index [24], and nearest neighbour [15]. Of these examples, index is reportedly the fastest algorithm for producing the entire skyline under all settings [21]. These algorithms are also performed in plaintext data without considering data privacy. When skyline computation is conducted over some sensitive data from multiple data sources, such as the scenario described in Fig. 1, it should be in a privacy-preserving manner. This is particularly important for organizations operating within the European Union and the European Economic Area, given the recent European Union General Data Protection Regulation (GDPR). In 2016, for example, Liu et al. [18] proposed an efficient privacy-preserving skyline computation framework to achieve skyline computation over multiple domains in distributed environments. However, in order to achieve global skyline computation, one data owner has to interact with other data owners multiple times. Thus, this is inefficient for some applications.

We posit the importance of achieving privacy-preserving and efficient skyline computation in a non-interactive manner. Therefore, in this paper, we propose an efficient privacy-preserving data merging and skyline computation scheme. The scheme allows different data owners to merge data in a privacy-preserving manner and provides skyline computation query service to client users. We combine the leftist tree, deployed to represent the data, with public key encryption technique and index skyline computation technique to achieve both secure data merging and skyline computation. Specifically, the main contributions of this paper are as follows.

1. We propose an efficient privacy-preserving data merging technique by integrating the leftist tree with public key encryption. In our scheme, data are considered to be multi-dimensional and data in each dimension can be represented by a leftist tree. When a data owner intends to merge its local data with the data in the cloud, it first builds an encrypted leftist tree for the data in each dimension and outsources a set of encrypted leftist trees to the cloud together. The cloud server can merge the outsourced trees with the trees in the cloud in a privacy-preserving way. At the time of this research, this is the first scheme to achieve privacy-preserving data merging with $O(\log_2(n_1+n_2))$ computational complexity for merging two leftist trees of size n_1 and n_2.
2. To facilitate both data merging and skyline computation, we design a non-interactive data comparison protocol over encrypted data. It consists of an encryption algorithm and a data comparison algorithm. The encryption algorithm can be used to achieve data privacy protection while storing data in the cloud, and the data comparison protocol can be used to compare two encrypted data in a non-interactive manner during the process of data merging and skyline computation.

3. We propose a privacy-preserving index and leftist tree based skyline computation algorithm, which allows client users to query the skyline over the encrypted data. Specifically, we integrate the leftist tree and data comparison protocol with the index based skyline computation algorithm [24] to achieve skyline computation in our scheme. In addition, we combine the skyline computation algorithm with data comparison protocol to achieve skyline computation with data privacy.

The remainder of this paper is organized as follows. Section 2 introduces the system model, security model and design goals. In Section 3, we review related background materials. In Section 4, we present the proposed index and leftist tree based skyline computation algorithm, data comparison protocol and privacy-preserving data merging and skyline computation scheme. Security analysis and performance evaluation are introduced in Sections 5 and 6, respectively. Related work is briefly discussed in Section 7. Finally, we conclude this paper in Section 8.

2. Models and design goals

In this section, we formalize the system model, security model, and design goals considered in this paper.

2.1. System model

Our scheme comprises four types of entities, namely: a service provider, a set of data owners, a cloud server and multiple client users. We then use healthcare as a use case – see Fig. 1.

• *Service provider:* In this paper, we consider a trusted third party to be a service provider, which is responsible for bootstrapping the entire scheme in the system initialization. Specifically, it will generate both public and private keys according to the parameters' setting, and distribute them to each type of entity. With these keys, data privacy can be ensured during the execution of our proposed scheme.

• *Data owners:* We assume that organizations, such as hospitals, generally do not have sufficient data to perform data mining on their own. Each of these data owners has a dataset and they are willing to merge their data with other data owners when data privacy can be guaranteed. In order to preserve data privacy, they tend to encrypt their data before outsourcing them to the cloud.

• *Cloud server:* We deploy a cloud server to store and process the data in the cloud, which has significant storage space and powerful computing capability. It is mainly tasked with merging of encrypted data from different data owners and processing of skyline query requests from client users.

• *Client users* $\mathcal{U} = \{U_1, U_2, \cdots\}$*:* Each client user $U_i \in \mathcal{U}$ must be authorized by the service provider before accessing the skyline query service. In other words, only authorized users can receive and recover skyline queries after sending a skyline query request to the cloud.

2.2. Security model

In our security model, it is crucial to ensure the data privacy of the scheme. The service provider is considered to be trusted, but the data owners and cloud server are considered to be *honest-but-curious* (i.e., these entities will follow the protocol specification, but they are curious to learn others' private information). Unauthorized client users are also likely to launch attacks, in order to access skyline services in the cloud. In this paper, we do not consider other attacks, passive or active (e.g., denial of service and data pollution attacks).

2.3. Design goals

In this paper, we aim to design an efficient privacy-preserving data merging and skyline computation scheme under our system model and security model. The following goals should be satisfied.

• *Correctness:* Data from different owners should be correctly merged and the cloud server should compute the accurate skyline to respond to the requests from the client users. Besides, when a client user receives the skyline query result, he/she can recover the correct data by decryption.

• *Data confidentiality:* This is a fundamental requirement that data is not available to unauthorized entities. Note that both data stored in the cloud and the skyline query result (returned to authorized users) are considered to be private data.

• *Access control of query service:* The skyline query service can be provided by the cloud server only if the current user is authorized by the service provider.

• *Computation efficiency:* The proposed scheme should be as efficient as possible, and the computational complexity of data merging and skyline computation should be minimal.

3. Preliminaries

In this section, we introduce relevant background information about the leftist tree and the skyline computation.

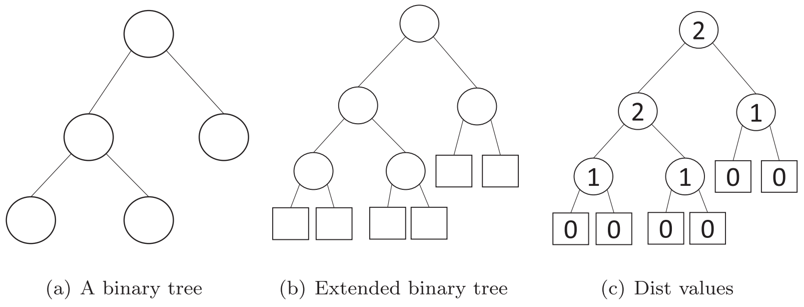


Fig. 2. Related concepts of leftist tree.

3.1. Leftist tree technique

The leftist tree, first proposed by Crane in [6], can be seen as a kind of mergeable priority queue. The leftist tree allows one to merge two trees with $O(\log_2 N)$ computational complexity, and each insertion or deletion takes $O(\log_2 N)$ time, where N is the total number of data records. Besides, finding a minimum value needs $O(1)$ time.

We first formalize the definition of the leftist tree. For a binary tree, it can be extended by replacing the empty node with a kind of new node, called external node. The non-empty nodes are called internal nodes. As shown in Fig. 2, a binary tree in Fig. 2(a) can be extended to the extended binary tree in Fig. 2(b), where the nodes with round shape represent internal nodes, and the rectangular nodes denote external nodes. Suppose that x is a node in the extended binary tree, and *left*(x) and *right*(x) represent the left child and right child of x respectively. Let *dist*(x) denote the shortest path from x to the external nodes. According to the definition of *dist*(x), if x is an external node, $dist(x) = 0$. If x is an internal node, $dist(x) = \min\{dist(left(x)), dist(right(x))\} + 1$. As shown in Fig. 2(c), the value on each node is its *dist* value. Without loss of generality, we use the min leftist tree in the following sections by default. That is, the data contained in each node is less than or equal to the data in that node's children as defined below.

Leftist tree [6]: A binary tree is a leftist tree if the following properties are satisfied.

(1) Heap-ordered: For any internal node x, the key value *key*(x) is less than the key values of both left child and right child. That is,

$$key(x) \le min\{key(left(x)), key(right(x))\}.$$

(2) Height-biased: For any internal node x, the *dist* value of the left child is greater than or equal to the *dist* value of the right child. That is,

$$dist(left(x)) \ge dist(right(x)).$$

Suppose that D denotes a one-dimensional dataset and it can be represented by a leftist tree T. Suppose that each tree node has five attributes, (*data, dist, left, right, parent*), where *data, dist, left, right* and *parent* respectively denote the key value of the node, the *dist* value, the pointers of left child, right child and parent node. In order to facilitate the narrative, we use *x.data, x.dist, x.left, x.right* and *x.parent* to denote the attributes of x respectively in the following algorithms.

Next, we recall related algorithms on the leftist tree, including tree merging, tree building, minimum deletion, insertion and deletion. We first introduce the tree merging algorithm, which is the basis of other algorithms.

- Tree Merging: Suppose that T_A, T_B denote two leftist trees. T_A and T_B can be merged as Algorithm 1.
- Tree Building: Given a one-dimensional dataset D, where the size of D is N, namely, $|D| = N$, a leftist tree can be built as follows. This is also shown in Algorithm 2.
  (1) For each data $x \in D$, build a leftist tree node as (x, 1, *null, null, null*) and each node can be regarded as a leftist tree. Thus, we have N leftist trees.
  (2) Put the N leftist trees into a first-in-first-out queue Q.
  (3) Take two leftist trees from the head of the queue and merge them together. Finally, put the merged tree to the tail of the queue.
  (4) Repeat step (3) until only one element left in the queue and return it.
- Insertion: If data x needs to be inserted to the leftist tree T, we first build a tree node for x, namely, (x, 1, *null, null, null*) and then merge this node with T as shown in Algorithm 2.
- Minimum Deletion: If the minimum value, namely the root node of the leftist tree T needs to be deleted, we just merge the left child and right child of the root node as shown in Algorithm 2.
- Deletion: If $x \in D$ needs to be deleted, we first find the deleted node and its parent node by traversing, and then merge the left child and right child of the deleted node. Finally, adjust *dist* values of related nodes as shown in Algorithm 3.

Algorithm 1 MergeTree(Node T_A, Node T_B).

```
1: if $T_A$ == null then
2:     return $T_B$
3: if $T_B$ == null then
4:     return $T_A$
5: if $T_A.data > T_B.data$ then
6:     swap($T_A, T_B$)
7: $T$ = MergeTree($T_A.right$, $T_B$)
8: $T_A.right = T$
9: $T.parent = T_A$
10: if $T_A.right.dist > T_A.left.dist$ then
11:     swap($T_A.left, T_A.right$)
12: if $T_A.right == null$ then
13:     $T_A.dist = 1$
14: else
15:     $T_A.dist = T_A.right.dist + 1$
16: return $T_A$
```

Algorithm 2 Tree building, Insertion and Minimum deletion.

```
1: function BuildTree(Dataset $D$, Queue $Q$)
2:     if $D$ == null then
3:         return null
4:     for $data$ in $D$ do
5:         $Q.enqueue((data, 1, null, null, null))$
6:     while size($Q$) $>$ 1 do
7:         $T_A = Q.dequeue()$
8:         $T_B = Q.dequeue()$
9:         $T$ = MergeTree($T_A$, $T_B$)
10:        $Q.enqueue(T)$
11:    return $T$
12: function Insertion(int $x$, Node $T$)
13:    $T_A$ = $(x, 1, null, null, null)$
14:    $T$ = MergeTree($T_A$, $T$)
15:    return $T$
16: function DeleteMin(Node $T$)
17:    $t = T.data$
18:    $T$ = MergeTree($T.left$, $T.right$)
19:    return $t$
```

Algorithm 3 Delete(int x, Node T).

```
1: $T_A$ = Find($x$, $T$) //the node that $data == x$
2: $T_B$ = $T_A.parent$
3: $T_A$ = MergeTree($T_A.left$, $T_A.right$)
4: if $T_B$ != null and $T_B.left.data == x$ then
5:     $T_B.left = T_A$
6: if $T_B$ != null and $T_B.right.data == x$ then
7:     $T_B.right = T_A$
8: $T_A.parent = T_B$
9: while $T_B$ != null do
10:    if $T_B.left.dist < T_B.right.dist$ then
11:        swap($T_B.left, T_B.right$)
12:    if $T_B.right.dist + 1 == T_B.dist$ then
13:        break
14:    $T_B.dist = T_B.right.dist + 1$
15:    $T_A = T_B$
16:    $T_B$ = $T_A.parent$
17: return $T$
```

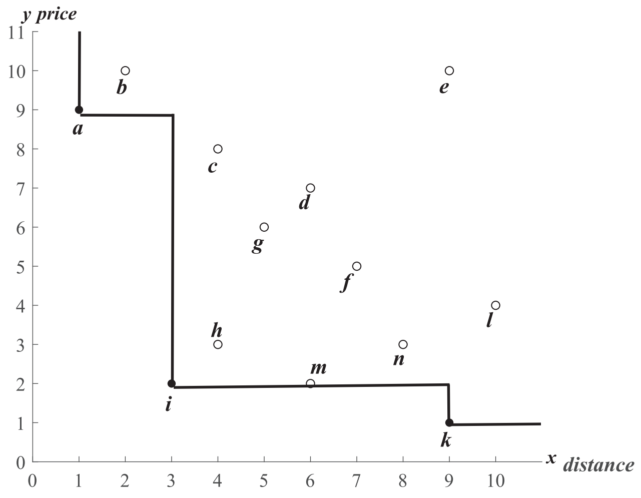


Fig. 3. A skyline example.

3.2. Skyline computation technique

Skyline computation is to compute a set of objects that are not dominated by any other objects. We take a common case in the literature as an example to explain the skyline, also shown in Fig. 3. Suppose that we have a set of hotels with corresponding prices and distances from the beach. The most interesting hotels are *a, i, k*, because there is no other better point in both dimensions. Next, we introduce the definition of the dominant point.

Dominant point: Let $P = (x_1, x_2, \cdots, x_l)$ and $Q = (y_1, y_2, \cdots, y_l)$. P dominates Q if and only if $x_i \le y_i$ for any $i = 1, 2, \cdots, l$, and there exists at least one i such that $x_i < y_i \, (1 \le i \le l)$.

A number of skyline computation algorithms, such as block nest loop (BNL) [3], sort first skyline [5], divide-and-conquer [3], bitmap [24], index [24] and nearest neighbor [15], have been proposed in the literature. Of these algorithms, BNL is the most straightforward one, which computes the skyline by comparing one point with other points directly and selecting the points that not dominated by any points. Specifically, at first, one point is put in the skyline list. For any other point P, one compares it with the points in the skyline list and there are three cases.

(1) If P is dominated by other points, then drop P.
(2) If P dominates other points, then insert P and drop other points that are dominated by P.
(3) If neither P dominates nor is dominated by other points, then insert P to the skyline list.

4. Our proposed scheme

In this section, we will present our data merging and skyline computation scheme. Before delving into the details of the proposed scheme, we first introduce an index and leftist tree based skyline computation algorithm (see Section 4.1) and a data comparison protocol (see Section 4.2), which are building blocks of our proposed scheme.

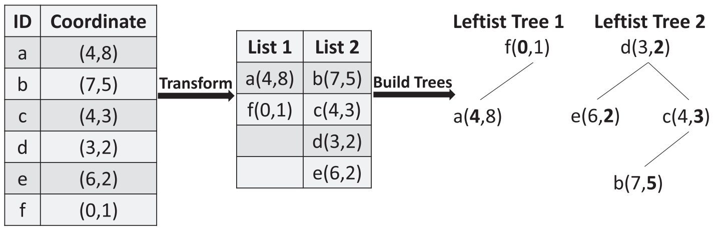
4.1. Index and leftist tree based skyline computation algorithm

The index-based skyline computation algorithm, first proposed by Dimitris et al. in [21], is reportedly the fastest algorithm to return the entire skyline under all parameter settings. Base on this algorithm, we design an index and leftist tree based skyline computation algorithm by using the leftist tree as the index structure. Suppose that D is a dataset with a set of l-dimensional data records. The index and leftist tree based skyline computation algorithm is able to find the skyline of the dataset D step-by-step as follows, also shown in Algorithm 4.

- *Split Dataset*
  We first split the dataset into l lists, where each data record $x = (x_1, \cdots, x_l)$ is assigned to the jth list if and only if x_j is the minimum value among the data in all dimensions. As shown in Fig. 4, data in the first table are a set of two-dimensional data records and can be transformed to two lists in the second table. In general, the dataset will be split into l lists and each list contains about $\frac{N}{l}$ data records on average.
- *Build Leftist Tree*
  We build a leftist tree for each list according to the values in that dimension. For example, in Fig. 4, we use the data in the first coordinate as the key values to build the leftist tree 1 for list 1.

Algorithm 4 SkylineIndex(Dataset D).

```
1: Transform $D$ to $l$ lists
2: Build leftist tree for each list as $\{T_i\}_{i=1}^{l}$
3: $deletedPoint = \emptyset$ // the set of deleted points
4: for $i = 1$ to $l$ do
5:     $f_i$ = true // the $i$th tree needs to be searched
6:     $t_i = T_i.data$, $max_i$ = maxValue($t_i$), $min_i$ = minValue($t_i$)
7: $mn$ = min$\{min_i | i = 1, \cdots, l\}$, $mx$ = min$\{max_i | i = 1, \cdots, l\}$
8: $j = 1$, $P_j = \emptyset$ // a temporary set for data points
9: $S = \emptyset$ // the skyline
10: while (true) do
11:     for $i = 1$ to $l$ do
12:         if $mx < min_i$ then
13:             $f_i$ = false // prune $i$th leftist tree
14:     if (there exists a $i$, such that $f_i$ == true) then
15:         for $i = 1$ to $l$ do
16:             $t_i = T_i.data$ // the data of root node in the $i$th leftist tree
17:             while minValue($t_i$) == $mn$ do
18:                 $mx$ = min$\{mx$, maxValue($t_i$)$\}$
19:                 $P_j = P_j \cup t_i$, $deletedPoint = deletedPoint \cup t_i$
20:                 $t_i$ = DELETEMIN($T_i$) // $T_i$'s next minimum value
21:             $min_i$ = minValue($t_i$)
22:         $S_j$ = BNLSkylineComputation($P_j$) // BNL algorithm
23:         $S$ = BNLSkylineComputation($S_j$, $S$)
24:         $j = j + 1$, $P_j = \emptyset$, $mn$ = min$\{min_i | i = 1, \cdots, l\}$
25:     else
26:         break
27: Build $l$ trees for deleted points, namely, the set of $deletedPoint$
28: Merge leftist trees of deleted points to original leftist trees
29: return $S$
```


ID	Coordinate
a	(4,8)
b	(7,5)
c	(4,3)
d	(3,2)
e	(6,2)
f	(0,1)

List 1	List 2
a(4,8)	b(7,5)
f(0,1)	c(4,3)
	d(3,2)
	e(6,2)

Fig. 4. Example of dataset splitting and leftist trees building.

- *Skyline Computation*
  From two leftist trees in Fig. 4, we can observe that (a) to a large extent, dominant points are at the top of each tree; For example, (3,2) is the dominant point in the leftist tree 2. (b) leftist tree T_2 can be pruned if the maximum value at root node in leftist tree T_1 is less than the minimum value at root node in leftist tree T_2; For example, the maximum value at root node $f(0, 1)$ in leftist tree 1 is 1, which is less than the minimum value at root node $d(3, 2)$ in the leftist tree 2. Thus, $f(0, 1)$ dominates all points in leftist tree 2, and the leftist tree 2 can be pruned.

Based on the above observations, we first consider the node with the minimum value in each leftist tree, i.e. the root node, is the skyline point and whether it can be pruned or not. Then we continue to consider the next minimum value of each leftist tree until all leftist trees have been pruned. The detailed procedure refers to Algorithm 4. Note that f_i is the tag to demonstrate whether the ith leftist tree can be pruned and maxValue(t_i) (minValue(t_i)) is to compute the maximum (minimum) value of data points t_i. The function DELETEMIN($\cdot$) in Algorithm 2 is used to compute the next minimum value of each leftist tree. BNLSkylineComputation(P_j) is used to compute the skyline of P_j by BNL algorithm and T_i denotes the ith leftist tree ($i = 1, 2, \cdots, l$). Besides, to guarantee the invariance of the tree structure after skyline computation, we merge the deleted points to the original tree at last.

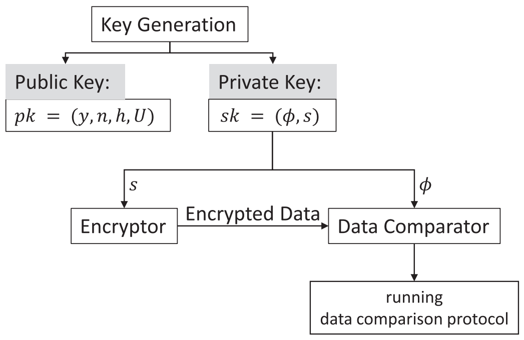


Fig. 5. Settings for data comparison protocol.

4.2. Data comparison protocol

In this subsection, we propose an encrypted data comparison protocol based on Benaloh public key encryption [2]. The purpose of our protocol is to achieve data comparison over encrypted data without leaking data privacy. Our protocol is comprised of two kinds of entities, namely, encryptor and data comparator. Encryptor encrypts data and data comparator is able to compare any two encrypted data without decryption. Based on the aforementioned purpose, the main idea of our data comparison protocol is to add a random number as the private key, which is distributed to the encryptor and kept secret to the data comparator. Then, the data comparator cannot decrypt the data without this private key. As shown in Fig. 5, part of private key s is only distributed to encryptor, without which data comparator cannot decrypt encrypted data, but it can perform data comparison.

Suppose that the data space is $\mathbb{Z}_r = \{0, 1, \cdots, r-1\}$, where r is a positive integer. Similar to assumption in Benaloh et al.'s scheme [2], we assume that the data space is small, namely, r is small. The proposed data comparison protocol consists of three phases: key generation, encryption and data comparison.

- *Key Generation*

In the data comparison protocol, the public key and the private key can be generated as follows.

First, select two large primes p, q such that $r|(p-1)$, $gcd(r, (p-1)/r) = 1$ and $gcd(r, q-1) = 1$, and set $n = pq$, $\phi = (p-1)(q-1)$.

Second, select $y, s \in \mathbb{Z}_n^*$ such that $y^{\phi/r} \neq 1 \bmod n$, $gcd(s, r) = 1$ and $gcd(s, \phi/r) = 1$, and choose a hash function $h(\cdot)$ at random.

Third, set $U = \{h(y^{s\phi i/r} \bmod n) | 0 < i < r\}$. Due to the fact that $y^{\phi/r} \neq 1 \bmod n$, $gcd(s, r) = 1$ and $gcd(s, \phi/r) = 1$, the set of $\{y^{s\phi i/r} \bmod n | 0 < i < r\}$ contains $r-1$ distinct values. For any i, it satisfies $0 < i < r$ if and only if $h(y^{s\phi i/r} \bmod n)$ belongs to U.

Finally, as shown in Fig. 5, publish the public key $pk = (y, n, h(\cdot), U)$. Then, divide the private key $sk = (s, \phi)$ into two parts, i.e., s and ϕ, which will be distributed to the encryptor and the data comparator respectively. Note that different from the common sense that encryptor only holds the public key, part of private key s is distributed to the encryptor and kept as a secret to the data comparator.

- *Encryption*

Given a message $m \in \mathbb{Z}_r$, the public key pk and part of private key s, the message m can be encrypted as $c = E(m, u) = y^{sm} u^r \bmod n$, where $u \in \mathbb{Z}_n^*$ is a random number.

- *Data Comparison*

Suppose that $m_1, m_2 \in \mathbb{Z}_r$, $E(m_1) = E(m_1, u_1) = y^{sm_1} u_1^r \bmod n$ and $E(m_2) = E(m_2, u_2) = y^{sm_2} u_2^r \bmod n$. Given two encrypted data $E(m_1)$ and $E(m_2)$, public key pk and part of the private key ϕ, the data comparator is able to compare m_1 and m_2 as follows.

– Compute $c = \frac{c_2^{\phi/r}}{c_1^{\phi/r}} = y^{s\phi(m_2 - m_1)/r} \bmod n$.
– If $c = 1$, $m_1 = m_2$. If $h(c)$ belongs to U, $0 < m_2 - m_1 < r$, namely, $m_2 > m_1$. Otherwise, if $h(c^{-1})$ is in U, $m_1 > m_2$.

Note that compared with the original public key encryption scheme in [2], the new proposed scheme can be only used to conduct data comparison, rather than decryption, since the private key s is distributed to the encryptor. Without the s value, the data comparator cannot recover the plaintext m from encrypted data $E(m)$ due to the discrete logarithm problem.

4.3. The detailed scheme

Here, we will introduce the proposed data merging and skyline computation scheme over encrypted outsourcing data, which consists of five phases, namely: system initialization, outsourcing encrypted data to the cloud, maintaining encrypted data in the cloud, skyline computation and skyline query.

1) System Initialization

In our scheme, the service provider bootstraps the entire scheme. He/She first generates public key $pk = (y, n, h(\cdot), U)$ and private key $sk = (\phi, s)$ according to the key generation algorithm in the data comparison protocol in Section 4.2. Then, he/she chooses a random number ak as the access key and selects AES as the encryption algorithm. After that, he/she publishes the public key pk, distribute ϕ to the cloud server, and distribute s and ak to each data owner. At the same time, the service provider is also responsible for authorizing client users. In specific, a client user U_i with identity id_i can be authorized as the following steps.

Step-1: U_i sends his/her identity id_i to the service provider.

Step-2: In order to authorize U_i, the service provider distributes the access key ak to U_i, and registers U_i in the cloud by sending $AES_{ak}(id_i)$ to the cloud server. All the data transmitted in this step are through secure channels that can be realized by using the public key of the service provider and cloud server.

Step-3: The cloud server stores $AES_{ak}(id_i)$ in the cloud.

2) Outsourcing Encrypted Data to the Cloud

In our scheme, the cloud is *honest-but-curious*. In other words, data owners will encrypt their data prior to outsourcing. Suppose that each data record is an l-dimensional vector plus a recording comment, i.e., $x = (x_1, x_2, \cdots, x_l, x_d)$. Without loss of generality, we assume that each x_i $(i = 1, 2, \cdots, l)$ is an integer, and if not, we can convert real number to integer easily. Suppose that each data owner has a dataset $D = \{(x_1, x_2, \cdots, x_l, x_d) | (x_1, x_2, \cdots, x_l) \in \mathbb{Z}_r^l\}$. According to the dataset, each data owner can build a set of leftist trees, encrypt all the tree nodes, and outsource encrypted leftist trees to the cloud as the following steps.

Step-1: Data owner transforms the dataset D to l lists as stated in Section 4.1. For the ith list, it contains the points that have the minimum value in the ith coordinate, namely, $L_i = \{(x_1, \cdots, x_l, x_d) | (x_1, \cdots, x_l) \in \mathbb{Z}_r^l, x_i \leq x_j (1 \leq j \leq l, j \neq i)\}$.

Step-2: Data owner builds the leftist tree for each list. When the maximum value at root node in leftist tree T_1 is less than the minimum value at root node in leftist tree T_2, pruning can be performed during the skyline computation. Therefore, we add *minLabel* and *maxLabel* to the leftist tree node, which denote the coordinates of the minimum and maximum values. Then, the leftist tree node in our scheme is a seven-tuple (*data, dist, minLabel, maxLabel, left, right, parent*), where attributes *data, dist*, *left*, *right*, and *parent* represent the data value of the node, *dist* value, pointers of left child, right child and parent node respectively. Note that the leftist tree in the preliminaries is one dimensional. Although the data in our scheme is multi-dimensional, the data owner actually refers to the values in the ith coordinate to build the ith leftist tree. As a result, the data owner can also build each leftist tree as function BUILDTREE$(\cdot)$ in Algorithm 2.

Step-3: Data owner encrypts each leftist tree, i.e., replacing each data record in the leftist tree with encrypted data record. Specifically, a data record $x = (x_1, \cdots, x_l, x_d)$ will be encrypted as $E(x) = (E(x_1), \cdots, E(x_l), AES_{ak}(x))$, where $E(x_i)$ denotes that x_i is encrypted by the encryption algorithm in the data comparison protocol, and $AES_{ak}(x)$ denotes that x is encrypted by encryption algorithm *AES* and encryption key *ak*.

Step-4: Data owner outsources l encrypted leftist trees to the cloud.

3) Maintaining Encrypted Data in the Cloud

In this phase, we consider how the cloud server maintains the leftist trees in the cloud, including data merging, insertion and deletion.

Data merging: When a data owner intends to merge its local data to the data in the cloud, the cloud server runs the data merging algorithm. Suppose that there exist l leftist trees in the cloud, and the ith leftist tree contains the data records that have the minimum value in the ith coordinate among all dimensions. When receiving the data merging request and l encrypted leftist trees from the data owner, the cloud server will merge the ith uploaded leftist tree with the ith leftist tree in the cloud according to the key values in the ith dimension respectively. The merging algorithm refers to Algorithm 1, where the data comparison protocol can be used to achieve data comparison over encrypted data. After each leftist tree is merged, the data merging is finished.

Insertion: When a data owner plans to insert a new data record to the data in the cloud, he/she first encrypts the new data record and outsources it to the cloud. Upon receiving the outsourced data record, the cloud server will insert it to the *minLabel*th leftist tree, where *minLabel* is the coordinate of the minimum value in the inserted data record. The insertion algorithm is shown in Algorithm 2.

Deletion: When a data owner intends to delete one obsolete data record from the cloud, it first encrypts the data record and outsources it to the cloud. Suppose that *minLabel* is the minimum value in the deleted data record. After receiving the encrypted data record, the cloud server finds the *minLabel*th leftist tree and deletes the node as the deletion algorithm, shown in Algorithm 3.

4) Skyline Computation

The skyline is computed by the cloud server using the index and leftist tree based skyline computation algorithm (see Algorithm 4), and it should be updated as the data in the cloud are updated. In the following, we consider data merging, insertion and deletion operations, and discuss how to update skyline in different cases.

Data merging: Since our proposed scheme has a significant advantage over data merging, data owners can outsource large amounts of data to the cloud at any time. The cloud server will conduct data merging and recalculate the skyline using the index and leftist tree based skyline computation algorithm, upon receiving the outsourced data.

Insertion: The data owner may have a small amount of important or urgent data that need to be outsourced to the cloud immediately. In this case, the cloud server will insert the outsourced data to the leftist trees in the cloud one-by-one using Algorithm 2, and update the skyline using the BNL skyline computation algorithm.

Deletion: If a data owner intends to delete one data record stored in the cloud, we consider two cases: (a) if the deleted data record is not in the skyline, delete it directly; and (b) if it is in the skyline, the cloud server has to recalculate the skyline on the l leftist trees after deleting the data record using Algorithm 4. This is because when a data record in the skyline is deleted, those data records that are dominated by it will have no dominant data record in the skyline.

5) Skyline Query

A client user U_i can enjoy the skyline query service as the following steps.

Step-1: U_i uses his/her access key ak to encrypt his/her identity id_i as $AES_{ak}(id_i)$, and sends the skyline query request together with $AES_{ak}(id_i)$ to the cloud server via a secure channel that can be realized by using the cloud server's public key.

Step-2: On receiving the skyline query request, the cloud server first checks whether U_i is authorized or not. In specific, it will check whether the service provider registers $AES_{ak}(id_i)$ in the cloud or not. If not, drop this skyline query request. Otherwise, the cloud server will directly return the encrypted skyline to U_i, because the encrypted skyline has been computed each time the data are updated. Note that for any encrypted data record $E(x) = (E(x_1), \cdots, E(x_l), AES_{ak}(x))$ in the skyline, the cloud server only returns $AES_{ak}(x)$ to U_i instead of $E(x)$.

Step-3: On receiving the encrypted skyline, U_i recovers each data record x in the skyline from $AES_{ak}(x)$ using the access key ak.

5. Security analysis

In this section, we analyze the security of the proposed scheme. According to our design goal, our proposed scheme should satisfy two security properties, i.e., data confidentiality and access control of query service.

5.1. Data confidentiality

We will now demonstrate that our scheme can achieve data confidentiality. Specifically, under our security model, the cloud server and other unauthorized users are not aware of the data stored in the cloud. Each data owner only knows his/her own outsourced data, but not data outsourced by other data owners.

5.1.1. Data confidentiality at cloud server

Since the data encryption algorithm and the data comparison protocol are the two most basic algorithms that provide data privacy protection for the entire scheme, we will show that both algorithms are privacy-preserving. In other words, both algorithms can ensure that the data-at-rest in the cloud are kept secret from the cloud server.

- *The encryption algorithm is privacy-preserving.* Our encryption algorithm is based on the Benaloh public key encryption algorithm [2], but there is a slight difference between both algorithms. In the original algorithm, the ciphertext c for message m is $E(m, u) = y^m u^r \bmod n$. Since the data space $\mathbb{Z}_r$ is small, it is easy to recover m by the exhaustive algorithm using private key ϕ. In our encryption algorithm, we additionally choose a random value s, which is the private key of data owners and kept secret to the cloud server. Without s, the cloud server cannot recover the plaintext m from the ciphertext c, but he/she can compute $c^{\phi/r} = y^{sm\phi/r} \bmod n$ using his/her private key ϕ. From the key generation phase, $y^{\phi/r} \neq 1 \bmod n, gcd(s, r) = 1$ and $gcd(s, \phi/r) = 1$. Hence, $y^{sm\phi/r} \neq 1 \bmod n$. Besides, due to $s \in \mathbb{Z}_n^*$, $sm \in \mathbb{Z}_n$, it is computationally infeasible for the cloud server to compute $sm\phi/r$ from y and $y^{sm\phi/r}$ according to the discrete logarithm problem, let alone obtaining m without s. Thus, the encryption algorithm can ensure that data-at-rest in the cloud is kept secret from the cloud server.
- *The data comparison protocol is also privacy-preserving.* When the cloud server runs the data comparison protocol, $c = \frac{c_2^{\phi/r}}{c_1^{\phi/r}} = y^{s\phi(m_2 - m_1)/r}$ will be computed. If $m_1 \neq m_2$ and $0 < m_2 - m_1 < r$, the cloud server cannot recover $m_2 - m_1$, which is similar to the case of recovering m from $y^{sm\phi/r}$ as described above. When m_1 is equal to m_2, $c = 1$. In this case, the equal relationship between m_1 and m_2 will be disclosed. Actually, the purpose of the data comparison protocol is to compare data and the data relationship cannot be regarded as private information to some degree. For practical applications, if the equal relationship between data is confidential, an alternative way is to add a negligible number to change their relationship. For example, suppose that $x = (x_1, x_2, \cdots, x_l, x_d), y = (y_1, y_2, \cdots, y_l, y_d)$, if $x_i = y_i$, we can set $x_i' = x_i + \varepsilon$, such that $x_i > y_i$, where ε is a negligible number. However, in order to guarantee that the data decrypted by the users are correct, x in $AES_{ak}(x)$ remains unchanged. From the above analysis, the data comparison protocol can ensure that data-at-rest in the cloud is kept secret from the cloud server.

5.1.2. Data confidentiality at data owner

In our scheme, data are horizontally distributed in different data owners and the data outsourced by each data owner only account for a small portion of all data in the cloud. In order to protect data confidentiality, each data owner should have no idea of the data from other data owners but are stored in the (same) cloud server. On the one hand, the cloud is *honest-but-curious*, i.e., it will not collude with others, so it is computationally challenging for the data owner to obtain such kind of data from the cloud server. On the other hand, even if the data owner gets the data, (s)he cannot recover the plaintext data without the decryption key because data are encrypted. Therefore, each data owner has no idea of the data outsourced by other data owners.

5.1.3. Data confidentiality at other unauthorized entities

In our scheme, all encrypted data are stored in the cloud. Under our security model, the cloud server is *honest-but-curious*, so it is computationally challenging for unauthorized entities to access the encrypted data from the cloud server. Besides, our encryption algorithm is based on the provably-secure Benaloh encryption technique [2]. As a result, the unauthorized entities still cannot recover the plaintext data even if they acquire a few data records, say by malicious attacks.

5.2. Access control of query service

In this section, we will show that the skyline query service can be provided only if the service provider authorizes the current user. On the one hand, the skyline query service from unauthorized entities will be declined by the cloud server. We also assume that unauthorized entities have the ability to impersonate an authorized user to launch a valid skyline query request and receive the skyline query result from the cloud. In this case, they also cannot recover the skyline query result due to the security of our encryption algorithm. Therefore, only authorized users can enjoy the skyline query service.

6. Performance evaluation

In this section, we evaluate the performance of the proposed scheme.

6.1. Theoretical analysis

We will now analyze the computational complexity of the proposed scheme from the perspective of data owners, cloud server and client users.

6.1.1. Computational complexity of data owner

The computational complexity of the data owners comes from building the leftist trees and encrypting the data records. Suppose that a data owner has a dataset that contains N l-dimensional data records. The dataset has been split to l lists and each list contains $\frac{N}{l}$ points on average. For the data owner, it will take $O(\frac{N}{l})$ to build the leftist tree for each list and the overall computational complexity of building l leftist trees is $O(N)$. According to the encryption algorithm in the data comparison protocol, encrypting an l-dimensional data record involves $2l$ modular exponentiations, $2l$ modular multiplications and one AES encryption. Let C_e, C_m and C_{AES} respectively denote the complexity of modular exponentiation, modular multiplication (division) and one AES encryption. The computational complexity of encrypting N data records is $N(2lC_e + 2lC_m + C_{AES})$. Therefore, the overall computational complexity for a data owner includes two parts. The first part deals with the leftist trees building with $O(N)$ computational complexity. The second part deals with the data encryption with $N(2lC_e + 2lC_m + C_{AES})$ computational complexity.

6.1.2. Computational complexity of cloud server

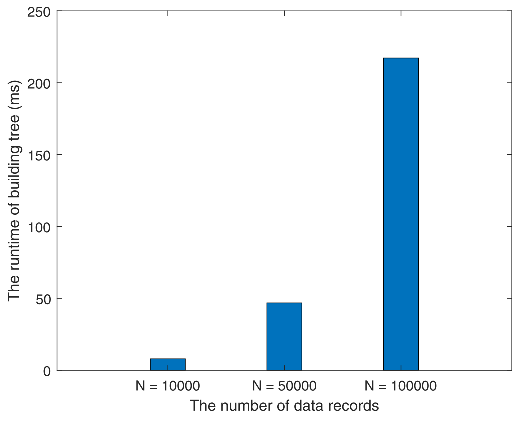
The computational complexity of the cloud server is from data merging and skyline computation. Both depend on the data comparison protocol, in which the computational complexity of comparing two encrypted data $x, y \in \mathbb{Z}_r$ is $2C_e + C_m$. For the data merging, the cloud server merges the existing ith leftist tree in the cloud with the outsourced ith leftist tree from the data owner. The computational complexity of merging two trees is $\log_2 \frac{N}{l}(2C_e + C_m)$ if the two merged trees contain $\frac{N}{l}$ data records in total. Suppose that there are $\frac{N}{l}$ data records on average in the ith merged leftist tree, where $1 \leq i \leq l$. Then, the overall computational complexity for merging both sets of leftist trees for all dimensions is $l * \log_2 \frac{N}{l}(2C_e + C_m)$. However, for the skyline computation, the computational complexity heavily depends on the datasets. In the best case, we just need to search the data at root node in each leftist tree, but in the worst case, we may need to traverse the whole tree.

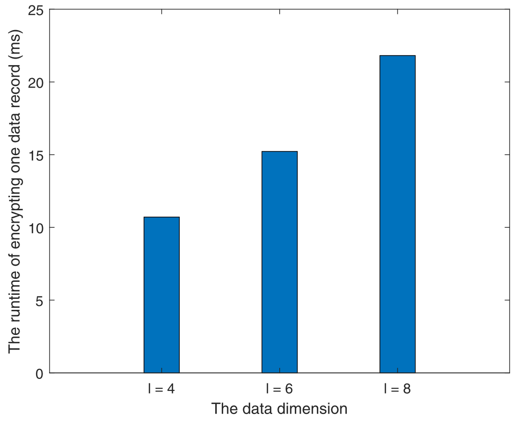
6.1.3. Computational complexity of client user

When client users send a skyline query request to the cloud server, they just need to decrypt the receiving skyline using the access key ak; thus, the computational complexity is C_{AES}.

6.2. Experimental analysis

In this section, we experimentally evaluate the efficiency of the proposed scheme.





(a) Runtime of building leftist trees with different N

(b) Runtime of encrypting one data record with different l

Fig. 6. Performance evaluation at the data owner.

6.2.1. *Experimental setting*

We implemented our scheme in Java and each experiment in this paper was performed on a machine with Windows 10 operating system, Intel(R) Core(TM) i7-3770CPU 3.40Hz processor, and 16GB RAM. We used the encryption algorithm and data comparison algorithm in Section 4.2 to encrypt data and achieve data comparison over encrypted data. The upper bound of data space r was set to be 10^6, and security parameters p and q were set to be 512 bits large primes. Thus, the public key, $n = pq$, is a 1024 bits big integer. The key size of AES is 128 bits and the hash function $h(\cdot)$ is "SHA-512". Besides, each runtime in this paper is the average value of multiple experiments.

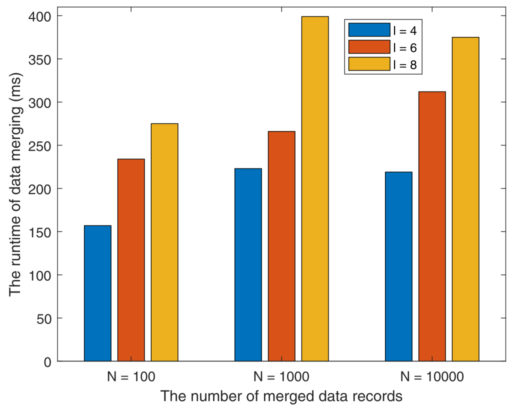
We used synthetic datasets to evaluate the performance of our scheme. These datasets satisfy the following key characteristics: i) Data in different dimensions are in the same domain. ii) Data are always integers. Note that both of them are easy to meet in practical applications. For example, condition (i) can be achieved by standardization and condition (ii) is much easier because any data can be converted to integers.

6.2.2. *Experimental results*

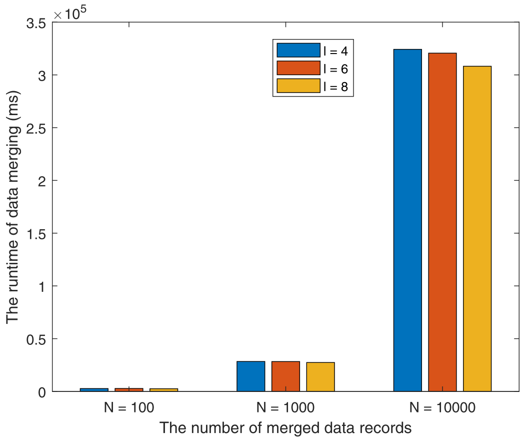
We now present the experimental results of our scheme from the perspective of data owner and cloud server. The computational complexity of the client users is not considered because it is quite small, namely: C_{AES}. Assume that l and N respectively denote the data dimension and the total number of the data records.

- *Data Owner*: In our scheme, the data owner is in charge of data preparation before outsourcing data to the cloud, including building and encrypting a set of leftist trees according to the dataset. According to the computational complexity analysis, the average computational complexity of building l leftist trees for each data owner is $O(N)$ and irrespective of the data dimension l. Thus, we do not take the data dimension into consideration when evaluating the performance of encryption. Fig. 6(a) presents the runtime of building l leftist trees changes with the number of data records N, where l is set to be 6 in our experiment and it can also be set to other values. From Fig. 6(a), the runtime increases as the number of data records increases. For example, the runtimes of building leftist trees are 7.9 ms, 46.8 ms, 217.2 ms with $N = 10000, 50000, 100000$ respectively.
  Next, we evaluate the performance of the encryption algorithm by the runtime of encryption. For simplicity, we only consider the runtime of one data record encryption, which is related to the data dimension l. As shown in Fig. 6(b), the encryption time linearly grows with l and the runtime is less than 25 ms with $l = 8$.
- *Cloud Server*: For the cloud server, it mainly processes data updating, such as data merging, insertion, deletion and skyline computation. Due to the fact that insertion and deletion algorithms are based on the data merging algorithm and they have the same performance. Thus, we only evaluate the performance of data merging and skyline computation.

The efficiency of the data merging was evaluated by comparing two different algorithms. One algorithm is the data merging algorithm in our scheme and the other one is an algorithm that merges data in a common way, i.e., inserting one-by-one. Suppose that there were 100,000 encrypted data records in the cloud, which had been represented by a set of leftist trees. The data owner attempted to merge a large amount of local data with the existing dataset in the cloud respectively. Upon receiving the data merging request and outsourced leftist trees from the data owner, the cloud server would merge the received trees with existing trees in the cloud. The computational complexity of data merging is $l * \log_2 \frac{N}{l}(2C_e + C_m)$, so the runtime of data merging changes with the number of data records N and the dimension l. Figs. 7(a) and (b) present the



(a) Runtime of data merging in our scheme



(b) Runtime of data merging by one-by-one insertion

Fig. 7. Efficiency comparison of different data merging algorithms.

performance of data merging algorithm in our scheme and the other algorithm that inserts data one-by-one, respectively. From Fig. 7(a), the runtime of data merging logarithmically increases with the number of data records N for a fixed l and linearly grows with the data dimension l for a specific N, but the overall runtime of data merging is not very large (i.e., less than 400 ms). For the other data merging algorithm, Fig. 7(b) presents the runtime of data merging by inserting data one-by-one. Compared with data merging in our scheme, this algorithm is inefficient and the runtime in this case is significantly large. For example, when $N = 1000$ and $l = 4$, the runtime of data merging by this approach is about 28378 ms, while that is only 223 ms by data merging algorithm in our scheme.

As far as the evaluation of the skyline computation, Tan et al. [24] pointed out that the performance of the index based skyline computation is associated with the type of the dataset. Similar to Tan et al. [24], we considered three kinds of datasets, namely: (i) Independent: data in different dimensions are independent. (ii) Correlated: the value of data is small in one dimension and also small in other dimensions. (iii) Anti-correlated: the value of data is small in one dimension but large in one or all other dimensions.

Fig. 8(a)–(c) present the runtimes of skyline computation under different N and l with respect to different datasets. For each dataset, the runtime generally increases with the number of data records N and the dimension l, while the overall runtime has a significant difference among the three kinds of datasets. The correlated dataset has the best performance, followed by the independent dataset, and anti-correlated is the worst. For instance, the overall runtime of the correlated dataset is less than 1000 ms when $l = 2, 3, 4$ and $N = 300, 500, 1000,$ while those of independent and anti-correlated datasets are less than 1.1×10^5 ms and 5.5×10^6 ms respectively. Thus, our skyline computation algorithm performs better in the correlated dataset.

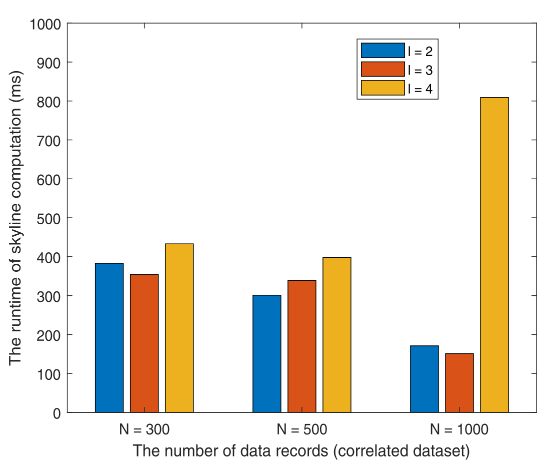
7. Related work

In this section, we briefly review related works on data merging and skyline computation.

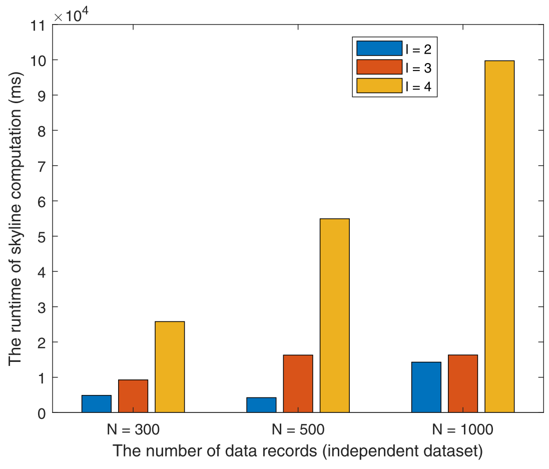
Data merging: With the increasing requirement to data scale in data mining and analysis, it is important to be able to merge data from different data sources, especially for those domains that data are scarce and distributed. Binary tree, B+ tree, heap and other tree data structures are widely used to represent data in database systems. Data merging, one of the basic algorithms in tree updating, can be used to merge two trees. However, there is no specialized algorithm to achieve data merging for these tree data structures, with the exception of some mergeable heaps.

We take the binary search tree as an example to explain two common approaches to merge a binary search tree of n_1 elements with another tree of n_2 elements, namely: (a) insert n_1 elements to the tree of n_2 elements one-by-one according to the insertion algorithm [11]; and (b) neglect original data structure and reconstruct a new binary search tree [23]. Both of them are not efficient. In 1972, Crane et al. [6] proposed a new data structure, leftist tree, which is a kind of mergeable heap and can achieve data merging with $O(\log_2 N)$ computational complexity. Subsequent to this work, binomial heaps [4] and Fibonacci heaps [10] were proposed to achieve (more) efficient data merging. However, to our best knowledge, security issues relating to data merging algorithms are not considered in current studies.

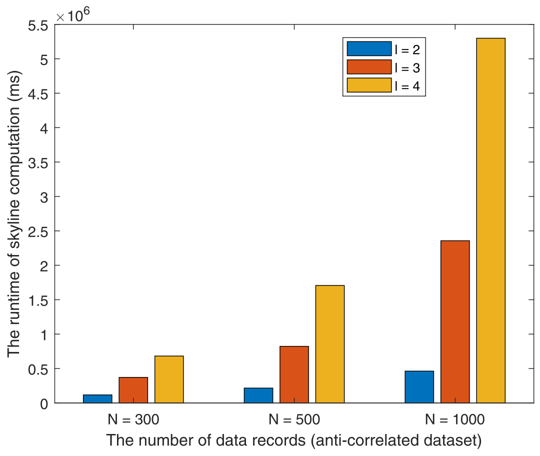
Skyline computation: Recently, skyline computation has gained attention from the industry and academia, because of its potential application on multiple targets decision, data mining and data visualization, etc., and it has been extensively studied [7,22,25,29]. In 1975, Kung et al. [16] proposed the first skyline computation algorithm, which is based on the divide



(a) Correlated dataset



(b) Independent dataset



(c) Anti-correlated dataset

Fig. 8. Runtime of skyline computation in different kinds of datasets.

and conquer principle and is quite complex. Since then, skyline computation has been widely studied and multiple related algorithms have been proposed. These algorithms can be broadly divided into two categories, namely: centralized algorithms and distributed algorithms, according to the skyline computation environment.

Centralized skyline computation is to compute skyline in a centralized database. The skyline computation scheme in [16] is also a centralized skyline computation scheme. Börzsönyi [3] extended the divide and conquer scheme of Kung et al., in order to deal with large database. They also proposed a new algorithm block-nested-loops approach. However, that paper only considered batch-oriented skyline algorithms. Tan et al. proposed two progressive skyline computation algorithms, bitmap and index [24]. Index is the fastest skyline computation algorithm under all parameters' settings [21]. In 2002, Kossmann et al. proposed a nearest neighbor approach based on R tree [15]. Lee et al. proposed skyline computation using ZBtree, storing data based on a Z-order curve [17]. However, security issues are seldom considered.

Distributed skyline computation is to compute skyline for objects that are distributed vertically or horizontally over data sources. In an online web-based application, data are often vertically distributed, e.g. *price* information residing in hotels.com and *distance* in maps.com, as typically assumed in [1,19]. For example, Balke et al. [1] proposed a distributed skyline computation, and Lo et al.[19] proposed a progressive skyline computation under the same model. In peer to peer scenarios, data are often horizontally distributed, as assumed in studies of [12,27,28,30].

8. Conclusion

In this paper, we proposed an efficient privacy-preserving data merging and skyline computation scheme over encrypted and outsourced data. Specifically, in our scheme, we designed a data comparison protocol that is able to securely compare encrypted data in a non-interactive manner. In order to achieve efficient privacy-preserving data merging, we used leftist tree as the underlying data structure to represent data and used encryption algorithm in our data comparison protocol to encrypt the data before outsourcing to the cloud. In order to achieve efficient skyline computation, we proposed an index and leftist tree based skyline computation algorithm, which is able to compute skyline over encrypted data. We then examined the security and performance of our scheme, and demonstrated its utility.

Future research includes implementing a proof of concept of the proposed scheme and evaluating it in a real-world application.

Conflict of interest

None.

References

[1] W.-T. Balke, U. Güntzer, J.X. Zheng, Efficient distributed skylining for web information systems, in: Advances in Database Technology - International Conference on Extending Database Technology, 2004, pp. 256–273.
[2] J. Benaloh, Dense probabilistic encryption, in: Proceedings of The Workshop on Selected Areas of Cryptography, 1994, pp. 120–128.
[3] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: International Conference on Data Engineering, 2001, pp. 421–430.
[4] M.R. Brown, Implementation and analysis of binomial queue algorithms, SIAM J. Comput. 7 (3) (1978) 298–319.
[5] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, Skyline with presorting, in: International Conference on Data Engineering, 2003, pp. 717–719.
[6] C. Crane, Linear lists and priority queues as balanced binary trees, Tech. Rep. CS-72-259, Dept. of Comp. Sci., Stanford University, 1972.
[7] E. Dellis, A. Vlachou, I. Vladimirskiy, B. Seeger, Y. Theodoridis, Constrained subspace skyline computation, in: International Conference on Information and Knowledge Management, 2006, pp. 415–424.
[8] S.E. Dilsizian, E.L. Siegel, Artificial intelligence in medicine and cardiac imaging: harnessing big data and advanced computing to provide personalized medical diagnosis and treatment, Curr. Cardiol. Rep. 16 (1) (2014) 441.
[9] C. Dobre, F. Xhafa, Intelligent services for big data science, Future Gener. Comput. Syst. 37 (2014) 267–281.
[10] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM. 34 (3) (1987) 596–615.
[11] G.H. Gonnet, J.I. Munro, Heaps on heaps, in: Automata, Languages and Programming, 1982, pp. 282–291.
[12] K. Hose, C. Lemke, K.-U. Sattler, Processing relaxed skylines in PDMS using distributed data summaries, in: International Conference on Information and Knowledge Management, 2006, pp. 425–434.
[13] J. Hua, H. Zhu, F. Wang, X. Liu, R. Lu, H. Li, Y. Zhang, Cinema: efficient and privacy-preserving online medical primary diagnosis with skyline query, IEEE Internet Things J. (2018).
[14] C. Huang, R. Lu, H. Zhu, J. Shao, X. Lin, FSSR: fine-grained ehrs sharing via similarity-based recommendation in cloud-assisted ehealthcare system, in: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, 2016, pp. 95–106.
[15] D. Kossmann, F. Ramsak, S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: International Conference on Very Large Data Bases, 2002, pp. 275–286.
[16] H.T. Kung, F. Luccio, F.P. Preparata, On finding the maxima of a set of vectors, J. ACM. 22 (4) (1975) 469–476.
[17] K.C.K. Lee, B. Zheng, H. Li, W.-C. Lee, Approaching the skyline in Z order, in: International Conference on Very Large Data Bases, 2007, pp. 279–290.
[18] X. Liu, R. Lu, J. Ma, L. Chen, H. Bao, Efficient and privacy-preserving skyline computation framework across domains, Future Gener. Comput. Syst. 62 (2016) 161–174.
[19] E. Lo, K.Y. Yip, K.-I. Lin, D.W. Cheung, Progressive skylining over web-accessible databases, Data Knowl. Eng. 57 (2) (2006) 122–147.
[20] Z. Lv, H. Song, P. Basanta-Val, A. Steed, M. Jo, Next-generation big data analytics: state of the art, challenges, and future research topics, IEEE Trans. Ind. Inf. 13 (4) (2017) 1891–1899.
[21] D. Papadias, Y. Tao, G. Fu, B. Seeger, Progressive skyline computation in database systems, ACM Trans. Database Syst. 30 (1) (2005) 41–82.
[22] J. Pei, W. Jin, M. Ester, Y. Tao, Catching the best views of skyline: a semantic approach based on decisive subspaces, in: International Conference on Very Large Data Bases, 2005, pp. 253–264.
[23] J.-R. Sack, T. Strothotte, An algorithm for merging heaps, Acta Inf. 22 (2) (1985) 171–186.
[24] K.-L. Tan, P.-K. Eng, B.C. Ooi, Efficient progressive skyline computation, in: International Conference on Very Large Data Bases, 2001, pp. 301–310.
[25] Y. Tao, X. Xiao, J. Pei, SUBSKY: efficient computation of skylines in subspaces, in: International Conference on Data Engineering, 2006, p. 65.
[26] C.-F. Tsai, W.-C. Lin, S.-W. Ke, Big data mining with parallel computing: a comparison of distributed and mapreduce methodologies, J. Syst. Softw. 122 (2016) 83–92.
[27] A. Vlachou, C. Doulkeridis, Y. Kotidis, M. Vazirgiannis, SKYPEER: efficient subspace skyline computation over distributed data, in: International Conference on Data Engineering, 2007, pp. 416–425.
[28] S. Wang, B.C. Ooi, A.K.H. Tung, L. Xu, Efficient skyline query processing on peer-to-peer networks, in: International Conference on Data Engineering, 2007, pp. 1126–1135.
[29] Y. Yuan, X. Lin, Q. Liu, W. Wang, J.X. Yu, Q. Zhang, Efficient computation of the skyline cube, in: International Conference on Very Large Data Bases, 2005, pp. 241–252.
[30] H. Zhang, Q. Zhang, Communication-efficient distributed skyline computation, in: Proceedings of the ACM on Conference on Information and Knowledge Management, 2017, pp. 437–446.
[31] Y. Zhang, M. Chen, D. Huang, D. Wu, Y. Li, Idoctor: personalized and professionalized medical recommendations based on hybrid matrix factorization, Future Gener. Comput. Syst. 66 (2017) 30–35.
[32] X. Zhao, Y. Wu, W. Cui, X. Du, Y. Chen, Y. Wang, D.L. Lee, H. Qu, Skylens: visual analysis of skyline on multi-dimensional data, IEEE Trans. Vis. Comput. Graph. 24 (1) (2018) 246–255.