J Med Syst (2019) 43: 123
https://doi.org/10.1007/s10916-019-1229-1

MOBILE & WIRELESS HEALTH

Achieving Efficient and Privacy-Preserving *k*-NN Query for Outsourced eHealthcare Data

Yandong Zheng[1] · **Rongxing Lu**[1] · **Jun Shao**[2]

Received: 29 October 2018 / Accepted: 27 February 2019 / Published online: 27 March 2019

Abstract

The boom of Internet of Things devices promotes huge volumes of eHealthcare data will be collected and aggregated at eHealthcare provider. With the help of these health data, eHealthcare provider can offer reliable data service (e.g., k-NN query) to doctors for better diagnosis. However, the IT facility in the eHealthcare provider is incompetent with the huge volumes of eHealthcare data, so one popular solution is to deploy a powerful cloud and appoint the cloud to execute the k-NN query service. In this case, since the eHealthcare data are very sensitive yet cloud servers are not fully trusted, directly executing the k-NN query service in the cloud inevitably incurs privacy challenges. Apart from the privacy issues, efficiency issues also need to be taken into consideration because achieving privacy requirement will incur additional computational cost. However, existing focuses on k-NN query do not (fully) consider the data privacy or are inefficient. For instance, the best computational complexity of k-NN query over encrypted eHealthcare data in the cloud is as large as $O(k \log^3 N)$, where N is the total number of data. In this paper, aiming at addressing the privacy and efficiency challenges, we design an efficient and privacy-preserving k-NN query scheme for encrypted outsourced eHealthcare data. Our proposed scheme is characterized by integrating the kd-tree with the homomorphic encryption technique for efficient storing encrypted data in the cloud and processing privacy-preserving k-NN query over encrypted data. Compared with existing works, our proposed scheme is more efficient in terms of privacy-preserving k-NN query. Specifically, our proposed scheme can achieve k-NN computation over encrypted data with $O(lk \log N)$ computational complexity, where l and N respectively denote the data dimension and the total number of data. In addition, detailed security analysis shows that our proposed scheme is really privacy-preserving under our security model and performance evaluation also indicates that our proposed scheme is indeed efficient in terms of computational cost.

Keywords k-NN query · kd-tree · eHealthcare data · Privacy preservation

Introduction

With the boom of Internet of Things (IoT) devices, especially the smart wearable/implantable body sensor devices, huge volumes of eHealthcare data will be collected and aggregated at the eHealthcare provider [5, 14]. With the help of these nearly real-time health data, the eHealthcare provider can offer reliable data service to doctors for better diagnosis, such as query service, recommendation service, etc. [12, 13, 22]. In particular, the k-NN query service is one of the most important applications in eHealthcare data. For example, doctors can use the service to find the top k patients who have the similar symptoms with a given patient A, then the treatment history of these k patients can provide some useful suggestions for the treatment of the current patient A. However, as the volume, velocity, and variety of the eHealthcare data increase significantly, the IT facility in the eHealthcare provider is gradually incompetent for storing and processing the huge volumes of eHealthcare data. One popular solution is to outsource the eHealthcare data to the cloud that is usually considered to be powerful in both storage capacity and computing capability. For example, Philips is building its healthcare digital platform over Amazon Web Services, which can

This article is part of the Topical Collection on *Mobile & Wireless Health*

✉ Rongxing Lu
rlu1@unb.ca

[1] Faculty of Computer Science, University of New Brunswick, Fredericton, New Brunswick, E3B5A3, Canada

[2] School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, Zhejiang, 310018, China

store and analyze 15PB patient data collected from studies, medical records and patient inputs [1]. However, since the eHealthcare data are very sensitive while the cloud servers are not fully trusted [20], directly processing the k-NN query over *plaintexts* eHealthcare data in the cloud will bring many security and privacy issues. Although the eHealthcare data privacy can be easily protected by using cryptographic technology, this method brings many efficiency challenges for processing the encrypted data. For instance, the best computational complexity of k-NN query over encrypted eHealthcare data in the cloud is as large as $O(k \log^3 N)$, where N is the total number of data [19] and the computational complexity is based on the number of operations over encrypted data.

Aiming at the privacy and efficiency challenges, in this work, we propose a new efficient and privacy-preserving k-NN query scheme for outsourced eHealthcare data by integrating the kd-tree data structure with homomorphic encryption technique. Specifically, compared with our previous conference version [23], the main contributions of this work are three-fold:

- First, we design a privacy-preserving kd-tree data structure to efficiently store, insert and delete encrypted eHealthcare data in the cloud.
- Second, we achieve efficient and privacy-preserving k-NN query by introducing two privacy-preserving protocols respectively for the encrypted data comparison and Euclidean distance computation in the cloud. The computational complexity of our proposal is only $O(lk \log N)$, where l is the data dimension and N is the total number of eHealthcare data. Generally, the value of l is smaller than 10, while N is bigger than 1,000,000. Thus, the computational cost of our proposal is less than that of the best k-NN query scheme, i.e., $O(k \log^3 N)$.
- Third, we conduct extensive simulations to evaluate the efficiency of our proposed scheme, and the results show that it is indeed efficient in terms of computational cost.

The remainder of this paper is organized as follows. We first present our system model, security model and design goals in "Models and design goals". Some preliminaries are introduced in "Preliminaries". In "Our proposed scheme", our scheme is proposed. Security analysis and performance evaluation are introduced in "Security analysis" and "Performance evaluation" respectively. Related work is discussed in "Related work". Finally, we conclude this paper in "Conclusion".

Models and design goals

In this section, we present our system model, security model, and identify our design goals in detail.

System model

In our system model, we consider a typical cloud-based k-NN query service, which consists three kinds of entities, i.e., a data owner (eHealthcare provider), a cloud with two cloud servers $\mathcal{CS} = \{S_1, S_2\}$ and multiple users (doctors) $\mathcal{U} = \{U_1, U_2, \cdots\}$, as shown in Fig. 1.

- Data owner: We consider an eHealthcare provider as the data owner in our model, which has collected high volumes of eHealthcare data. In order to make use of these data, the eHealthcare provider intends to offer k-NN query service to users. However, since he/she does not have abundant storage and computational resources, the eHealthcare provider deploys a powerful cloud to store data and provide k-NN query service to users. As eHealthcare data are usually sensitive, the eHealthcare provider tends to encrypt these data before outsourcing them to the cloud. In addition, when eHealthcare data gradually increase or decrease as time goes, the eHealthcare provider is also responsible for updating data in the cloud, such as inserting new data or removing obsolete data.
- Cloud $\mathcal{CS} = \{S_1, S_2\}$: In our model, the cloud is deployed as a link to connect data owner and multiple users. That is, he/she not only stores the outsourced data from data owner, but also offers reliable and efficient k-NN query

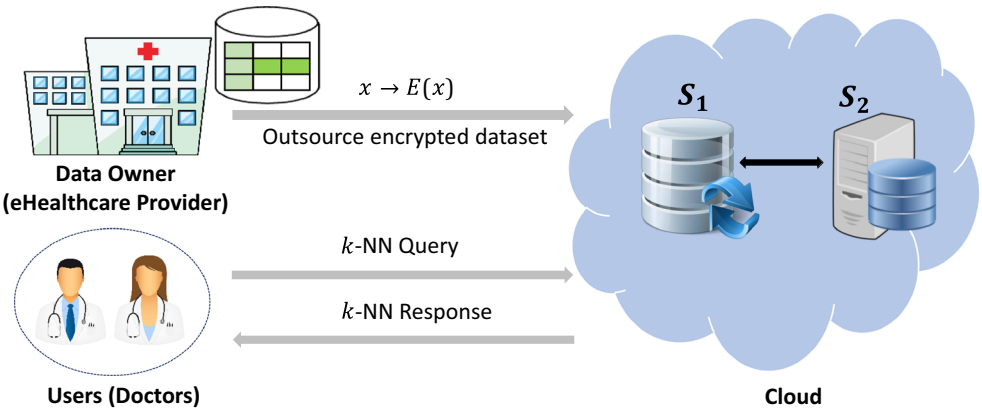


Fig. 1 System model under consideration

service to the users. In particular, we deploy two servers S_1 and S_2 in the cloud, both of them possess abundant storage space and powerful computing capability, and will cooperate together to provide k-NN query service to users.

- Users $\mathcal{U} = \{U_1, U_2, \cdots\}$: We consider doctors as the users in our model and each user $U_i \in \mathcal{U}$ must be authorized by the data owner before he/she enjoys the k-NN query service. In other words, only authorized users can receive and recover the k-NN query results from the cloud, while unauthorized users cannot. Note that, in our model, Euclidean distance is used to measure the distance between two data records.

Security model

In our security model, we consider the data owner is *honest*, i.e., he/she will honestly encrypt his/her data and outsource them to the cloud. However, both cloud servers, S_1 and S_2, are considered to be *honest-but-curious*, i.e., they will faithfully store the outsourced data from data owner and offer reliable k-NN query service to users, but may be curious about some private information including the data stored in the cloud and the query data records given by authorized users. At the same time, S_1 and S_2 are not allowed to collude with each other. This assumption is reasonable since the cloud servers are usually companies with high reputation. For the users, authorized users are considered to be *honest*, while unauthorized users may launch some malicious attacks in order to use the k-NN query service from the cloud without authorization. Note that, there may be other active attacks or passive attacks occurring in the cloud, such as the data pollution attack and deniable of service attack, which will be considered in our future work.

Design goals

Our design goal is to present an efficient and privacy-preserving k-NN query scheme for eHealthcare data. In specific, the following objectives should be satisfied.

- *Privacy preservation:* The basic security requirement of our proposed scheme is privacy preservation. That is, the data, including those stored in the cloud and the query data records given by authorized users, should be kept secret from unauthorized entities, including both cloud servers S_1 and S_2, and unauthorized users.
- *Computation efficiency:* In order to achieve the above privacy requirement, additional computational cost will be incurred. Specifically, processing k-NN query over encrypted data in the cloud will consume more computational resources compared with doing that in plaintext data. Therefore, in our proposed scheme, we aim to minimize the computational cost of the k-NN query in the cloud.

Preliminaries

In this section, we briefly review the Paillier public key encryption (PKE) scheme and the kd-tree technique, which will serve as the building blocks in our proposed scheme.

The paillier PKE

The Paillier PKE (described in [15]) is a popular homomorphic encryption technique and has been widely used in privacy-preserving computation in recent years. It consists of three algorithms, i.e., key generation $KeyGen(\kappa)$, encryption $Enc(pk, m)$ and decryption $Dec(sk, c)$.

- $KeyGen(\kappa)$: Given a security parameter $\kappa \in Z^+$, large prime numbers p, q, p' and q' are randomly selected such that $p = 2p' + 1, q = 2q' + 1$ and $|p| = |q| = \kappa$. Let $n = pq$ and $\lambda = lcm(p-1, q-1) = 2p'q'$. In addition, define a function $L(x) = \frac{x-1}{n}$ and randomly choose a generator $g \in \mathbb{Z}^*_{n^2}$. Then, let $u = L(g^\lambda \bmod n^2)^{-1} \bmod n$ and $KeyGen(\kappa)$ outputs the public key $pk = (n, g)$ and private key $sk = (\lambda, \mu)$.
- $Enc(pk, m)$: Given the public key pk and a message $m \in \mathbb{Z}_n$, the message m can be encrypted as $c = E(m) = g^m \cdot r^n \bmod n^2$, where $r \in \mathbb{Z}^*_n$ is a random number.
- $Dec(sk, c)$: Given the private key sk and a ciphertext $c = E(m)$, the message m can be recovered as $m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$.

The Paillier PKE satisfies the following two homomorphic properties.

- *Homomorphic Addition:* Given two ciphertexts $E(m_1)$ and $E(m_2)$, we have $E(m_1) \cdot E(m_2) = E(m_1 + m_2)$.
- *Homomorphic Multiplication:* Given a ciphertext $E(m_1)$ and a plaintext $m_2 \in \mathbb{Z}_n$, we have $E(m_1)^{m_2} = E(m_1 \cdot m_2)$.

The *k*d-tree technique

The kd-tree (described in [6]) is a kind of data structure, which can efficiently deal with the nearest neighbor search problem in multi-dimensional Euclidean space. In the following, we first give the formal definition of the kd-tree, and then introduce several algorithms related to the kd-tree.

Definition of kd-tree [6]: The kd-tree is a binary tree in which each tree node is a k-dimensional data record. In addition, each tree node x contains four attributes $cd, data, left$ and $right$, which denote the cutting

dimension, the key value, the left child and the right child, respectively. In order to facilitate the narrative, we use $x.cd$, $x.data$, $x.left$ and $x.right$ to denote the attributes of x, respectively. The kd-tree satisfies *order relation*. In specific, for each tree node x, the key value of x in the current cutting dimension $x.cd$ is larger than or equal to the key value of its left child in $x.cd$, and less than the key value of its right child in $x.cd$. That is, $x.left.data[x.cd] \leq x.data[x.cd] < x.right.data[x.cd]$. Next, we recall several algorithms related to the kd-tree, i.e., tree building, insertion, deletion, and k-NN query.

Algorithm 1 Tree building and insertion algorithm.

```
1: function BuildTree(Dataset D, Tree T)
2:     if D == null then
3:         return null
4:     end if
5:     compute the variance for data values in each dimension
6:     cd = the dimension with the largest variance
7:     s = the data record with median value of {x[cd]|x ∈ D}
8:     for x in S do
9:         if x[cd] ≤ s[cd] then
10:            put x into D_l
11:        else
12:            put x into D_r
13:        end if
14:    end for
15:    T.cd = cd; T.data = s
16:    T.left = BuildTree(D_l, T.left)
17:    T.right = BuildTree(D_r, T.right)
18:    return T
19: end function
20: function Insert(Record x, Tree T)
21:    if T == null then
22:        T = BuildTree(x, T)
23:    else if x == T.data then
24:        // error! duplicate
25:    else if x[T.cd] ≤ T.data[T.cd] then
26:        T.left = Insert(x, T.left)
27:    else
28:        T.right = Insert(x, T.right)
29:    end if
30:    return T
31: end function
```

- Tree Building: Given a k-dimensional dataset D, a kd-tree T can be built as the function $BuildTree(\cdot)$ in Algorithm 1.
  - First, compute data variance of all data records in each dimension according to the dataset S and choose the dimension with the largest variance as the cutting dimension cd. Note that choosing the dimension with the largest variance as the cutting dimension can make the built kd-tree more balanced. If the initial dataset is empty and later data records are inserted one by one, the built kd-tree may be unbalanced and the k-NN query efficiency will decrease. Thus, we recommend that data owner should outsource large numbers of data records in the initialization phase and small numbers of data records are inserted one by one.
  - Second, compute the median value among all data values in the cutting dimension cd, i.e., $\{x[cd]|x \in D\}$, and choose the data record with the median value as the splitting data record, denoted by s.
  - Third, split D into two subsets D_l and D_r according to the splitting data record s. For each $x \in D$, if $x[cd] \leq s[cd]$, it will be put into D_l. Otherwise, it will be put into D_r.
  - Finally, create the tree root T with cd and s as the cutting dimension and splitting data record, respectively. After that, the left child and right child of the root node can be recursively built from D_l and D_r.

Algorithm 2 Minimum finding and deletion algorithm.

```
1: function FindMin(Tree T, int i)
2:     if T == null then
3:         return null
4:     end if
5:     if T.cd == i then
6:         if T.left == null then
7:             return T.data
8:         else
9:             return FindMin(T.left, i)
10:        end if
11:    else
12:        return min(FindMin(T.left, i)[i], FindMin(T.right, i)[i], T.data[i])
13:    end if
14: end function
15:
16: function Delete(Record x, Tree T)
17:    if x == T.data then
18:        if T.right != null then
19:            T.data = FindMin(T.right, T.cd)
20:            T.right = Delete(T.data, T.right)
21:        else if T.left != null then
22:            T.data = FindMin(T.left, T.cd)
23:            T.right = Delete(T.data, T.left)
24:        else
25:            T = null
26:        end if
27:    else if x[T.cd] ≤ T.data[T.cd] then
28:        T.left = Delete(x, T.left)
29:    else
30:        T.right = Delete(x, T.right)
31:    end if
32:    return T
33: end function
```

- Insertion: Give a k-dimensional data record x and a kd-tree T, x can be inserted into T as the function $Insert(\cdot)$ in Algorithm 1.
- Minimum Finding: Given a dimension i and a kd-tree T, the function $FindMin(\cdot)$ in Algorithm 2 can be used to find the data record with the smallest value in the i-th

dimension. It is a recursive function, which starts from the root node T and checks the cutting dimension of T (i.e., $T.cd$). On the one hand, if $T.cd == i$, continue to recurse on the left subtree because the smallest value cannot be in the right subtree. In this case, if $T.left == null$, the current data record (i.e., $T.data$) is exactly the data record with the smallest value in the i-th dimension. One the other hand, if $T.cd \neq i$, the smallest value could be in either left subtree or right subtree, so continue to recurse on both subtrees.

- Deletion: The function *Delete*$(\cdot)$, as shown in Algorithm 2, is used to delete a data record x from a kd-tree T. It is a recursive function, which starts from the root node T and checks the key value of T (i.e., $T.data$). If $T.data == x$, delete this data record and return. Otherwise, x could be in either left subtree or right subtree, so continue to run the recursive algorithm on both subtrees to find x and delete it.

Algorithm 3 k-NN(Record x, Tree T, Queue PQ).

```
1: if T == null then
2:     return PQ
3: end if
4: if size(PQ) < k or dist(x, T.data) <dist(PQ_1, x) then
5:     PQ.enqueue(T.data)
6:     Adjust the order of the queue
7:     bestdist = dist(PQ_1, x)
8: end if
9: //Best Bin First Search strategy
10: if x[T.cd] ≤ T.data[T.cd] then
11:     k-NN(x, T.left, PQ)
12: else
13:     k-NN(x, T.right, PQ)
14: end if
15: //Prune strategy
16: if (x[cd] − T.data[cd])^2 < bestdist then
17:     if x[cd] ≤ T.data[cd] then
18:         k-NN(x, T.right, PQ)
19:     else
20:         k-NN(x, T.left, PQ)
21:     end if
22: end if
```

- k-NN Query: The k-NN query is a popular searching algorithm to find the k nearest neighbours with a given data record x. A straight solution to achieve the kNN query over a kd-tree is to traverse the whole tree, but it is inefficient obviously. In order to improve the efficiency of k-NN query, in this work, we adopt the *Best Bin First Search* strategy (described in [3] in detail), which will give a higher searching priority for those subtrees that are more likely to contain the k nearest data records. For instance, for a tree node T, if $x[T.cd] \leq T.data[T.cd]$, the left subtree of T (i.e., $T.left$) is more likely to contain the k nearest data records of x, compared with the right subtree T (i.e., $T.right$). In this case, $T.left$ will have a higher searching priority. In addition, in order to further improve the efficiency of kNN query, we use a priority queue PQ with size k to store the current top k closest data records with the query data record x. At the same time, the data records in the queue are stored in the descending order of distances with the query data record x, i.e., PQ_1 has the largest distance with x. With the priority queue, a subtree T can be pruned when it satisfies $(x[cd] - T.data[cd])^2 > bestdist$, i.e., it cannot contain any data records that are closer than the data records in the queue. Based on the *Best Bin First Search* strategy and the priority queue, the k-NN query algorithm is presented in Algorithm 3.

Our proposed scheme

In this section, we present our proposed k-NN query scheme. Before delving into the details, we first introduce a monotonically increasing and one-way function, which will serve as the building block of our proposed scheme for privacy preservation.

The monotonically increasing and one-way function

Suppose that $D = \{x = (x_1, x_2, \cdots, x_l) | x_i \in Z^+ \text{ and } x_i \leq U, i = 1, 2, \cdots, l\}$ is an l-dimensional dataset, where U is the upper bound of all data values in D. At the same time, let $DS = \{dist^2(x, y) = \sum_{i=1}^{l}(x_i - y_i)^2 | x, y \in D\}$ denote a set of Euclidean distances, which contains the distance of any two data records in D. Then, we can construct a function f, which maps each element $d^2 \in DS$ to $f(d^2)$. In specific, for each $d^2 \in DS$, $f(d^2)$ is

$$f(d^2) = a_1(d^2 \bmod \Delta) + a_2(d^2 \bmod \Delta)^2 + \cdots + a_n(d^2 \bmod \Delta)^n + e$$

where $\Delta = l \cdot U^2$, each coefficient a_i is an integer and $a_i > \Delta^i$ for $i = 1, 2, \cdots, n$. In addition, e is a noise and randomly chosen from $(\Delta, a_1 + a_2 + \cdots + a_n)$.

Next, we will prove that the function f is not only a monotonically increasing function, but also a one-way function.

Theorem 1 (Monotonically Increasing Function) *The function f is a monotonically increasing function. That is, for any $d_1^2, d_2^2 \in DS$, if $d_1^2 > d_2^2$, $f(d_1^2) > f(d_2^2)$.*

Proof For any $d_1^2, d_2^2 \in DS$, suppose that $d_1^2 > d_2^2$ and

$$f(d_1^2) = a_1(d_1^2 \bmod \Delta) + \cdots + a_n(d_1^2 \bmod \Delta)^n + e_1 \quad (1)$$

$$f(d_2^2) = a_1(d_2^2 \bmod \Delta) + \cdots + a_n(d_2^2 \bmod \Delta)^n + e_2 \quad (2)$$

where $\Delta = l \cdot U^2$, and e_1, e_2 are randomly chosen from $(\Delta, a_1 + \cdots + a_n)$. For any $x, y \in D$, $dist^2(x, y) = \sum_{i=1}^{l}(x_i - y_i)^2$. Since $x_i \in Z^+$, $y_i \in Z^+$, and they satisfy

$x_i \le U$ and $y_i \le U$ for $i = 1, 2, \cdots, l$, we can obtain that $dist^2(x, y) \in Z^+$ and

$$1 \le dist^2(x, y) \le l \cdot U^2 \triangleq \Delta \quad (3)$$

Since DS is a set of Euclidean distances, which contains the distance of any two data records in D, thus any $d^2 \in DS$ satisfies that $d^2 \in Z^+$ and $1 \le d^2 \le \Delta$. Thus, $1 \le d_1^2 \le \Delta$, $1 \le d_2^2 \le \Delta$ and we have

$$d_1^2 \bmod \Delta = d_1^2,\ d_2^2 \bmod \Delta = d_2^2 \quad (4)$$

In this case, the result of Eqs. (1) – (2) is

$$f(d_1^2) - f(d_2^2) = a_1(d_1^2 - d_2^2) + \cdots + a_n((d_1^2)^n - (d_2^2)^n) + (e_1 - e_2) \quad (5)$$

In addition, we have $d_1^2 \in Z^+$, $d_2^2 \in Z^+$ and $d_1^2 > d_2^2$, so $d_1^2 - d_2^2 \ge 1$. Hence, for any $1 \le i \le n$, $(d_1^2)^i - (d_2^2)^i \ge 1$ always holds. Then, we have

$$f(d_1^2) - f(d_2^2) \ge a_1 + a_2 + \cdots + a_n + (e_1 - e_2) \quad (6)$$

Since $e_1, e_2 \in (\Delta, a_1 + \cdots + a_n)$, it is easy to obtain that

$$\Delta - (a_1 + \cdots + a_n) < e_1 - e_2 < (a_1 + \cdots + a_n) - \Delta \quad (7)$$

and

$$f(d_1^2) - f(d_2^2) \ge a_1 + a_2 + \cdots + a_n + (e_1 - e_2) > 0 \quad (8)$$

Therefore, for any $d_1^2, d_2^2 \in DS$, if $d_1^2 > d_2^2$, $f(d_1^2) > f(d_2^2)$ holds. In other words, f is a monotonically increasing function. □

Theorem 2 (One-way Function) *The function f is a one-way function, i.e., it is infeasible to recover d^2 from $f(d^2)$ for any $d^2 \in DS$.*

Proof For any $d^2 \in D$, we have

$$f(d^2) = a_1(d^2 \bmod \Delta) + a_2(d^2 \bmod \Delta)^2 + \cdots + a_n(d^2 \bmod \Delta)^n + e \quad (9)$$

where e is a noise and randomly chosen from $(\Delta, a_1 + \cdots + a_n)$. If an attacker attempts to recover d^2 from $f(d^2)$, he/she must first recover the coefficients $\{a_1, a_2, \cdots, a_n\}$. Two attack methods can be considered to recover $\{a_1, a_2, \cdots, a_n\}$, i.e., Lagrange interpolation attack and modular operation attack. However, the function f can resist both of them, i.e., modular operation attack or Lagrange interpolation attack are invalid to recover the coefficient $\{a_1, a_2, \cdots, a_n\}$ as discussed below.

- *The function f can resist modular operation attack.* An attacker may try to use modular operation to recover a_i from a data pair $(d^2, f(d^2))$, where $f(d^2) = a_1(d^2 \bmod \Delta) + \cdots + a_n(d^2 \bmod \Delta)^n + e$. On the one hand, the attacker can obtain $e \bmod d^2$ by computing $f(d^2) \bmod d^2 \equiv e \bmod d^2$. However, e is randomly chosen from $(\Delta, a_1 + \cdots + a_n)$ and $e > \Delta > d^2$, so $e > d^2$ and attacker cannot deduce e from $e \bmod d^2$. On the other hand, if $(d^2 \bmod \Delta)^n$ is big enough to satisfy: (i) $(d^2 \bmod \Delta)^n > a_n$; (ii) $(d^2 \bmod \Delta)^n > \sum_{i=1}^{n-1} a_i(d^2 \bmod \Delta)^i + e$, a_n can be recovered by

$$\begin{aligned} \frac{f(d^2)}{(d^2 \bmod \Delta)^n} &= \frac{a_1(d^2 \bmod \Delta) + \cdots + a_n(d^2 \bmod \Delta)^n + e}{(d^2 \bmod \Delta)^n} \\ &= \frac{\sum_{i=1}^{n-1} a_i(d^2 \bmod \Delta)^i + e}{(d^2 \bmod \Delta)^n} + a_n \\ &= a_n \end{aligned} \quad (10)$$

However, as described in the definition of f, $a^n > \Delta^n$. At the same time, it is obvious that $(d^2 \bmod \Delta)^n < \Delta^n$, so $(d^2 \bmod \Delta)^n < a_n$ and $(d^2 \bmod \Delta)^n$ cannot satisfy conditions (i), let alone condition (ii). Thus, the attacker cannot recover these coefficients by the modular operation attack.
- *The function f can resist Lagrange interpolation attack.* According to the Lagrange interpolation formula, recovering n unknown coefficients $\{a_1, a_2, \cdots, a_n\}$ requires at least n data pairs like $(d^2, f(d^2))$, where $d^2 \in DS$. Without loss of generality, assume that the attacker has collected such kind of dataset, denoted by $DA = \{(d_i^2, f(d_i^2) = \sum_{j=1}^{n} a_j(d_i^2 \bmod \Delta)^j + e_i) | d_i^2 \in DS, 1 \le i \le n\}$. Then, the attacker will try to recover the coefficients of f using the Lagrange interpolation formula. However, during the process of running formula, he/she will find that he/she has no idea on these randomly chosen noises $\{e_1, e_2, \cdots, e_n\}$. In addition, based on the above discussion, it is difficult for him/her to obtain them. Without these noises $\{e_1, e_2, \cdots, e_n\}$, the Lagrange interpolation formula does not work, so the attacker cannot recover these coefficients by the Lagrange interpolation attack.

Therefore, both modular operation attack and Lagrange interpolation attack are invalid to recover the coefficient $\{a_1, a_2, \cdots, a_n\}$, let alone recovering d^2 from $f(d^2)$. In addition, even if the attacker has obtained the coefficients $\{a_1, a_2, \cdots, a_n\}$, it is still difficult for him/her to solve the equation $f(d^2) = a_1(d^2 \bmod \Delta) + \cdots + a_n(d^2 \bmod \Delta)^n + e$ without knowing the random noise e. Thus, an attacker cannot recover d^2 from $f(d^2)$ from any $d^2 \in DS$, so the function f is a one-way function.

Note that choosing the degree of function f, i.e., n, we need to consider both security and efficiency issues. As n increases, the computational efficiency of function f will decrease. Thus, an optimal n can be chosen according to the security and efficiency requirements in the specific scenarios. □

The description of our proposed scheme

Now, we present our proposed scheme, which is comprised of four phases: system initialization, outsourcing encrypted data to the cloud, maintaining encrypted data in the cloud, and k-NN query over encrypted data.

System initialization

In our scheme, we consider the trusted data owner (eHealthcare provider) bootstraps the whole system. In specific, given the security parameter κ, α, β, where $\alpha + \beta << \kappa$, the data owner can generate the public key $pk = (n, g)$ and private key $sk = (\lambda, \mu)$ according to the $keyGen(\kappa)$ algorithm in "The paillier PKE". In addition, data owner randomly selects ak as the access key and chooses AES algorithm as the basic encryption algorithm. Then, it will publish the public key pk, and distribute the private key sk to cloud server S_2. At the same time, it also sends the access key ak to each authorized user. Note that the Paillier PKE can only encrypt and decrypt integers, thus data owner will convert all data into integers in the same way before encrypting them. Without loss of generality, we assume that all of the plaintext data in our scheme belong to $\{0, 1\}^{\alpha}$, including the Euclidean distance.

Outsourcing encrypted data to the cloud

Assume that data owner has an l-dimensional eHealthcare dataset $D = \{(x_1, x_2, \cdots, x_l) | x_i \in \{0, 1\}^{\alpha}, 1 \leq i \leq l\}$. After system initialization, he/she will encrypt his/her data and outsource them to the cloud as follows.

- ***Step-1:*** Build an ld-tree for the dataset D using the function $BuildTree(\cdot)$ in Algorithm 1.
- ***Step-2:*** Encrypt the built ld-tree, i.e., encrypting the key value of each tree node. Specifically, for the key value of each tree node $x = (x_1, x_2, \cdots, x_l)$, the data owner first uses the Paillier PKE to encrypt $(x_1, x_2, \cdots, x_l)$ as $(E(x_1), E(x_2), \cdots, E(x_l))$, and then uses the AES algorithm to encrypt the whole record x as $\text{AES}_{ak}(x)$. Then, the key value becomes $E(x) = (E(x_1), E(x_2), \cdots, E(x_l), \text{AES}_{ak}(x))$.
- ***Step-3:*** Outsource the encrypted ld-tree to cloud server S_1.

Maintaining encrypted data in the cloud

After the encrypted ld-tree is outsourced to the cloud, data owner maintains the ld-tree in the cloud by either inserting a new record or deleting an obsolete one.

Insertion Data owner can insert a new data record $x = (x_1, x_2, \cdots, x_l)$ into the encrypted ld-tree with the help of two cloud servers S_1 and S_2. First, he/she sends the encrypted record $E(x) = (E(x_1), E(x_2), \cdots, E(x_l), \text{AES}_{ak}(x))$ to S_1. Then, S_1 inserts $E(x)$ to the encrypted ld-tree by running the function $Insert(\cdot)$ in Algorithm 1. Since S_1 cannot access the plaintext ld-tree, he/she will face two challenges when running the algorithm: i) how to check $x == t.data$; and ii) how to compare $x[T.cd] < T.data[T.cd]$. Luckily, since the AES algorithm is a deterministic encryption, S_1 can check $\text{AES}_{ak}(x) == \text{AES}_{ak}(T.data)$ for solving the first challenge. For solving the second challenge, S_1 and S_2 can run the privacy-preserving data comparison protocol, as shown in Fig. 2.

Deletion Data owner can delete an obselete data record $x = (x_1, x_2, \cdots, x_l)$ from the encrypted ld-tree with the help of two cloud servers S_1 and S_2. First, he/she sends the encrypted data record $E(x) = (E(x_1), E(x_2), \cdots, E(x_l), \text{AES}_{ak}(x))$ to S_1. Then, S_1 deletes $E(x)$ from the encrypted ld-tree according to the deletion algorithm described in the Algorithm 2. During the deletion process, S_1 faces the same

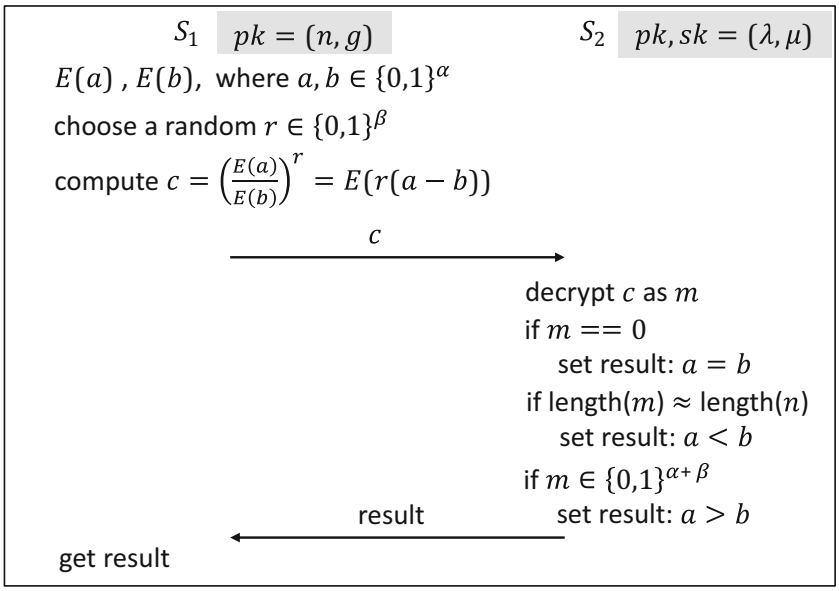


Fig. 2 Privacy-preserving data comparison protocol run between S_1 and S_2

challenges as that in the **Insertion** process, and S_1 can use the same strategies to deal with the two challenges.

k-NN query over encrypted data

A user $U_i \in \mathcal{U}$ can query the k nearest data records with a given data record $y = (y_1, y_2, \cdots, y_l)$ as the following steps.

- ***Step-1:*** U_i first encrypts the query data record y as $E(y) = (E(y_1), E(y_2), \cdots, E(y_l))$ using Paillier PKE algorithm.
- ***Step-2:*** U_i sends the k-NN query request together with encrypted query data record $E(y)$ to the cloud server S_1.
- ***Step-3:*** On receiving the query request from U_i, S_1 first checks whether he/she is authorized. If not, drop the current query request and finish. Otherwise, S_1 will continue with the following steps.
- ***Step-4:*** S_1 cooperates with S_2 to search the k nearest data records on the encrypted ld-tree according to the k-NN query algorithm (see Algorithm 3). Similarly, any two encrypted data comparison can be processed by running the privacy-preserving data comparison protocol, as shown in Fig. 2. However, apart from encrypted data comparison, S_1 will face another challenge: how to compute Euclidean distance for two encrypted data records. Aiming at this challenge, we design a privacy-preserving Euclidean distance computation protocol by integrating the permutation technique with a monotonically increasing and one-way function f (described in "The monotonically increasing and one-way function"), as shown in Fig. 3. With these two protocols, S_1 is able to obtain the k nearest data records of the query data record $E(y)$ with the help of S_2. At the same time, as described in Algorithm 3, the query results (i.e., k nearest data records) are stored in the priority queue $PQ = \{PQ[i] = E(x) = (E(x_1), E(x_2), \cdots, E(x_l), AES_{ak}(x)) | 1 \le i \le k\}$.
- ***Step-5:*** S_1 returns the query results $\{AES_{ak}(x) | E(x) \in PQ\}$ to U_i.
- ***Step-6:*** On receiving the query results, the authorized user U_i uses the access key ak to recover the k nearest data records.

Security analysis

In this section, we analyze the security of our proposed scheme. We particularly focus on the privacy property, i.e., the data including those stored in the cloud and the query data records given by authorized users should be privacy-preserving. In other words, these data will not be leaked to unauthorized entities including S_1, S_2 and unauthorized users, when they are stored in the cloud and processed by the cloud servers. Since the encryption technique, the encrypted data comparison protocol and the Euclidean distance computation protocol are the three significant techniques to support the privacy property (i.e., data privacy) of our scheme, the privacy preservation of our scheme can be achieved once all of them are privacy-preserving. Next, we analyze the security of these three techniques.

- *The Encryption Technique is Privacy-Preserving:* In our scheme, all data records outsourced by data owner or authorized users is encrypted by the Paillier PKE technique or AES algorithm. The security of these two algorithms guarantees that any unauthorized entity including S_1, S_2 and unauthorized users cannot obtain the data content from encrypted data. Thus, the encryption technique is privacy-preserving.

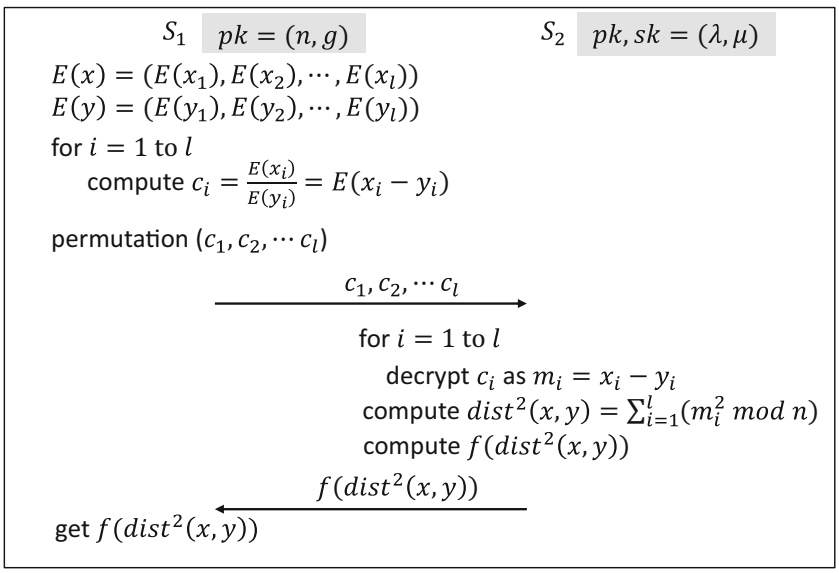


Fig. 3 Privacy-preserving Euclidean distance computation protocol run between S_1 and S_2

- *The Encrypted Data Comparison Protocol is Privacy-Preserving:* The data comparison protocol run between S_1 and S_2, as shown in Fig. 2, is a basic component for the insertion, deletion, and k-NN query algorithm in our proposed scheme. In the protocol, S_2 returns one comparison result of two encrypted data $E(a)$ and $E(b)$ to S_1 each time. Thus, S_1 cannot obtain the data a and b from the data comparison protocol. At the same time, S_1 uses the homomorphic property to blind the data, i.e., $(\frac{E(a)}{E(b)})^r \xrightarrow{encrypt} E(r(a-b))$ and sends it to S_2. Although S_2 can recover $r(a-b)$ from $E(r(a-b))$ with his/her private key sk and obtain the comparison result from the length of $r(a-b)$, he/she also has no idea on a and b due to the blind factor, i.e., random number r. Thus, both S_1 and S_2 cannot obtain the data content a or b during the data comparison protocol. Similarly, unauthorized users also cannot. Therefore, the data comparison protocol is privacy-preserving.
- *The Euclidean Distance Computation Protocol is Privacy-Preserving:* The Euclidean distance computation protocol is a critical component in the k-NN query algorithm and run between S_1 and S_2. As shown in Fig. 3, in the protocol, when computing the Euclidean distance between $E(x)$ and $E(y)$, S_1 first computes $(E(x_1-y_1), E(x_2-y_2), \cdots, E(x_l-y_l)) \triangleq (c_1, c_2, \cdots, c_l)$. Then, he/she applies data permutation strategy to $(c_1, c_2, \cdots, c_l)$ before sending it to S_2. If there is no permutation, S_2 can use his/her private key sk to recover $(x_1-y_1, x_2-y_2, \cdots, x_l-y_l)$, i.e, $x-y$, after receiving $(c_1, c_2, \cdots, c_l)$. At this time, if S_2 has a chance to get x (or y), the other one, i.e., y (or x) will be disclosed. Therefore, in our scheme, the permutation technique is applied to prevent the above attack. With the permutation technique, even if S_2 knows x and each permutated distance $m_i = x_i - y_i$ for $E(x)$ and $E(y)$, S_2 can correctly recover y only with probability $\frac{1}{l!}$. When $l = 10$, the probability is only $\frac{1}{3,628,800}$. Besides, in our scheme, each data record has been encrypted, the security of the encryption algorithm guarantees that it is difficult for S_2 to get any data record. Therefore, in the Euclidean distance computation protocol, S_2 cannot obtain data content x or y from encrypted data.

At the same time, S_2 returns $f(dist^2(x, y))$ instead of $dist^2(x, y)$ to S_1 when computing Euclidean distance between two encrypted data records $E(x)$ and $E(y)$. As discussed in "The monotonically increasing and one-way function", the function f is a one-way function, so it is impossible for S_1 to recover $dist^2(x, y)$ from $f(dist^2(x, y))$. If there is no function f and S_2 directly returns $dist^2(x, y)$ to S_1, S_1 is likely to recover the data content from a set of encrypted data records by running the Euclidean distance computation protocol. In specific, suppose that S_1 has n data records, then he/she has $l \times n$ unknown numbers because each encrypted data record is l-dimensional. In addition, S_1 could obtain at most $\frac{n(n-1)}{2}$ distances by running the Euclidean distance computation protocol for any two encrypted data records. When $l \times n \leq \frac{n(n-1)}{2}$, i.e., $2 \times l + 1 \leq n$, it is possible for S_1 to solve $l \times n$ unknown numbers from $\frac{n(n-1)}{2}$ equations and further obtain the data content of these n data records. Thus, in our scheme, the function f is applied to prevent the above attack. With the one-way function f, S_1 cannot recover d^2 from $f(d^2)$, let alone the data content. As a result, in the Euclidean distance computation protocol, S_1 cannot obtain the data content from encrypted data.

From the above analysis, we can know that both S_1 and S_2 cannot obtain any data content from the Euclidean distance computation protocol. Similarly, unauthorized users also cannot. Therefore, the Euclidean distance computation protocol is privacy-preserving.

Performance evaluation

In this section, we evaluate the performance of our proposed scheme in terms of computational cost. The evaluation includes theoretical analysis and experimental analysis.

Theoretical analysis

In this subsection, we theoretically analyze the performance of our proposed scheme in terms of computational cost. In specific, we analyze the computational complexity for each kind of entity, i.e., data owner, cloud servers (S_1 and S_2), and users in our proposed scheme.

For the data owner, the computational complexity comes from two aspects, i.e., i) building and encrypting the ld-tree in the *Outsourcing Encrypted Data to the Cloud* phase and ii) encrypting the updated data record in the *Maintaining Encrypted Data in the Cloud* phase. Suppose that the initial dataset of data owner contains N data records and each of them is l-dimensional. Then, in the *Outsourcing Encrypted Data to the Cloud* phase, building an ld-tree for the initial dataset requires $O(l \times N)$ computational complexity and encrypting these N data records requires $O(l \times N \times \log N)$ computational complexity. Note that the computational cost in the *Outsourcing Encrypted Data to the Cloud* phase can be negligible since it happens only once. In the *Maintaining Encrypted Data in the Cloud* phase, data owner needs to encrypt an l-dimensional updated data record, and as described in 4.2, encrypting one data record requires $O(l+1)$ computational complexity.

For the cloud server S_1 and S_2, the computational cost comes from the insertion, deletion and k-NN query, where the computational complexity of insertion is the same as that of deletion, i.e., $O(\log N)$. However, in the *k-NN query*

phase, the average computational complexity is $O(l \times k \times \log N)$ according to the Algorithm 3. As a result, the average computational complexity for the cloud is $O(l \times k \times \log N)$.

For each user $U_i \in \mathcal{U}$, as described in 4.2.4, when sending a k-NN query request to the cloud server S_1, U_i needs to encrypt an l-dimensional query data record, which requires $O(l)$ time complexity. In addition, on receiving the k-NN query results, he/she needs to use the access key ak to recover the query results, which requires $O(k)$ computational complexity. As a result, the computational complexity at the end of the user is $O(l + k)$.

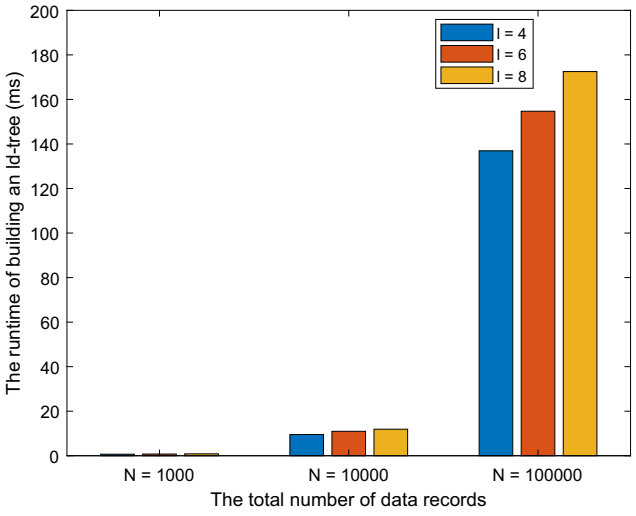
Experimental analysis

In this subsection, we experimentally evaluate the performance of the proposed scheme in term of the computational cost from the perspective of the data owner and the cloud respectively. We have implemented our proposed scheme in Java and conduct experiments on a Windows machine with Intel(R) Core(TM) i7-3770 CPU @3.40GHz processor and 16GB RAM. In our experiment, we set $\kappa = 512$, $\alpha = 256$ and $\beta = 128$, thus $|p| = |q| = 512$ and $|n| = 1024$. At the same time, we set the length of the access key is 128 bits, i.e., $|ak| = 128$. In addition, we evaluate our proposed scheme on three synthetic datasets with uniform distribution and counts of these datasets are 1000, 10000 and 100000, respectively. We run our experiment for multiple times and the average results are reported.

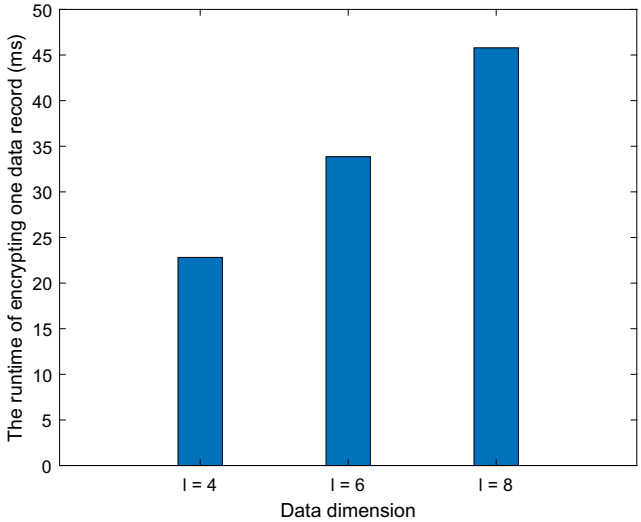
- *Computational Cost at Data Owner:* The computational cost at the data owner consists of two components, i.e., building and encrypting the ld-tree in the *Outsourcing Encrypted Data to the Cloud* phase, and encrypting the updated data record in the *Maintaining Encrypted Data in the Cloud* phase. Since encrypting the ld-tree and updating encrypted data in the cloud are based on one data record encryption. Thus, we mainly discuss the computational cost of building an ld-tree and encrypting one data record.

  First, Fig. 4a presents the the runtime of building an ld-tree varying with the number of data records N and the data dimension l. From Fig. 4a, we can see that the runtime of building an ld-tree greatly increases with the total number of data records N for a fixed data dimension l. For example, when l is equal to 6, the runtime of building tree is about 0.76 ms, 10.97 ms and 154.7 ms, respectively. A similar trend can be found in the relationship between the runtime of building ld-tree and the data dimension l, but the growth is slower. Overall, the running time of building an ld-tree increases with the total number of data records and the data dimension.

  Second, for the data record encryption, the computational cost is about $2 \times l$ exponentiation and l multiplication operations in Z_{n^2} and one AES operation. We take the synthetic dataset of size 1000 as an example to evaluate the relationship between the encryption time and data dimension, as shown in Fig. 4b. From Fig. 4b, we can see that the encryption time linearly grows with the data dimension. For example, the encryption time is almost 22.8 ms, 33.8 ms and 45.7 ms with dimension $l = 4$, $l = 6$ and $l = 8$, respectively.
- *Computational Cost at Cloud Servers:* We consider the computational cost of both cloud servers S_1 and S_2, which is comprised of two components, i.e., maintaining the encrypted ld-tree including insertion



(a) The runtime of building an ld-tree varying with N and l



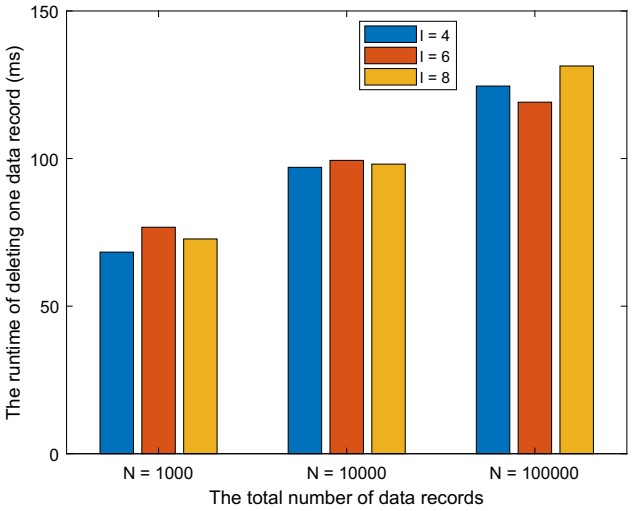
(b) The runtime of encrypting one data record varying with l

Fig. 4 Experimental results at the data owner

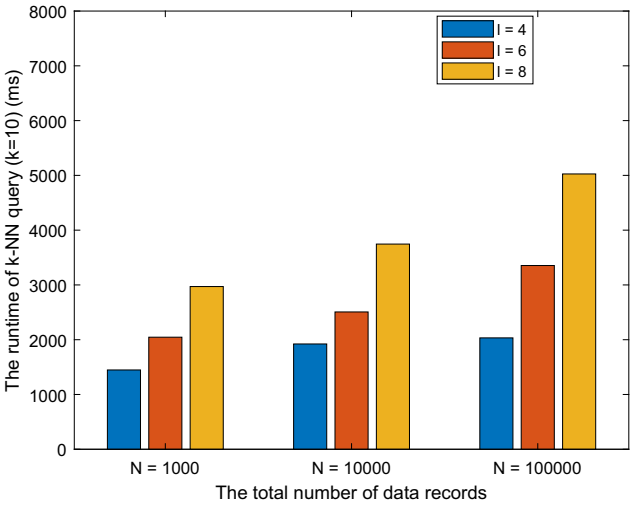
and deletion, and processing the k-NN query requests from authorized users.

– *The computational cost of maintaining the encrypted ld-tree:* The ld-tree updating includes insertion and deletion. The theoretical complexity of insertion and deletion is the same, i.e, $O(\log N)$, so we just evaluate the runtime of deletion varying with the total number of data records (i.e., N) and the data dimension (i.e., l), as shown in Fig. 5a. From Fig. 5a, we can see that the runtime of deletion logarithmically grows with the number of data records N. For instance, when $l = 6$, the average runtime of deletion are 76.73 ms, 99.38 ms and 119.1 ms with respect to $N = 1000$, 10000, 100000. However, there is no obvious difference in the runtime of deletion among different data dimensions for the same dataset.

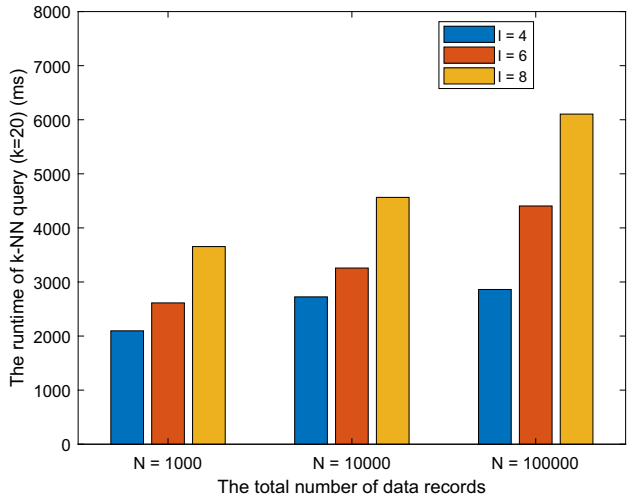
– *The computational cost of k-NN query:* The k-NN query complexity comes from searching the k-nearest neighbours for a given data record and the theoretical complexity of the k-NN query is $O(lk \log N)$. Next, we evaluate the k-NN query complexity by three synthetic datasets and consider three different cases, i.e., $k = 10$, $k = 20$ and $k = 30$ as shown in Fig. 5b, c



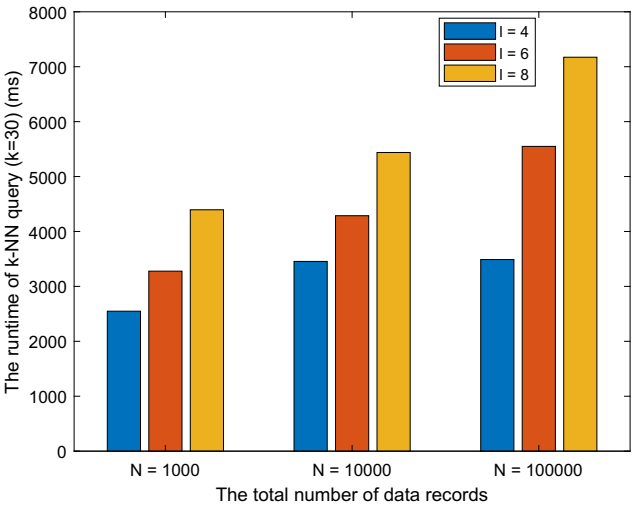
(a) The runtime of deleting one data record varying with N and l



(b) The runtime of k-NN query varying with N and l ($k = 10$)



(c) The runtime of k-NN query varying with N and l ($k = 20$)



(d) The runtime of k-NN query varying with N and l ($k = 30$)

Fig. 5 Experimental results at the cloud servers

and d respectively. These three figures have the same characteristics and show that the average runtime of k-NN query linearly grows with the data dimension when fixing the total number of data records N and also logarithmically increases with N when fixing data dimension l. In addition, the runtime of k-NN query shows an increasing trend with the increase of k value. For instance, when $N = 10000$ and $l = 6$, the runtime of k-NN query is about 2506 ms, 3257 ms and 4286 ms with respect to $k = 10$, 20, and 30 respectively.

Related work

In this section, we briefly review some related works regarding the privacy-preserving k-NN query. First, k-NN query is a fundamental problem in various areas, such as data mining, pattern recognition, similarity search and so on. Overall, there are two strategies to address this problem. According to whether the data is encrypted or not, strategies are divided into centralized methods and distributed methods. Some existing data distributed methods to achieve secure k-NN query [7, 16, 17] can only process the data in plaintext format, and cannot process query service over encrypted data. For this reason, we here only consider the centralized model.

In the centralized model, data are outsourced to an untrusted cloud server without encryption. The data owner is responsible for managing the data and offering some query service (e.g. k-NN) to the trusted clients. Due to the involvement of the untrusted server, security issues arise, especially for the data privacy. To solve this problem, data anonymization models or cryptographic techniques should be adopted to protect the data security. For example, data is encrypted before outsourced to the cloud.

Encryption is a good approach to protect data privacy. However, how to process the query problem on encrypted data becomes a new challenge, as it is not so straight as the operation in the plaintext data. In the literature, there are various research results on range query [2, 9, 10] and aggregation query [8] for encrypted data. Since k-NN query is one of the most important applications in eHealthcare data, we focus on discussing k-NN query over encrypted data in this work.

Aiming at addressing the k-NN query problem, different techniques have been proposed [4, 11, 18, 19, 21]. A novel encryption scheme was proposed by Wong et al. [18]. In this scheme, the scalar product between the query vector and any tuple vector from database for distance comparison is preserved and the data owner adopted different encryption algorithms for data and query. Zhu et al. [24] improved the scheme of Wong et al. to provide privacy-preserving k-NN query. However, the data owner has to participate in the query process and the scheme lacks a rigorous security proof. Hu et al. [11] proposed a k-NN query scheme based on privacy homomorphism encryption scheme. In their scheme, the k-NN query on the encrypted data is achieved by homomorphic properties. However, Yao et al. [21] pointed out that both schemes in [11, 18] cannot resist chosen-plaintext attacks, and thus they proposed a new scheme based on partition-based secure Voroni diagram. Elmehdwi et al. [4] designed a homomorphic encryption based k-NN query scheme over encrypted data. Although the data owner and the client achieve the privacy, the computation and communication overheads are not very efficient. Recently, Xu et al. [19] also proposed a scheme with the sublinear computational complexity during the k-NN query process. They achieved k-NN computation over encrypted data by using the garbled circuit between two non-colluding cloud servers. Different from Xu et al.'s scheme, our proposed scheme computes k nearest neighbors over encrypted data by designing privacy-preserving data comparison protocol and Euclidean distance computation protocol.

Conclusion

In this paper, we have proposed a new efficient and privacy-preserving k-NN query scheme over encrypted eHelathcare data in the cloud by integrating kd-tree and homomorphic encryption techniques. The proposed scheme achieves the following two characteristics i) efficient storing encrypted data in the cloud and ii) privacy-preserving k-NN query over encrypted data. Specifically, two privacy-preserving protocols for data comparison and Euclidean distance computation are introduced for the proposed scheme. Security analysis shows that our proposed scheme is privacy-preserving. In addition, performance evaluation also indicates that our proposed scheme is efficient in terms of computational complexity. In our future work, we will evaluate the proposed scheme on real eHealthcare datasets.

Funding Information This research was supported in part by NSERC Discovery Grants (04009), NBIf Start-Up Grant (Rif 2017-012), HMF2017 YS-04, LMCRF-S-2018-03, ZJNSF under Grant LZ18F020003 and NSFC under Grants 61472364.

Compliance with Ethical Standards

Conflict of interests The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Philips healthcare case study. https://aws.amazon.com/cn/solutions/case-studies/philips/.
2. Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y., Order-preserving encryption for numeric data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 563–574, 2004.
3. Beis, J. S., and Lowe, D. G., Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), pp. 1000–1006, 1997.
4. Elmehdwi, Y., Samanthula, B. K., and Jiang, W., Secure k-nearest neighbor query over encrypted data in outsourced environments. In: International Conference on Data Engineering, pp. 664–675, 2014.
5. Firouzi, F., Rahmani, A. M., Mankodiya, K., Badaroglu, M., Merrett, G. V., Wong, P., and Farahani, B., Internet-of-things and big data for smarter healthcare: From device to architecture, applications and analytics. *Futur. Gener. Comput. Syst.* 78:583–586, 2018.
6. Friedman, J. H., Baskett, F., and Shustek, L. J., An algorithm for finding nearest neighbors. *IEEE Trans. Comput.* 24(10):1000–1006, 1975.
7. Ghinita, G., Kalnis, P., Khoshgozaran, A., Shahabi, C., and Tan, K., Private queries in location based services: anonymizers are not necessary. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 121–132, 2008.
8. Hacigümüs, H., Iyer, B. R., and Mehrotra, S., Efficient execution of aggregation queries over encrypted relational databases. In: Database Systems for Advances Applications, pp. 125–136, 2004.
9. Hore, B., Mehrotra, S., Canim, M., and Kantarcioglu, M., Secure multidimensional range queries over outsourced data. *VLDB J.* 21(3):333–358, 2012.
10. Hore, B., Mehrotra, S., and Tsudik, G., A privacy-preserving index for range queries. In: (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, pp. 720–731, 2004.
11. Hu, H., Xu, J., Ren, C., and Choi, B., Processing private queries over untrusted data cloud through privacy homomorphism. In: Proceedings of the 27th International Conference on Data Engineering, pp. 601–612, 2011.
12. Huang, C., Lu, R., Zhu, H., Shao, J., and Lin, X., 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, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016, pp. 95–106, 2016.
13. Jiang, R., Lu, R., and Choo, K. R., Achieving high performance and privacy-preserving query over encrypted multidimensional big metering data. *Futur. Gener. Comput. Syst.* 78:392–401, 2018.
14. Karuppiah, M., Li, X., Das, A. K., Kumari, S., and Liu, Q., Introduction to the special section on big data and iot in e-healthcare. *Comput. Electr. Eng.* 65:261–264, 2018.
15. Paillier, P., Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, pp. 223–238, 1999.
16. Qi, Y., and Atallah, M. J., Efficient privacy-preserving k-nearest neighbor search. In: 28th IEEE International Conference on Distributed Computing Systems, pp. 311–319, 2008.
17. Shaneck, M., Kim, Y., and Kumar, V., Privacy preserving nearest neighbor search. In: Workshops Proceedings of the 6th IEEE International Conference on Data Mining, pp. 541–545, 2006.
18. Wong, W. K., Cheung, D. W., Kao, B., and Mamoulis, N., Secure knn computation on encrypted databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 139–152, 2009.
19. Xu, R., Morozov, K., Yang, Y., Zhou, J., and Takagi, T., Efficient outsourcing of secure k-nearest neighbour query over encrypted database. *Comput. Secur.* 69:65–83, 2017.
20. Yang, X., Lu, R., Choo, K. K. R., Yin, F., and Tang, X., Achieving efficient and privacy-preserving cross-domain big data deduplication in cloud. IEEE Transactions on Big Data, 2017.
21. Yao, B., Li, F., and Xiao, X., Secure nearest neighbor revisited. In: 29th IEEE International Conference on Data Engineering, pp. 733–744, 2013.
22. Yekta, N. I., and Lu, R., Xrquery: Achieving communication-efficient privacy-preserving query for fog-enhanced iot. In: 2018 IEEE International conference on communications, ICC 2018, Kansas city, May 20-24, 2018, pp. 1–6, 2018.
23. Zheng, Y., and Lu, R., An efficient and privacy-preserving k-nn query scheme for ehealthcare data. In: 2018 IEEE Conference on green computing and communications, pp. 358–365, 2018.
24. Zhu, Y., Xu, R., and Takagi, T., Secure k-nn computation on encrypted cloud data without sharing key with query users. In: Proceedings of the 2013 International Workshop on Security in Cloud Computing, pp. 55–60, 2013.