Peer-to-Peer Networking and Applications (2022) 15:1076–1089
https://doi.org/10.1007/s12083-021-01290-4

Updatable privacy-preserving *K*-nearest neighbor query in location-based s-ervice

Songyang Wu[1] · Wenju Xu[1] · Zhiyong Hong[2] · Pu Duan[3] · Benyu Zhang[3] · Yupu Hu[1] · Baocang Wang[1]

Received: 6 July 2021 / Accepted: 29 December 2021 / Published online: 7 January 2022

Abstract
The *K*-nearest neighbor (*K*-NN) query is an important query in location-based service (LBS), which can query the nearest *k* points to a given point, and provide some convenient services such as interest recommendations. Hence the privacy protection issue of *K*-NN query has been a popular research area, protecting the information of queries and the queried results, especially in the information era. However, most of existing schemes fail to consider the privacy protection of location points already stored on servers. Or some schemes support no update of location points. In this paper, we present an updatable and privacy-preserving *K*-NN query scheme to address the above two issues. Concretely, our scheme utilizes the *K*D-tree (*K*-Dimensional tree) to store the location points of data owners in location service provider and encrypts the points with a distributed double-trapdoor public-key cryptosystem. Then, based on the Ciphertext Comparison Protocol and Ciphertext Euclidean Distance Calculation Protocol, our scheme can protect the privacy of location and query contents. Experimental analyses show our proposal supports some new location points for a fixed location service provider. Moreover, the queried results show a high accuracy of more than 95%.

Keywords Location-based service · *K*-nearest neighbor · Privacy protection · *K*D-tree · Data security update

1 Introduction

Due to the development of the mobile Internet and positioning technology, LBS (location-based service) brings a great deal of convenience to people's lives and is applied in many fields of life, such as social networks, smart travel and interest recommendations, etc. In the case of contagion of COVID-19, health codes allow residents to record their movements by scanning QR (quick response) codes as they pass through particular locations. Authorized testing agencies can thus analyze the location information and health status of people in order to track and isolate infected and potentially infected people in a timely manner.

While bringing convenience, LBS also brings serious privacy and security problems. When some user wants to get a location service, he usually needs to upload his location information and other identity-related information to the LSP (location service provider). After obtaining the information from the user, the service provider offers the location service and sends the service results to the user. But during the interaction process, the LSP can fully grasp the user's location or even more private and sensitive information. When the LSP gets the user's location information, it

✉ Wenju Xu
xuwenjuxwj@126.com

✉ Baocang Wang
bcwang@xidian.edu.cn

Songyang Wu
1393893083@qq.com

Zhiyong Hong
hzy_wyu@163.com

Pu Duan
p.duan@antgroup.com

Benyu Zhang
benyu.z@antgroup.com

Yupu Hu
yphu@mail.xidian.edu.cn

[1] The State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China

[2] Facility of Intelligence Manufacture, Wuyi University, Jiangmen 529020, China

[3] Ant Group, Hangzhou 310000, China

may immediately determine the user's travel route or sell the user's personal location information to a malicious third party. More seriously, LSP may infer the users' habits, health status [1], family, social relationships [2, 3] and other personal sensitive information. Once an attacker gets hold of this information, in extreme cases, it may even bring damage to the user's life and property security, causing unimaginable consequences and loss.

Hence, it is necessary to address the privacy protection in LBS. Over the years, many researchers have contributed to solving this problem. In 2003, Beresford and Stajano [4] proposed the concept of location privacy protection for the first time. Many privacy protection techniques have been proposed according to different application scenarios and protection means. They can be broadly classified according to the privacy protection techniques used: spatial obfuscation techniques [5–7], data transformation techniques [8], false location perturbation techniques [9], and encryption techniques [10–12].

Spatial Obfuscation: The *k*-anonymity technique is one of the main means to achieve spatial obfuscation, first proposed by Sweeney [13] in 2002 and applied to the field of location privacy protection by Gruteser and Grunwald [14]. The user forms an anonymity zone with the location information of *k* other users using fuzzy data acquisition services, thus protecting the user's location privacy so that an attacker can not associate the query information with the user. The R-tree structure maintaining the users' location data and the Casper model was proposed by Mokbel et al. [15] in order to improve the user's anonymity security level. Bamba et al. [16] applied the *l*-diversity policy to increase the success rate of anonymity using temporal properties.

Data Transformation: The principle of data transformation technique is to map the user's location coordinates and information data to another data space. The most representative is the privacy preserving *K*-NN query method based on the Hilbert curve proposed by Khoshgozaran and Shahabi [17]. Lien et al. [18] used the Moore curve as the medium of spatial mapping and proposed a secret cyclic query protocol that gets rid of the third-party anonymous server.

False Location Perturbation: False location perturbation was first proposed by Kapadia et al. [19] as a way of querying through false position points instead of real position points. After a user makes a query by adding location points and receives the queried result, the queried result corresponding to the real location points is confirmed by the geometric relationship among the fake location points, the user's real location points and the query result location points. Yiu et al. [20] proposed the well-known SpaceTwist algorithm to search the user's points of interest through anchor points. Gong et al. [21] improved on the Yiu's scheme based on the central server so that the SpaceTwist algorithm satisfies the *k*-anonymity constraint.

Encryption Techniques: The core of encryption technique is to protect the location privacy of users by directly encrypting the location data using cryptographic algorithm to avoid location service providers or other third parties from obtaining the user's location information. Lian et al. [22] proposed a fast location *K*-NN query scheme for outsourced data based on space filling curve and AES encryption algorithm. Utsunomiya et al. [23] proposed a privacy-preserving *K*-NN query scheme via Hilbert curve and NTRU algorithm.

The above schemes present location privacy protection methods in various application scenarios, but their schemes are carried out for fixed interest point datasets and not suitable for updatable datasets supporting new location points. For query scenarios that require frequent updates of interest point sets, they are not as effective as they should be while ensuring query privacy for the query user.

To address the above problem, combining the above mentioned background of epidemic prevention, this paper proposes a privacy-preserving *K*-NN(nearest neighbor) query scheme to support points security update. The scheme encrypts the data owner's location information by the DT-PKC (double-trapdoor public key cryptosystem) [24] homomorphic algorithm and stores it in the form of *K*D-tree nodes in the cloud server to protect the location privacy. Ciphertext Comparison Protocol and Ciphertext Euclidean Distance Calculation Protocol are constructed under the double-cloud model. Cloud server runs the protocol to make the insertion and deletion algorithms of the *K*D-tree. The *K*-nearest neighbor query algorithm of the *K*D-tree are applicable to the encrypted data nodes, while they are originally only applicable to plaintext node data. The scheme effectively protects the privacy of the location information uploaded by the user and the query information of the querier. The accuracy of the query is relatively higher.

The remainder of our paper is structured as follows. Section 2 introduces some necessary background knowledge. The system model and design goal are given in Sect. 3. In Sect. 4, we describe the specific construction of the scheme. Moreover, Sects. 5 and 6 discuss the security analysis and performance evaluation, respectively. Finally, we present the conclusion in Sect. 7.

2 Preliminaries

In this section, we first introduce some basic notations in Table 1. Then, we briefly introduce two necessary background techniques, DT-PKC scheme and *K*D-tree technique, for a better understanding of our proposed scheme later.

Table 1 Notations

Notation	Descriptions
$\bmod n$	$\{0, 1, 2, \ldots, n-1\}$
$\mathbb{Z}$	The integer ring
$\mathbb{Z}_{N^2}$	$\mathbb{Z}_{N^2} = \{0, 1, 2, \ldots, N^2 - 1\}$
$\mathbb{Z}^*_{N^2}$	$\mathbb{Z}^*_{N^2} = \{a \mid 0 < a < N^2, \gcd(a, N^2) = 1\}$
$\mathcal{L}(p)$	The bit length of the integer p
$\text{lcm}(a, b)$	Least common multiple of a and b
k	Query parameter of K-NN
$[\cdot]_{pk_i}$	The encryption under the public key pk_i

2.1 DT-PKC

The DT-PKC scheme was presented by Liu et al. [24], based on BCP cryptographic system [25]. Furthermore, the DT-PKC is suitable for multi-user environments. The DT-PKC works as follows.

Key Generation (KeyGen): Given a security parameter κ, generate two large primes p and q, s.t. $\mathcal{L}(p) = \mathcal{L}(q) = k$. Then compute $N = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. Define function $L(x) = (x-1)/N$, then choose an element g of order $\text{lcm}(p-1, q-1)$ from $\mathbb{Z}_{N^2}$. Next, randomly select an integer $\theta_i \in [1, N/4]$ (See [24] for details.) for the i-th party and compute $h_i = g^{\theta_i} \pmod{N^2}$.

The public key for the i-th party is $pk_i = (N, g, h_i)$, and the weak private key is $sk_i = \theta_i$. The strong private key is $SK = \lambda$.

Encryption (Enc): Given a message $m \in \mathbb{Z}_N$, choose a random integer $r \in [1, N/4]$. The ciphertext under pk_i generated as $[m]_{pk_i} = (T_{i,1}, T_{i,2})$, in which $T_{i,1} = h^r(1 + mN) \pmod{N^2}$ and $T_{i,2} = g^r \pmod{N^2}$.

Decryption With Weak Private Key (WDec): Input a ciphertext $[m]_{pk_i}$, decrypt it under the weak private key $sk_i = \theta_i$ and get the plaintext

$$m = L\left(\frac{T_{i,1}}{T_{i,2}^{\theta_i}} \bmod N^2\right). \tag{1}$$

Decryption With Strong Private Key (SDec): The ciphertext $[m]_{pk_i}$ can be decrypted by $SK = \lambda$. Calculate $T_{i,1}^{\lambda} \bmod N^2$ and compute the plaintext

$$m = \frac{L\left(T_{i,1}^{\lambda} \bmod N^2\right)}{\lambda} \bmod N. \tag{2}$$

Strong private key splitting (Skey): The strong private key $SK = \lambda$ can be split into two parts. The partial strong private key $SK_j = \lambda_j (j = 1, 2)$, $s.t. \lambda_1 + \lambda_2 = 0 (\bmod \lambda)$ and $\lambda_1 + \lambda_2 = 1 \pmod{N^2}$ hold at the same time.

Partial Decryption With Strong Private Key Step One (PSDec1): Given $[m]_{pk_i}$ and a partial private key $SK_1 = \lambda_1$, output the partial decrypted ciphertext $CT_i^{(1)} = T_{i,1}^{\lambda_1} = h_i^{r\lambda_1} \cdot (1 + mN\lambda_1) \pmod{N^2}$.

Partial Decryption With Strong Private Key Step Two (PSDec2): Input $CT_i^{(1)}$ and $[m]_{pk_i}$ then execute $CT_i^{(2)} = T_{i,1}^{\lambda_2}$. At last, compute $m = L(CT_i^{(1)} \cdot CT_i^{(2)})$.

Given $[m_1]_{pk}, [m_2]_{pk}$, denote the multiplication between $[m_1]_{pk}, [m_2]_{pk}$ as the component-wise multiplication. Then we have

$$\begin{aligned} &\text{PDec}([m_1]_{pk} \cdot [m_2]_{pk}) \\ =&\frac{L\left(\left(1 + (m_1 + m_2)N\right)^{\lambda} \bmod N^2\right)}{\lambda} \bmod N \\ =&[m_1 + m_2]_{pk} \end{aligned} \tag{3}$$

and

$$\text{PDec}([m_1]_{pk}^{N-1}) = [-m_1]_{pk}. \tag{4}$$

In this paper, we restrict m with $\mathcal{L}(m) < \mathcal{L}(N)/8$.

2.2 *K*D-Tree

The KD-tree is a kind of data structure that organizes points in a multi-dimensional Euclidean space. Each node of the KD-tree is a k-dimensional numerical point, representing a hyperplane which is perpendicular to the coordinate axis of the current division dimension. That is, if the current node is divided in dimension d, the coordinate values of all points in its left subtree in dimension d are less than the current value, and the coordinate values of all points in its right subtree in dimension d are greater than or equal to the current value. In this paper, since the query is performed on the geographic location, the dimension of the tree is 2-dimensional, i.e., $d = 2$.

A simple example of the KD-tree for $d = 2$ is depicted in Fig. 1, which is divided into eight parts. In the two-dimensional KD-tree, we take a dimensional partitioning of the point set by alternating the x-axis and y-axis. To elaborate, in the first partitioning, the x-dimension of the points is used as the partitioning dimension, their median is found (if the base of the point set is even, it can be taken down), and a vertical line is made through the node to divide the point set into two equal subsets of the base, which is the root node of the tree. The second partitioning takes the y-axis as the partitioning dimension, and each resultant subset is further partitioned along a horizontal line. The process is repeated until the base number of nodes drops below a certain threshold.

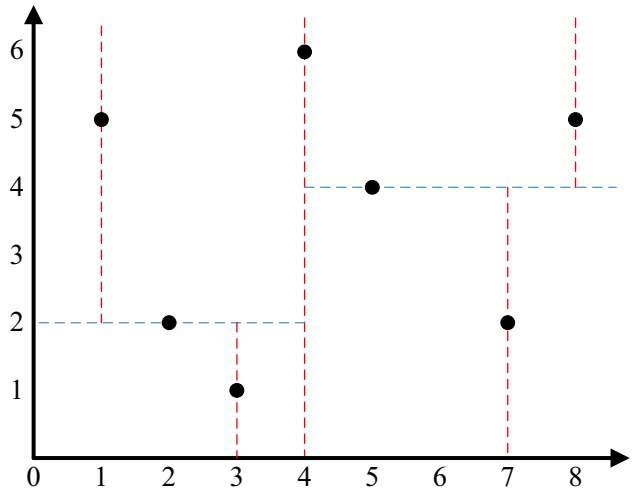


Fig. 1 An example of the *K*D-tree for $d = 2$

3 Models and design goal

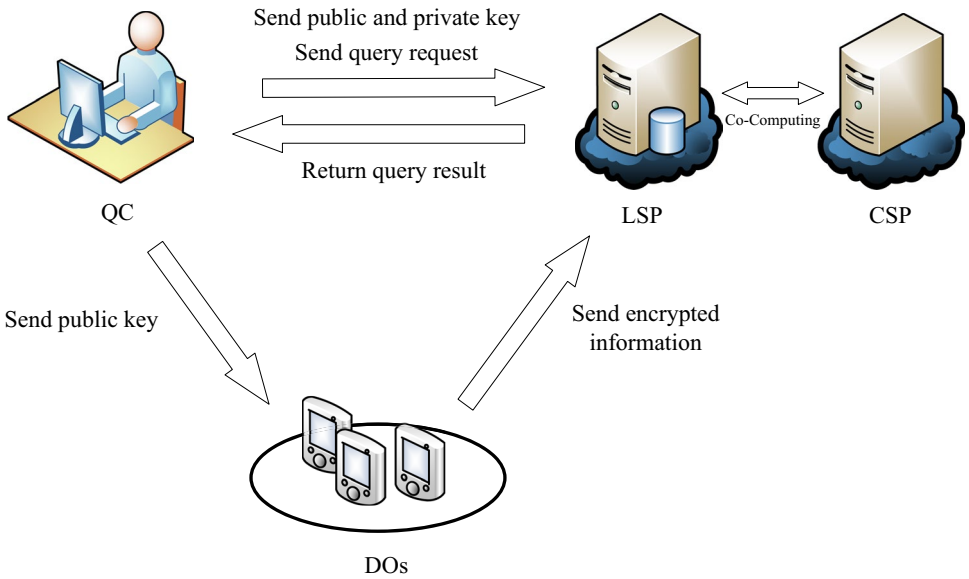
In this section, we formalize our system model, threat model and design goal.

3.1 System model

In our system model, four entities are shown in Fig. 2, including query center, data owners, location server provider and compute server provider.

- **Query Center (QC):** In this model, the monitoring center of the epidemic acts as the query center. It generates and distributes public and private keys during the system initialization. In addition, after the system starts to run, query center makes *K*-NN query requests to the cloud server. It is important to note that the *K*-NN query in this scheme is based on Euclidean distance.
- **Data Owners (DOs):** In this paper, the data owners are a large number of independent individuals who upload their location data to the cloud server with encryption through mobile terminals. The set of data owners is denoted as $D = \{d_i | 1 \le i \le I, I > 1\}$, where d_i denotes the i-th data owner and I is the total number of data owners. The corresponding set of location information is $P = \{p_i = (x_i, y_i, u_i) | 1 \le i \le I, I > 1)\}$, where x_i and y_i denote the horizontal and vertical coordinates of d_i's location in the two-dimensional plane, u_i denotes the identity information and health status of d_i. In addition, due to the mobility of the population, the provided data have to be updated and maintained frequently, e.g., new location information data need to be inserted into the database when people register, or obsolete data need to be deleted when people leave, etc.
- **Location Service Provider (LSP):** In this model, LSP has rich storage resources. It saves the storage of all the encrypted location information data uploaded by data owners, records the intermediate and final results generated in the query process. LSP provides computations over encrypted location information data and sends queried results to the query center.
- **Compute Service Provider (CSP):** CSP provides online computing service in the model. Together with LSP, it provides privacy-protecting *K*-NN query service to query center via protocols running in double-cloud model.

Fig. 2 System model



3.2 Threat model

In the threat model, assume that the query center is trusted: it generates and distributes public and partial private keys for LSP and CSP. And the query center is authorized to view all plaintext data and send query requests to the server so that it can make the maximum reasonable judgement on the prevention and control of the epidemic. Meanwhile, LSP and CSP are assumed to be honest-but-curious. That is, they will honestly execute protocols and return correct results, but still try to obtain sensitive information about others from the data they receive and store. We also assume there is no collude between LSP and CSP.

3.3 Design goal

In this paper, the design goal of our scheme is to present an updatable and privacy-preserving K-NN query scheme in LBS. The scheme works with the above system model and threat model. Specifically, the following objectives should be satisfied.

- **Correctness.** In our scheme, one of the objectives is to obtain the information of the k nearest location points to the query point by performing K-NN query. Therefore a high accuracy of query is what this scheme should guarantee so that the querying center can get the correct result.
- **Privacy protection.** The information of queried request, data sets and queried result are required to be protected effectively during the K-NN query scheme. Once an attacker had access to any of them, he would pose a great threat to the normal life of the data owners, such as habits, family and social relationships. In extreme cases, he may even bring damage to the user's life and property security. Hence it is necessary to ensure the privacy-preserving of K-NN query scheme.
- **Update.** Some new location points of data owners may come into being for a fixed K-NN query model. In this case, an updatable K-NN query scheme supporting these new location points is more practical. Concretely, new location points may replace the role of existing ones, or are inserted into the data sets.

4 Proposed scheme

In this section, we elaborate the K-NN query scheme including four parts: system initialization, location data upload, location data update, and K-NN query.

4.1 System initialization

System initialization consists of three steps: key generation and distribution, initialization of KD-tree and KD-tree upload.

i) Key Generation and Distribution

The query center invokes the DT-PKC's key generation algorithm to generate the public key pk_0 and strong private key mk for query center, and the public key pk_i for each data owner d_i. Then the DT-PKC's private key splitting algorithm is applied for query center to partition the master private key mk into partial private keys mk_1 and mk_2. mk_1, mk_2 are sent to LSP and CSP through trusted channel respectively.

ii) Initialization of KD-tree

We assume the query center has the initial location information set $P' = \{p_s = (x_s, y_s, u_s)\}$ before system initialization. As shown in Algorithm 1, the query center uses the KD-tree initialization algorithm to partition the location information set P' alternately with x_s and y_s as the partition dimension to obtain the KD-tree $T = \{t_s : t_s.data = p_s, t_s.right, t_s.left\}$, where t_s denotes the node in T, $t_s.data$ denotes the data stored in the node t_s, the elements in the location information correspond to the $t_s.data$ in the node of T, and $t_s.right$ and $t_s.left$ denote the right and left children of t_s, respectively.

Algorithm 1 Tree building algorithm

Input: $flag = 0$: Record the depth of the tree, PointSet P: The set of the points, Node t: KD-tree with t as root node
Output: The KD-tree

```
function BUILDTREE(flag = 0, PointSet P, Node t)
    if P = NULL then
        return NULL
    end if
    δ = the data record with median value of {x[flag] ∈ P}
    for x in P do
        if x[flag] < m[flag] then
            x → S_L
        else
            x → S_R
        end if
    end for
    t.data = m
    t.left = BuildTree(flag ⊕ 1, S_L, t.left)
    t.right = BuildTree(flag ⊕ 1, S_R, t.right)
end function
```

Algorithm 1 is the initialization algorithm of the K D-tree. We input an initial set of location information, an empty root node to store the data, and an identifier *flag* to record the dimension of the KD-tree partition. For *flag*, $flag = 0$ denotes the partition axis is x and $flag = 1$ means it is y. The median of the set is selected in each segmentation according to the corresponding dimension, and the set is divided into two subsets according to median value δ. The elements of one subset are less than δ in the corresponding dimension, and the elements of the other subset are greater than or equal to m in the

corresponding dimension. Until all data are assigned to the tree nodes, the final output is a complete KD-tree T.

iii) KD-tree Upload

The query center encrypts the $t_s.data$ of each node t_s of KD-tree T with the encryption algorithm of DT-PKC under the public key pk_0. Then the encrypted KD-tree T' with encrypted $t_s.data$ to location information vector p'_s instead of the original p_s stored in the node t_s will be uploaded to LSP, where

$$p'_s = (E_{pk_0}(p_s), pk_0) = (E_{pk_0}(x_s), E_{pk_0}(y_s), E_{pk_0}(u_s), pk_0). \tag{5}$$

4.2 Location data upload

In this section, the data owner d_i uploads his location information to the cloud via smart mobile terminal. Concretely, the data owner d_i encrypts the location data using DT-PKC's encryption algorithm with his own public key pk_i. Then he can go offline after uploading the encrypted data to LSP until the next update, which does not affect the storage of the data in the server. The location information vector p'_i for d_i is denoted by

$$p'_i = (E_{pk_i}(p_i), pk_i) = (E_{pk_i}(x_i), E_{pk_i}(y_i), E_{pk_i}(u_i), pk_i). \tag{6}$$

4.3 Location data update

After verifying the data owner d_i, LSP executes the KD-tree deletion algorithm to remove the old location node of d_i from the encrypted KD-tree, then uses the KD-tree insertion algorithm to add the newly encrypted node.

i) Insertion

Algorithm 2 describes how to insert an encrypted location vector into an encrypted KD-tree T'. There will be a recursive search for the new data to be inserted in the left and right subtrees until the data is successful to form a new node.

Algorithm 2 Insertion algorithm

Input: $flag = 0$: Record the depth of the tree, Point $E(P)$: the point with encryption, Node t: KD-tree with t as root node
Output: The KD-tree

```
function INSERT(flag = 0, Point E(P), Node t)
    if E(P) = t.data then
        return
    else if E(P)[flag] < t.data[flag] then
        if t.left = NULL then
            t.left = new NodeE(P)
        else
            t.left = INSERT(flag ⊕ 1, E(P), t.left)
        end if
    else
        if t.right = NULL then
            t.right = new NodeE(P)
        else
            t.right = INSERT(flag ⊕ 1, E(P), t.right)
        end if
    end if
    return t
end function
```

This algorithm involves the Ciphertext Comparison Protocol which is perform by LSP and CSP interactively, and the protocol is shown in Fig. 3.

Input two ciphertexts $[a]_{pk_1}$ and $[b]_{pk_2}$, LSP and CSP jointly perform the Ciphertext Comparison Protocol to obtain their corresponding plaintexts a and b in terms of their size relationship. The protocol is executed as follows.

Step1(@LSP): LSP selects a random integer $r \in [1, N/4]$ and computes $A = \left([a]_{pk_1}\right)^r = [a \cdot r]_{pk_1}$, $B = \left([b]_{pk_2}\right)^r = [b \cdot r]_{pk_2}$. Then he performs the partial decryption algorithm **PSDec1()** of DT-PKC over A and B respectively and obtains the results denoted as A' and B'.

Step2(@LSP): LSP randomly selects a bit $\mu \in \{0, 1\}$, and sends $\{A, A_1, B, B_1\}$ to CSP when $\mu = 1$; sends $\{B, B_1, A, A_1\}$ when $\mu = 0$.

Step3(@CSP): CSP performs **PSDec2()** of DT-PKC to decrypt $\{A, A_1, B, B_1\}$ or $\{B, B_1, A, A_1\}$ by the partial private key mk_2 to obtain the plaintext $\{ar, br\}$ or $\{br, ar\}$. Then he computes $d = (a - b) \cdot r$ by $\{ar, br\}$ or $d = (b - a)r$ by $\{br, ar\}$.

Step4(@CSP): When $d > 0$, CSP returns $\omega = 1$ to LSP; when $d < 0$, CSP returns $\omega = -1$; when $d = 0$, CSP returns $\omega = 0$.

Step5(@LSP): LSP acquires the comparison result between a and b based on μ and ω, where

$$a \begin{cases} > b & \mu = 1, \omega = 1 \text{ or } \mu = 0, \omega = -1 \\ < b & \mu = 1, \omega = -1 \text{ or } \mu = 0, \omega = 1 \\ = b & \mu = 1, \omega = 0 \text{ or } \mu = 0, \omega = 0. \end{cases} \tag{7}$$

ii) Deletion

Algorithm 3 describes how to delete expired nodes before the new nodes are inserted. Search from the root node of the tree. If the current node is the node to be deleted, check the left and right subtrees of the node. If the right subtree is not empty, the current node is replaced by the same dimension node that is the smallest in the right subtree. Then the replaced node becomes the new node to be deleted, which is recursively searched

Algorithm 3 Delete algorithm

Input: $flag = 0$: Record the depth of the tree, Point $E(P)$: the point with encryption, Node t: KD-tree with t as root node
Output: The KD-tree

```
function DELETE(flag = 0, Point E(P), Node t)
    temp = flag
    if E(P) = t.data then
        if t.right ≠ NULL then
            t.data = FINDMIN(flag ⊕ 1, t.right, temp)
            t.right = DELETE(flag ⊕ 1, t.data, t.right)
        else if t.left ≠ NULL then
            t.data = FINDMIN(flag ⊕ 1, t.left, temp)
            t.left = DELETE(flag ⊕ 1, t.data, t.left)
        else
            t = NULL
        end if
    else if E(p)[flag] ≤ t.data[flag] then
        t.left = DELETE(flag ⊕ 1, E(p), t.left)
    else
        t.right = DELETE(flag ⊕ 1, E(p), t.right)
    end if
    return t
end function
```

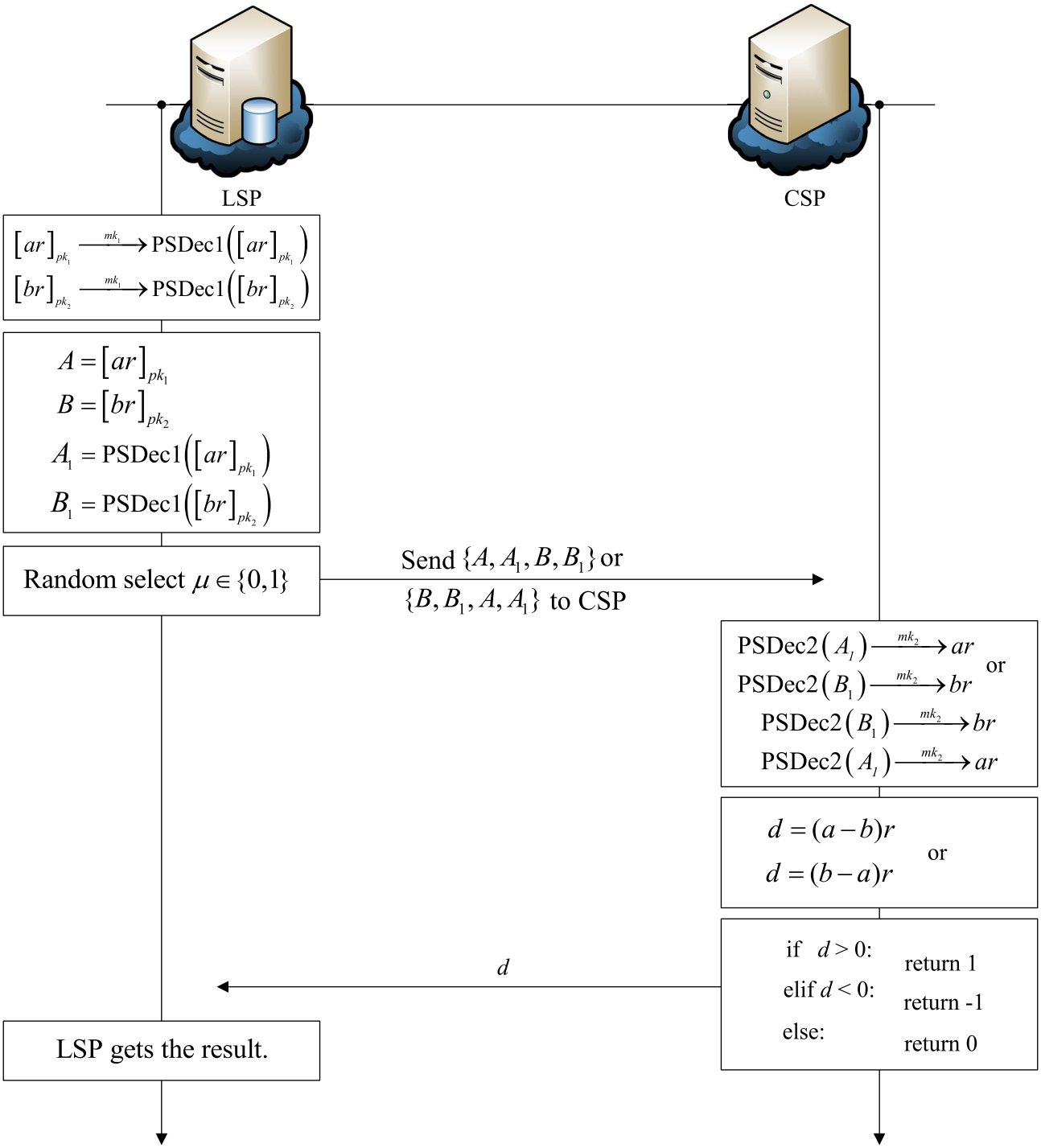


Fig. 3 Ciphertext Comparison Protocol

and deleted in the right subtree, likewise for the non-empty left subtree. If both the left and right subtrees are empty, it means that the current node is the node to be deleted, and the node is directly set to be empty.

Note that Algorithm 3 is also called a sub-algorithm to find the smallest point in the specified dimension in the subtree. Input a sub-tree to be searched and the partition dimension to be found, first determine whether the specified dimension and the current partition dimension of the tree are the same. If the dimensions are the same, then judge whether the left subtree of the current node is empty or not. If the left subtree is empty, return the

current node directly. If not, continue to search in the left subtree. If the dimensions are different, then search in both the left subtree and the right subtree. Finally, the minimum of the current node, the result of the left subtree and the result of the right subtree are returned, as shown in Algorithm 4.

Algorithm 4 Findmin algorithm

Input: $flag = 0$: Record the depth of the tree, Node t: KD-tree with t as root node, cd: The dimension of the current node
Output: The minimum tree node

```
function FINDMIN(flag, Node t, cd)
    if t = NULL then
        return NULL
    end if
    if flag = cd then
        if t.left = NULL then
            return t.data
        else
            return FINDMIN(flag ⊕ 1, t.left, cd)
        end if
    else
        return min(FINDMIN(flag ⊕ 1, t.left, cd),
FINDMIN(flag ⊕ 1, t.right, cd), t.data)
    end if
end function
```

4.4 *K*-NN query

Firstly, the query center encrypts the location information $p_0 = (x_0, y_0, u_0)$ of the point to be queried with DT-PKC under the public key pk_0, and then denotes the query location information vector p'_0 with the ciphertext and pk_0. After that, query center sends the query request $\{p'_0, k\}$ to LSP, where

$$p'_0 = \left(E_{pk_0}(p_0), pk_0\right) = \left(E_{pk_0}(x_0), E_{pk_0}(y_0), E_{pk_0}(u_0), pk_0\right). \tag{8}$$

After receiving the request, LSP executes the *K*-NN query shown in Algorithm 5 for the encrypted *K*D-tree.

Algorithm 5 *K*-NN algorithm

Input: $flag = 0$: Record the depth of the tree, Point $E(P_0)$: Encrypted data points to be queried, Node t: KD-tree with t as root node, PQ: The result queue, k: Parameters of the K-NN query and the maximum length of the queue
Output: The result queue PQ with encryption

```
function K-NN (flag = 0, Point E(P0), Node t, QueuePQ, k)
    if t = NULL then
        return PQ
    else if PQ.size < k or dist(E(P0), t.data) < bestdist then
        PQ.Enqueue(t.data)
        PQ.Order()
        bestdist = dist(PQ1, E(P0))
    end if
    if E(P0)[flag] < t.data[flag] then
        K − NN(flag ⊕ 1, E(P0), t.left, PQ)
    else
        K − NN(flag ⊕ 1, E(P0), t.right, PQ)
    end if
    if (E(P0)[flag] − t.data[flag])^2 < bestdist then
        if E(P0)[flag] ≤ t.data[flag] then
            K − NN(flag ⊕ 1, E(P0), t.right, PQ)
        else
            K − NN(flag ⊕ 1, E(P0), t.left, PQ)
        end if
    end if
end function
```

Input the encrypted node $E(p_0)$ to be queried and the empty query result queue PQ, where PQ is the queue structure that stores the position information as the returned result and is continuously updated during the recursive traversal of the *K*D-tree. When the result queue is not full, i.e., the number of elements in the queue is less than k or the Euclidean distance between the current node and the query point is less than *bestlist*, the current node will be queued and *bestlist* will be updated. Next, if the value of the node to be queried is less than the current node in the current partition dimension, it is recursively queried in the left subtree, and likewise for the right subtree.

One more case should also be noticed is that the nearest neighbor of the point to be queried may also exist in another subtree of the tree. Hence the same operation is required to be performed in another subtree as well. At the end of the recursive query, the elements in the returned result queue PQ are the query results sent by LSP to the query center.

Algorithm 5 requires the Ciphertext Comparison Protocol and the Ciphertext Euclidean Distance Calculation Protocol. The latter is shown in Fig. 4. The necessary descriptions are presented below.

i) For two coordinate points (x_1, y_1) and (x_2, y_2) in the two-dimensional plane, their Euclidean distance is

$$dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \tag{9}$$

Since the square of distance and the distance is equivalent when the former is larger than 0, it is enough to find the square of distance of the query point and each point for our *K*-NN query scheme.

In order to calculate the square of the difference between two numbers a and b, we resort to the following formula

$$(a - b)^2 = (a - b + R)^2 - 2R(a - b) - R^2, \tag{10}$$

ii) Input two ciphertexts $[a]_{pk_1}$ and $[b]_{pk_2}$, LSP and CSP jointly perform the Euclidean Distance Calculation Protocol for the ciphertext data to obtain the square of their corresponding plaintext difference. The protocol is performed as follows.

Step1(@LSP): LSP selects two unequal random numbers $r_1, r_2 \in \mathbb{Z}_N$ and calculates $R = r_1 - r_2$. Then he encrypts r_1, r_2, R, R^2 with the encryption algorithm of DT-PKC under pk_1, pk_2, PK respectively to get $[r_1]_{pk_1}$, $[r_2]_{pk_2}$, $[R]_{PK}$ and $[R^2]_{PK}$.

Step2(@LSP): LSP computes $A = [a]_{pk_1} \otimes [r_1]_{pk_1} = [a + r_1]_{pk_1}$ and $B = [b]_{pk_2} \otimes [r_2]_{pk_2} = [b + r_2]_{pk_2}$, where $\otimes$ represents the component-wise multiplication of DT-PKC. Then he performs PSDec1() of DT-PKC over A

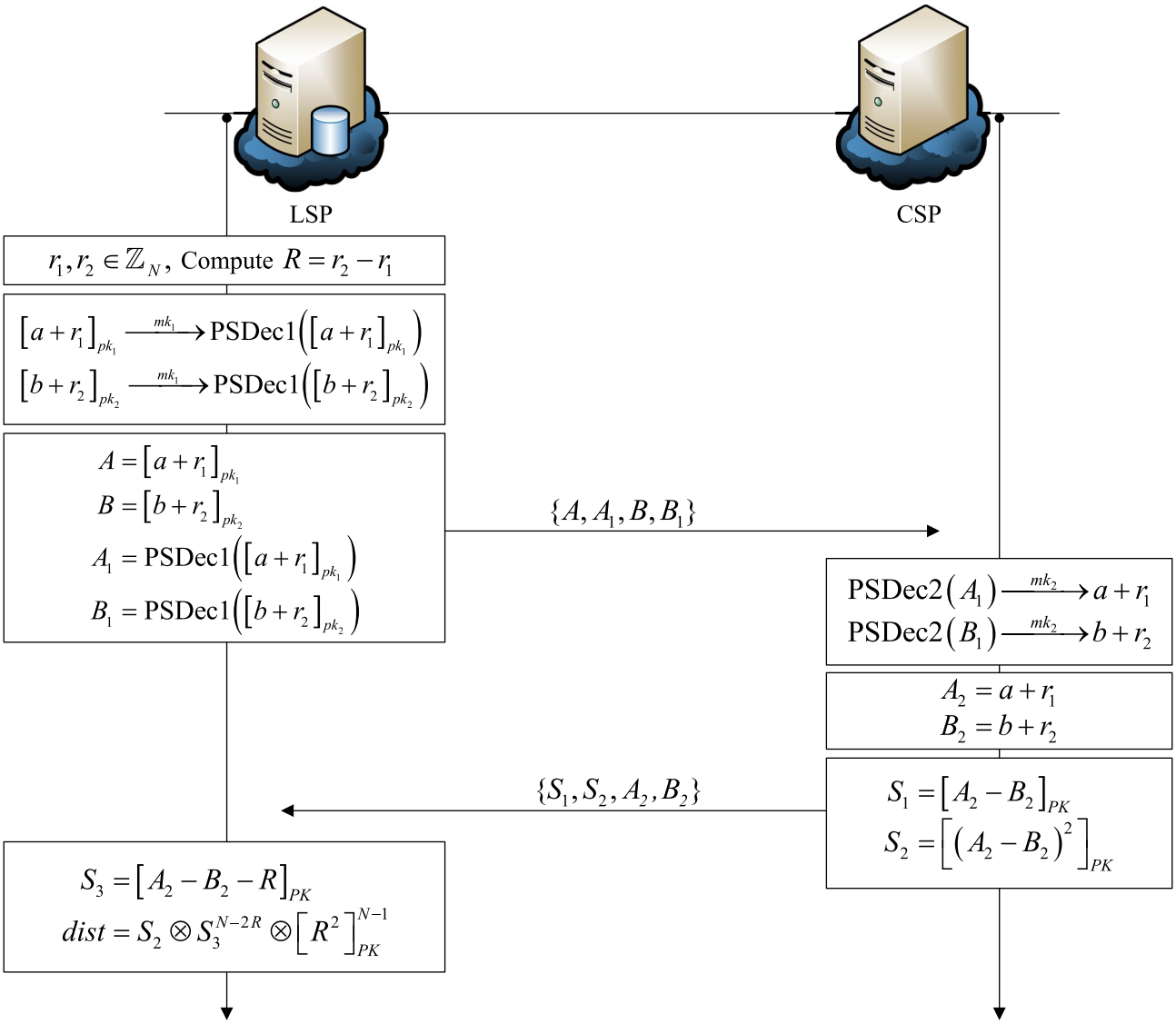


Fig. 4 Euclidean Distance Calculation Protocol

and B, gets the results A_1 and B_1, and sends $\{A, A_1, B, B_1\}$ to CSP.

Step3(@CSP): CSP performs PSDec2() over A_1 and B_1 to obtain $A_2 = a + r_1$ and $B_2 = b + r_2$ with mk_2. Then he encrypts $A_2 - B_2$ and $(A_2 - B_2)^2$ with PK to get $S_1 = [A_2 - B_2]_{PK}$ and $S_2 = [(A_2 - B_2)^2]_{PK}$, and sends $\{S_1, S_2, A_2, B_2\}$ back to LSP.

Step4(@LSP): After receiving $\{S_1, S_2, A_2, B_2\}$ from CSP, LSP calculates $S_3 = S_1 \otimes ([R]_{PK})^{N-1}$ and finally gets $dist = S_2 \otimes S_3^{N-2R} \otimes [R^2]_{PK}^{N-1}$.

Finally, the query center decrypts k elements in the queue PQ with the master private key mk. And he finally obtains the location information of k nearest neighbors points of the location information to be queried. The correctness of this algorithm follows from the following equation, which depends on the homomorphism of DT-PKC.

$$\begin{aligned} dist =& S_2 \otimes S_3^{N-2R} \otimes [R^2]_{PK}^{N-1} \\ =& [(A_2 - B_2)^2 + (A_2 - B_2 - R)^{N-2R} - R^2]_{PK} \\ =& [(a - b + R)^2 + (a - b)^{N-2R} - R^2]_{PK} \\ =& [(a - b + R)^2 - 2R(a - b) - R^2]_{PK} \\ =& [(a - b)^2]_{PK}. \end{aligned} \tag{11}$$

Therefore, the above K-NN query scheme holds.

5 Security analysis

In this section, we analyse the security of the proposed *K*-NN query scheme.

Our prior concern is the privacy of the scheme, i.e., the location data stored in LSP, the queried records and queried results of the query center. Recall that the location data the queried records and queried results are all encrypted with DT-PKC, which is secure based on the assumed intractability of the decisional Diffie-Hellman assumption over $\mathbb{Z}_{N^2}$. Then the remain is to prove that the Ciphertext Comparison Protocol and Ciphertext Euclidean Distance Calculation Protocol leak no information of the location data the queried records or queried results.

Lemma 1 *The Ciphertext Comparison Protocol in Fig. 3 is secure to protect the information of the plaintexts and the secret keys of LSP, CSP.*

Proof The information of the plaintexts, including the corresponding plaintexts, is protected effectively since what always run through the Ciphertext Comparison Protocol are ciphertexts. Now let us first prove the zero-knowledge for LSP, implying that no information of LSP will be exposed. We describe a simulator $\mathcal{S}im_{LSP}$ playing the role of LSP in front of any PPT (probability polynomial time) adversary $\mathcal{B}$ playing the role of CSP.

1. $\mathcal{S}im_{LSP}$ takes input public key, outputs $mk_2^{'}$ as the secret key instead of mk_2 and feeds the adversary with it. Then the simulator $\mathcal{S}im_{LSP}$ performs the Ciphertext Comparison Protocol as LSP with $-mk_2^{'}$ instead of mk_1.
2. After the adversary $\mathcal{B}$ obtains $\{A, A_1, B, B_1\}$ or $\{B, B_1, A, A_1\}$ from $\mathcal{S}im_{LSP}$ dependent on the random value of μ, where $A = [ar]_{pk_1}, B = [br]_{pk_2}$, $A_1 = \text{PDec1}([ar]_{pk_1}, -mk_2^{'}), B_1 = \text{PDec1}([br]_{pk_2}, -mk_2^{'})$ he will compute

   $$\text{PDec2}(A_1, mk_2^{'}), \text{PDec2}(B_1, mk_2^{'}).$$

The correctness follows from $mk_1 + mk_2 \equiv 0 (\text{mod } \lambda)$. From the view of $\mathcal{B}$ playing the role of CSP, the behavior of $\mathcal{S}im_{LSP}$ and LSP is indistinguishable due to the indistinguishability of mk_1 and $-mk_2^{'}$. Hence the advantage for adversary $\mathcal{B}$ to distinguish $\mathcal{S}im_{LSP}$ and LSP is negligible. That is, it is zero-knowledge for LSP.

There is no need to illustrate the zero-knowledge of semi-honest CSP, as CSP just performs some auxiliary computations instead of obtaing the final results. Consequently, we finish the proof of the lemma. □

Lemma 2 *The Euclidean Distance Calculation Protocol in Fig. 4 is secure to protect the information of the plaintexts and the secret keys of LSP, CSP.*

Proof Here we omit the detailed proof, since it is similar with the simulated-based Lemma 1 except the executions of LSP and CSP for different functionality. One more important is the assumption of no collusion between LSP and CSP in the system model. □

To sum up, our *K*-NN query scheme applies the DT-PKC, Ciphertext Comparison Protocol, Ciphertext Euclidean Distance Calculation Protocol and *K*D-tree. Notice all the tools leak no information of the location points, queried requests and queried results, we can conclude our *K*-NN query scheme is privacy-preserving.

6 Preformance evaluation

This section includes theoretical and experimental analyses to illustrate the performance of our *K*-NN query scheme.

6.1 Theoretical analysis

In this section, we perform a theoretical comparison analysis of some *K*-NN query schemes, from the perspective of data structure, multi-user support, support data update and security against server, as shown in Table 2.

It can be seen that, except [23], the other three schemes are resistant to server attacks because they store the location points in ciphertext on the server. The scheme proposed by Zhang et al. [10] supports multi-user environment while no update of location points. The update of location points in [22] holds with a heavy overhead. One is the online interaction between the data owner and the query center, the other is that the data owner needs to reset the Moore curve for all location points even for an updated location point. Relatively speaking, our *K*-NN query scheme is more practical, which supports not only data security updates but also multi-user environments. Meanwhile, the data owner can go offline after uploading the data, reducing the cost of the data owner. The update of location points takes place in part nodes of *K*D-tree, not the whole *K*D-tree.

6.2 Experimental analysis

In this section we perform performance evaluation of the proposed *K*-NN query scheme. The scheme is tested on an Inter Core I5-9300H 2.40GHz CPU and 4GBRAM on a windows 10 computer running an Ubuntu18.04.1 LTS virtual machine via python charm package and the length of the public modulus *N* is 1024 bits.

We randomly generate 1000, 10000 and 100000 coordinate points in the plane area as the test set to test the performance of this scheme under different scale of data. The

Table 2 Comparison of schemes

Scheme	data structure	Multi-user support	Support data update	Security against server
Utsunomiya et al. [23]	Moore curve	×	×	×
Zhang et al. [10]	Linear quad-tree	√	×	√
Lian et al. [22]	Moore curve	×	√	√
Our scheme	*K*D-tree	√	√	√

final results are averaged for each scale of data set for several experiments.

i) Query Accuracy

Let R and G denote the set of K-nearest neighbor results returned by the experiment and the true set of K-nearest neighbor results, respectively, then the query accuracy can be expressed as

$$r = \frac{|R \cap G|}{|G|}. \tag{12}$$

As shown in Fig. 5, a line graph of query accuracy with query parameter k varying is shown.

We can see that when using our scheme for K-nearest neighbor query, the query accuracy can be maintained above 0.95 for all three data sets. For a fixed number of location points n, the query accuracy decreases with an increasing k. Meanwhile, for a fixed k, the query accuracy is larger for a larger n.

ii) Time Cost

The time cost for query center mainly depends on constructing the *K*D-tree in the initialization phase, encrypting the tree nodes and uploading them. Even these operations can be performed in the phase of a pretreatment, query center also need to spend time to deal with them. As shown in Fig. 6, we show the time cost comparison of running the initialization algorithm to construct a *K*D-tree for three datasets over plaintexts and ciphertexts respectively. It can be seen that the time increases as the number of dataset increases. Under the same condition, the time in Fig. 6b is much larger than the one in Fig. 6a, since the Ciphertext Comparison Protocol for creating *K*D-tree occupy much time. For a dataset containing 100000 location information points, it takes about 600ms to construct a plaintext *K*D-tree, while it takes nearly half an hour to construct an encrypted *K*D-tree.

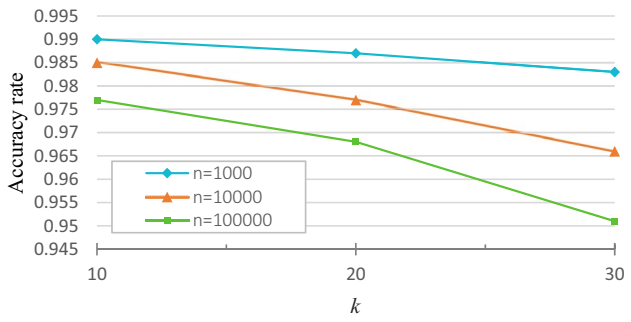


Fig. 5 Variation of query accuracy with k

As shown in Fig. 7, the time to insert or delete encrypted nodes to generate an update *K*D-tree under different data sets is presented varying from the number of initial nodes and updatable nodes. Fig. 7a is for insertion 1 or 10 new nodes with the total number of initial nodes varying 1000, 10000, 100000, and Fig. 7b is for deletion under the same circumstance. The update of the encrypted *K*D-tree is mainly performed by LSP and CSP via Ciphertext Comparison Protocol and Ciphertext Euclidean Distance Calculation Protocol. Take 100000 location points as an example, we note that the average time to insert an encrypted node is 4ms, while the average time to delete a ciphertext node is 96ms.

The reason for such a large difference in time is that when deleting a node, there are more operations. Concretely, LSP and CSP performs not only the node deletion algorithm, but also the sub-algorithm to find the smallest node in the current dimension. In order to improve the efficiency of deletion in the application of practical scenarios, we can consider the marker deletion method. That is, the dense data nodes do not need to be deleted immediately, but are marked to remain in the *K*D-tree. These marked points will not affect the insertion of subsequent nodes and marker deletion. When the number of marked points exceeds the threshold value, LSP can delete them voluntarily, which is flexibly decided by LSP according to the size of the tree and his own computational capacity.

Figure 8 shows the total time of K-NN query with varying datasets and query parameters without considering the pretreatment phase. We can see that the average time is increasing with the number of new location points k for a fixed number of initial location points n. With respect to a fixed k, the average time is increasing slowly with varying n. Even when $n = 100000$ and $k = 30$, the average time for our K-NN query scheme is only less than 3s.

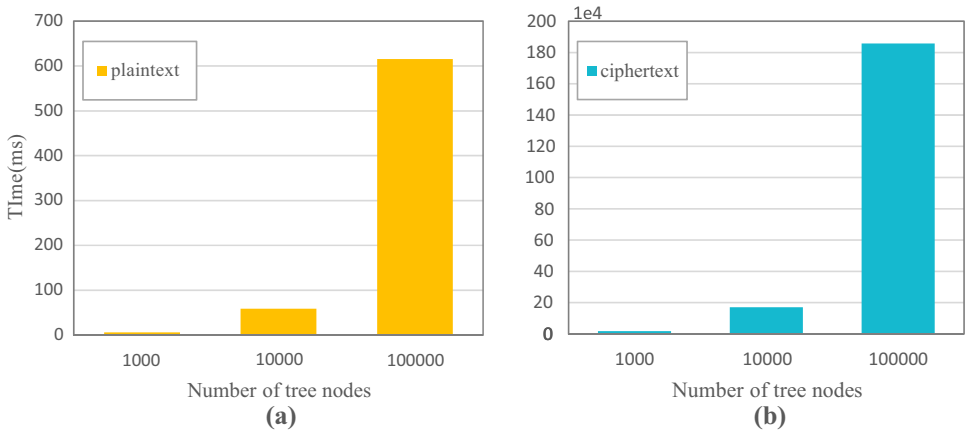


Fig. 6 Time cost for creating *K*D-trees

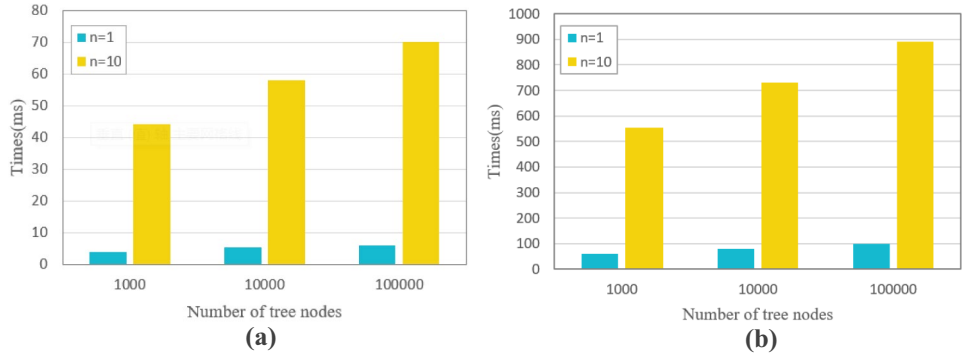


Fig. 7 Time cost for insertion or deletion *K*D-tree nodes

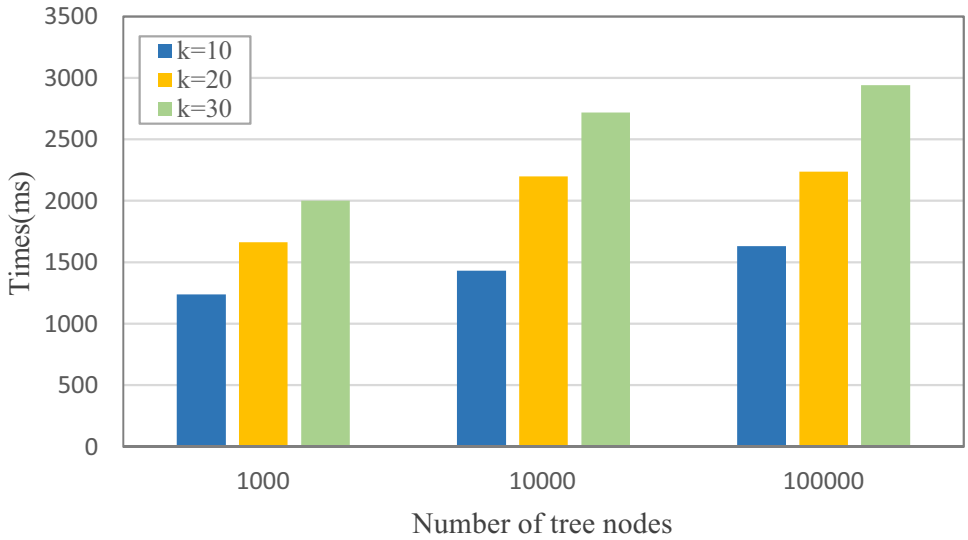


Fig. 8 Time cost for *K*-NN query

7 Conclusion

In this paper, we focus on *K*-NN query which supports updatable location points without compromising the privacy of the data owners and query center. At the core of our scheme, every data owner encrypts his location points with DT-PKC cryptosystem under his own public key. Two novel secure protocols Ciphertext Comparison Protocol and Ciphertext Euclidean Distance Calculation Protocol under the double-cloud model provide great help for *K*-NN query based on *K*D-tree of dimension 2. Theoretical analyses show that our scheme can protect the confidentiality of location data, queried requests and queried results. Meanwhile, experimental evaluations demonstrate the higher efficiency of our scheme.

Acknowledgements This research was supported by the National Natural Science Foundation of China under Grant Nos. U19B2021, 61972457, and Key Research and Development Program of Shaanxi under Grant No. 2020ZDLGY08-04.

Declarations

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

References

1. Matsuo Y, Okazaki N, Izumi K, Nakamura Y, Nishimura T, Hasida K, Nakashima H (2007) Inferring long-term user properties based on users' location history. In: IJCAI, pp 2159–2165
2. Gambs S, Killijian MO, del Prado Cortez MN (2010) Show me how you move and i will tell you who you are. In: Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS, pp 34–41
3. Krumm J (2007) Inference attacks on location tracks. In: International Conference on Pervasive Computing. Springer, pp 127–143
4. Beresford AR, Stajano F (2003) Location privacy in pervasive computing. IEEE Pervasive Comput 2(1):46–55
5. Das AK, Tabassum A, Sadaf S, Sinha D (2020) Attack prevention scheme for privacy preservation (apsp) using k anonymity in location based services for iot. In: Computational Intelligence in Pattern Recognition. Springer, pp 267–277
6. Gedik B, Liu L (2007) Protecting location privacy with personalized k-anonymity: Architecture and algorithms. IEEE Trans Mob Comput 7(1):1–18
7. Wang J, Li Y, Yang D, Gao H, Luo G, Li J (2017) Achieving effective *k*-anonymity for query privacy in location-based services. IEEE Access 5:24580–24592
8. Indyk P, Woodruff D (2006) Polylogarithmic private approximations and efficient matching. In: Theory of Cryptography Conference. Springer, pp 245–264
9. Yiu ML, Jensen CS, Huang X, Lu H (2008a) Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In: 2008 IEEE 24th International Conference on Data Engineering. IEEE, pp 366–375
10. Zhang S, Yao T, Liang W, Sandor VKA, Li KC (2020) An efficient privacy-preserving multi-keyword query scheme in location based services. IEEE Access 8:154036–154049
11. Zhu H, Lu R, Huang C, Chen L, Li H (2015) An efficient privacy-preserving location-based services query scheme in outsourced cloud. IEEE Trans Veh Technol 65(9):7729–7739
12. Zhu Y, Huang Z, Takagi T (2016) Secure and controllable k-nn query over encrypted cloud data with key confidentiality. J Parallel Distrib Comput 89:1–12
13. Sweeney L (2002) k-anonymity: A model for protecting privacy. Int J Uncertainty Fuzziness Knowledge Based Syst 10(05):557–570
14. Gruteser M, Grunwald D (2003) Anonymous usage of location-based services through spatial and temporal cloaking. In: Proceedings of the 1st International Conference on Mobile Systems, Applications and Services, pp 31–42
15. Mokbel MF, Chow CY, Aref WG (2006) The new casper: Query processing for location services without compromising privacy. In: Proceedings of the 32nd International Conference on Very Large Data Bases, pp 763–774
16. Bamba B, Liu L, Pesti P, Wang T (2008) Supporting anonymous location queries in mobile environments with privacygrid. In: Proceedings of the 17th international conference on World Wide Web, pp 237–246
17. Khoshgozaran A, Shahabi C (2007) Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In: International symposium on spatial and temporal databases. Springer, pp 239–257
18. Lien IT, Lin YH, Shieh JR, Wu JL (2013) A novel privacy preserving location-based service protocol with secret circular shift for k-nn search. IEEE Trans Inf Forensics Secur 8(6):863–873
19. Kapadia A, Triandopoulos N, Cornelius C, Peebles D, Kotz D (2008) Anonysense: Opportunistic and privacy-preserving context collection. In: International Conference on Pervasive Computing. Springer, pp 280–297
20. Yiu ML, Jensen CS, Huang X, Lu H (2008b) Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In: 2008 IEEE 24th International Conference on Data Engineering. IEEE, pp 366–375
21. Gong Z, Sun GZ, Xie X (2010) Protecting privacy in location-based services using k-anonymity without cloaked region. In: 2010 Eleventh International Conference on Mobile Data Management. IEEE, pp 366–371
22. Lian H, Qiu W, Yan D, Huang Z, Tang P (2020) Efficient and secure k-nearest neighbor query on outsourced data. Peer Peer Netw Appl 13(6):2324–2333
23. Utsunomiya Y, Toyoda K, Sasase I (2016) Lpcqp: Lightweight private circular query protocol with divided poi-table and somewhat homomorphic encryption for privacy-preserving k-nn search. J Inf Process 24(1):109–122
24. Liu X, Deng RH, Choo KKR, Weng J (2016) An efficient privacy-preserving outsourced calculation toolkit with multiple keys. IEEE Trans Inf Forensics Secur 11(11):2401–2414
25. Bresson E, Catalano D, Pointcheval D (2003) A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer, pp 37–54

Songyang Wu receives his master degree in School of Telecommunications Engineering, Xidian University. His main research interests include privacy-protection, homomorphic encryption.

Wenju Xu received the B.S. and M.S. degree in mathematics from Henan Normal University in 2014, 2017 respectively. She is currently pursuing the Ph.D.'s degree in Xidian University. Her main research interests include public key cryptography and fully homomorphic encryption.

Zhiyong Hong received the B.S. and M.S. degrees in computer science and technology from the Shenyang Institute of Technology, Shenyang, China, in 2001 and 2004, respectively, and the Ph.D. degree in computer science and technology from Southwest Jiaotong University, Chengdu, China, in 2014. From 2006 to 2014, he was a Lecturer with the School of Computer, Wuyi University, Jiangmen, China. Since 2015, he has been an Associate Professor. His research interests include intelligent information processing, block chain, and rough set theory.

Pu Duan has been a twenty-year veteran on cryptography, information security and networking security. He has published 20+ papers on areas of cryptography, networking security and system security. He joined Cisco after obtaining his Ph.D. degree from Texas A&M University in 2011. At Cisco he led the research and development of new cryptographic algorithms for TLS.13 on cisco firewall product. Currently Dr. Duan works in Secure Collaborative Lab (SCI) lab at Ant Group as a senior staff engineer, leading the team on research and implementation of privacy preserving technologies.

Benyu Zhang has been engaged in AI research and development for two decades. He published 49 peer reviewed papers with more than 11,000 citations, has 70 US patents approved and 84 US patents pending. Since he went into industry, he has Initiated and led numbers of core AI systems R&D in advertising, search, and recommendation at Google and FB. Currently he leads the Secure Collaborative Intelligence Lab (SCI Lab) at Ant Group to build the next generation privacy-preserving data mining and AI platform. The vision is to form the "data internet" and enable data mining and AI applications on data from multiple parties securely, efficiently, and effectively.

Yupu Hu received the M.S. degree in mathematics and the Ph.D. degree in cryptology from Xidian University, Xi'an, China, in 1987 and 1999, respectively, where he is currently a Professor with the Telecommunication College. He is also serving as one of the directors of the Chinese Association for Cryptologic Research. His major research interests include cryptology, including stream ciphers, block ciphers, and public key ciphers.

Baocang Wang is a professor in the School of Telecommunications Engineering, Xidian University. He received his Ph.D. degree in cryptography from Xidian University in 2006, and received his MS and BS degrees in mathematics from Xidian University in 2004 and 2001, respectively. His main research interests include public key cryptography, encryption data processing, data mining security.