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

Yandong Zheng, Rongxing Lu* and Jun Shao

Contact Email: rlu1@unb.ca

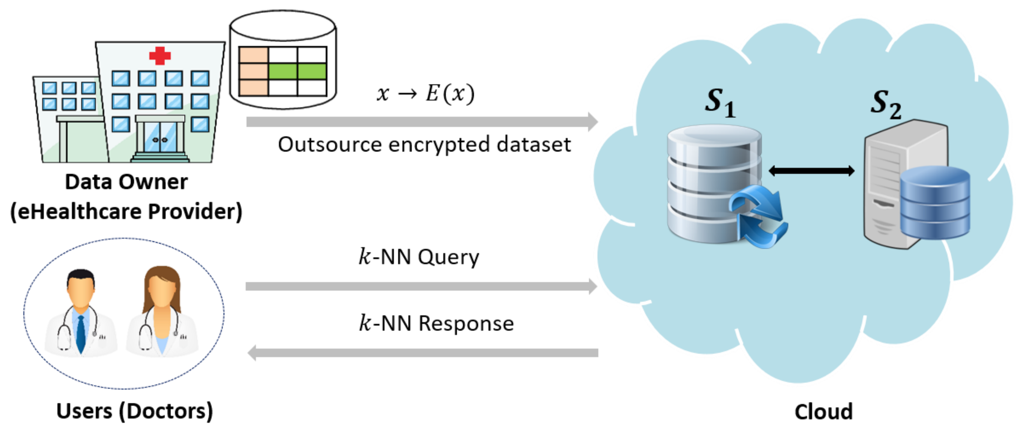
Canadian Institute for Cybersecurity (CIC), Faculty of Computer Science, University of New Brunswick (UNB)

ABSTRACT

The boom of Internet of Things devices promotes huge volumes of eHealthcare data are collected and aggregated at eHealthcare provider. As eHealthcare data are very sensitive yet cloud servers are not fully trusted today, many security, privacy and efficiency challenges will arise when cloud meets eHealthcare data. In this work, aiming at addressing the privacy and efficiency challenges, we present an efficient and privacy-preserving k Nearest Neighbors (k-NN) query scheme for encrypted eHealthcare data in cloud. The proposed scheme is characterized by integrating kd-tree, homomorphic encryption technique as well as a proposed monotonically increasing and one-way function for efficient storing encrypted data in the cloud and privacy-preserving k-NN query over encrypted data.

System Model

- **Data Owner**: Data owner outsources encrypted data to the cloud.
- **Cloud Server** $CS = \{S_1, S_2\}$: S_1 and S_2 cooperate to store outsourced data and process k-NN query requests from users.
- **Users** $U = \{U_1, U_2 \cdots\}$: Each U_i can request a k-NN query to the cloud and receive the desirable result.



Security Model

- **Data Owner:** The data owner is considered to be honest.
- **Cloud Server:** Both S_1 and S_2 are honest-but-curious, but they are not allowed to collude.
- **Users:** The authorized users are honest, but unauthorized users may launch some malicious attacks.

Design Goals

- Privacy preservation: The data stored in the cloud and the k-NN query records and corresponding query results should be privacy-preserving.
- Computation efficiency: The proposed scheme should be computation efficient in terms of k-NN query.

A. kd-tree Technique

Definition of k-tree: The kd-tree is a binary tree and each tree node is a k dimensional data record. Meanwhile, each tree node x contains four attributes cd, $data$, $left$, and $right$, which denote the cutting dimension, key value, left child and right child, respectively. We use $x.cd$, $x.data$, $x.left$ and $x.right$ to denote attributes of x, respectively. In addition, the kd-tree satisfies order relation, i.e., for each tree node x,

$$x.left.data[x.cd] \leq x.data[x.cd] < x.right.data[x.cd].$$

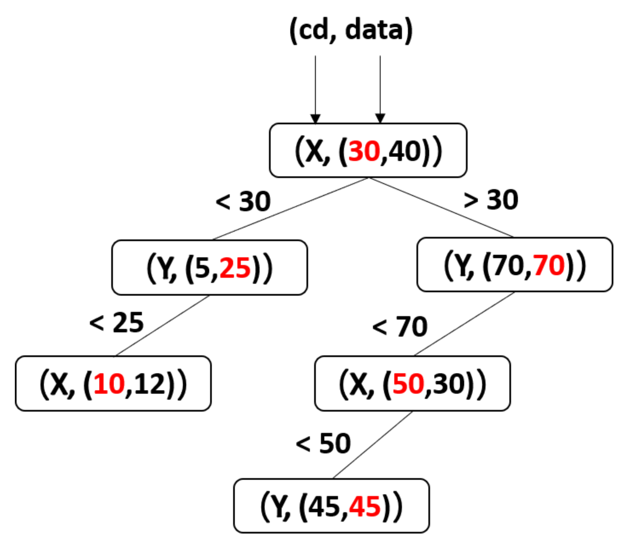


Fig. 1. An example of kd-tree

Algorithm of k-NN query: As shown in Algorithm 3, in the k-NN query, we adopt the *Best Bin First Search* strategy, which will give a higher searching priority for those subtrees that are more likely to contain the k nearest data records. At the same time, we use a priority queue PQ with size k to store the current top k closest data records with the query data record x. 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 $(x[cd] - T.data[cd])^2$.

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

```
if T == null then
    return PQ
end if
if size(PQ) < k or dist(x, T.data) <dist(PQ_1, x) then
    PQ.enqueue(T.data)
    Adjust the order of the queue
    bestdist = dist(PQ_1, x)
end if
//Best Bin First Search strategy
if x[T.cd] ≤ T.data[T.cd] then
    k-NN(x, T.left, PQ)
else
    k-NN(x, T.right, PQ)
end if
//Prune strategy
if (x[cd] − T.data[cd])^2 < bestdist then
    if x[cd] ≤ T.data[cd] then
        k-NN(x, T.right, PQ)
    else
        k-NN(x, T.left, PQ)
    end if
end if
```

B. The Monotonically Increasing and One-way Function

Suppose $D = \{x = (x_1, x_2, \cdots, x_l) | x_i \in Z^+,\ x_i \leq U, i = 1,2, \cdots, l\}$ is an l dimensional dataset, where U is the upper bound of all data values in D. 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 $d^2 \in DS$ to $f(d^2)$. In specific, for each $d^2 \in DS$, $f(d^2)$ is

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

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 $(\Delta, a_1 + a_2 + \cdots + a_n)$.

Theorem 1: The function f is a monotonically increasing and one-way function.

C. The Proposed k-NN Query Scheme

(1) System Initialization

Data owner bootstraps the whole system. In specific, the data owner generates Paillier's public key pk and private key sk, randomly selects access key ak and chooses AES algorithm as the basic encryption algorithm. Then, it will publish the public key pk, and distribute sk to cloud server S_2. At the same time, it also sends ak to each authorized user.

(2) Outsourcing Encrypted Data to the Cloud

The data owner encrypts and outsources dataset $D = \{x = (x_1, x_2, \cdots, x_l)\}$as follows.

Step-1: Build an ld-tree for the dataset D using ld-tree building algorithm.

Step-2: Encrypt the built tree. In specific, for the key value of tree node $x = (x_1, x_2, \cdots, x_l)$, the data owner encrypts it as $E(x) = (E(x_1),\ E(x_2), \cdots,\ E(x_l),\ AES_{ak}(x))$, and outsources the encrypted ld-tree to the cloud server S_1.

(3) Maintaining Encrypted Data in the Cloud

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

Insertion: Data owner can insert $x = (x_1, x_2, \cdots, x_l)$ into the encrypted ld-tree as follows. First, data owner sends $E(x) = (E(x_1),\ E(x_2), \cdots,\ E(x_l),\ AES_{ak}(x))$ to S_1. Then, S_1 inserts $E(x)$ into the encrypted ld-tree by the insertion algorithm. Since S_1 cannot access the plaintext ld-tree, he/she will face a challenge when running the insertion algorithm, i.e., how to compare two encrypted data. This can be solved by running the privacy-preserving data comparison protocol.

Deletion: Data owner can delete $x = (x_1, x_2, \cdots, x_l)$ from the encrypted ld-tree. First, he/she $E(x) = (E(x_1),\ E(x_2), \cdots,\ E(x_l), AES_{ak}(x))$ to S_1. Then, S_1 deletes $E(x)$ from the encrypted ld-tree according to the deletion algorithm. During the deletion process, S_1 also solves the encrypted data comparison by the privacy-preserving data comparison protocol.

(4) k-NN Query over Encrypted Data

User U_i can query the k nearest data records with $y = (y_1, y_2, \cdots, y_l)$ as the following steps.

Step-1: U_i encrypts y as $E(y) = (E(y_1), E(y_2), \cdots, E(y_l))$, and sends a k-NN query request as well as $E(y)$ to the cloud server S_1.

Step-2: On receiving the query, S_1 cooperates with S_2 to run the k-NN query algorithm over encrypted ld-tree. Similarly, encrypted data comparison can be processed by the privacy-preserving data comparison protocol, and Euclidean distance computation can be computed by the privacy-preserving Euclidean distance computation protocol, which integrates the permutation technique with a monotonically increasing and one-way function f. After running k-NN algorithm, the query results (i.e., k nearest data records) are stored in PQ.

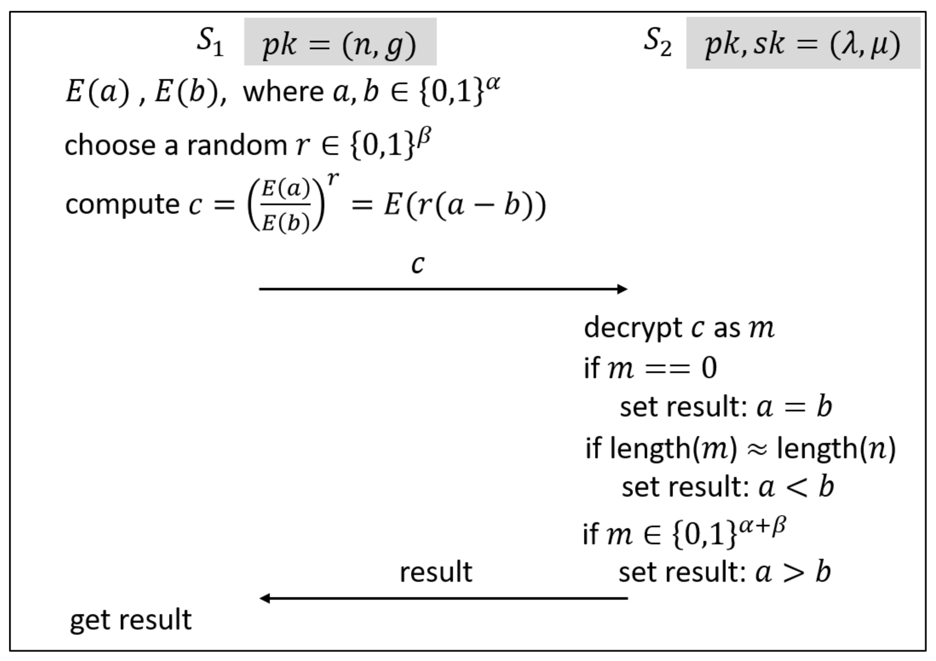


Fig. 2. Privacy-preserving data comparison protocol

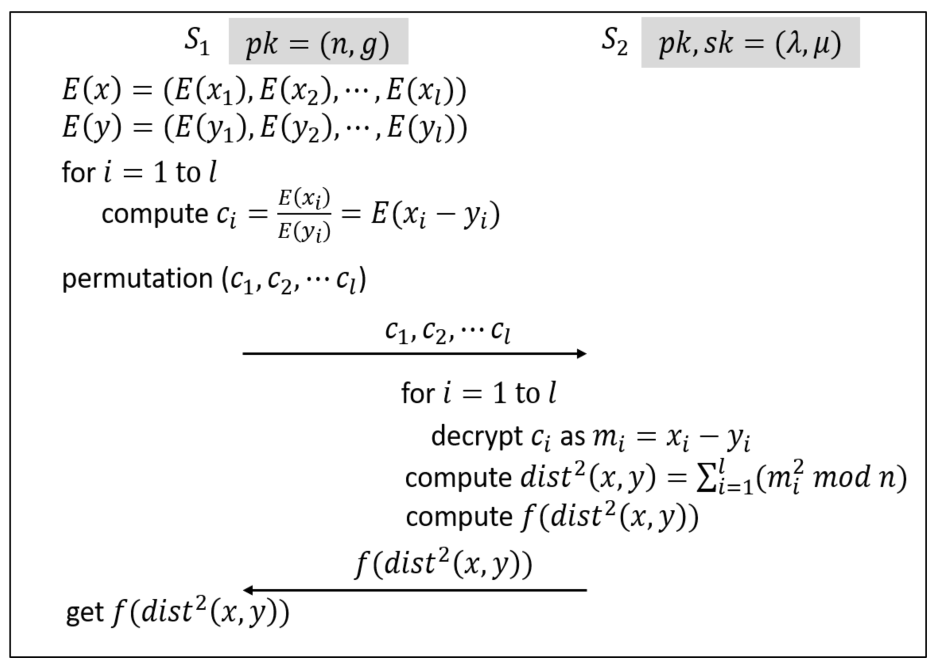


Fig. 3. Privacy-preserving Euclidean distance computation protocol

*This work has been accepted by the Journal of Medical Systems.