computers

MDPI

Article

A Privacy Preserving Cloud-Based K-NN Search Scheme with Lightweight User Loads

Yeong-Cherng Hsu [1], Chih-Hsin Hsueh [2,*] and Ja-Ling Wu [3]

[1] MediaTek Inc., Hsinchu 30078, Taiwan; hyc1227@gmail.com
[2] Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei 10617, Taiwan
[3] Department of Computer Science and Information Engineering, National Taiwan University, Taipei 10617, Taiwan; wjl@cmlab.csie.ntu.edu.tw
* Correspondence: d05944010@ntu.edu.tw

Received: 4 November 2019; Accepted: 29 December 2019; Published: 1 January 2020

Abstract: With the growing popularity of cloud computing, it is convenient for data owners to outsource their data to a cloud server. By utilizing the massive storage and computational resources in cloud, data owners can also provide a platform for users to make query requests. However, due to the privacy concerns, sensitive data should be encrypted before outsourcing. In this work, a novel privacy preserving K-nearest neighbor (K-NN) search scheme over the encrypted outsourced cloud dataset is proposed. The problem is about letting the cloud server find K nearest points with respect to an encrypted query on the encrypted dataset, which was outsourced by data owners, and return the searched results to the querying user. Comparing with other existing methods, our approach leverages the resources of the cloud more by shifting most of the required computational loads, from data owners and query users, to the cloud server. In addition, there is no need for data owners to share their secret key with others. In a nutshell, in the proposed scheme, data points and user queries are encrypted attribute-wise and the entire search algorithm is performed in the encrypted domain; therefore, our approach not only preserves the data privacy and query privacy but also hides the data access pattern from the cloud server. Moreover, by using a tree structure, the proposed scheme could accomplish query requests in sub-liner time, according to our performance analysis. Finally, experimental results demonstrate the practicability and the efficiency of our method.

Keywords: K-NN; privacy preserving; cloud; encryption; security

1. Introduction

By outsourcing data and/or tasks to the cloud, even devices with low computational ability can conduct analytic works with a large amount of data. Otherwise, data owners need to build a data warehouse for hosting their data so that other users can make queries on it, for further researches or services. Cloud server does benefit to the above application scenarios; however, it is well-known that security issues are arisen if sensitive data are involved. For example, the medical records have not yet been stored on the cloud platform because the records may contain sensitive information which needs specific privacy protection before being released to public. A compromised cloud server might expose the medical dataset outsourced from medical data owners or infringe upon patients' privacy by leaking out the associated symptoms or diagnosis results.

For the protection of sensitive data, data owners usually encrypt their data before outsourcing them to the cloud. Moreover, to make sure none of the others can retrieve the private data, no one except the data owner themselves can access the decryption key. At the same time, other users who want to access the data can make query requests to the cloud. Interestingly, sometimes these users also want to preserve their query privacy from data owners and the cloud server. Therefore, the query

would be encrypted as well. The above-mentioned security issues can be solved if we can process the queries directly over the encrypted dataset and make sure that the whole scheme works as a searchable privacy preserving data storage system.

In this work, we focus on solving the K-nearest neighbor (K-NN) search problem over the encrypted dataset in the cloud. The K-NN search is a basic method widely used for classification and regression in pattern recognition or data mining areas. The K-NN search algorithm takes a dataset and a query as input and outputs K data points closest to the query. Both data points in the dataset and the query can be multi-dimensional data and the measurement of the distance between them is the norm defined in a specific space, usually the Euclidean space. In the proposed scheme, data points are encrypted attribute-wise and so is the query. The entire search algorithm is performed in the encrypted domain and the user obtains the unencrypted results relative to his query. Apparently, the user has to be trusted by data owners since they can obtain a small portion of the plaintext data, eventually.

There are several works proposed to solve the pre-described privacy-preserving K-NN (PPkNN) search problem, but all of them have certain limitations, such as the need of sharing the data owner's decryption key to others, adding additional storage to the user's end, using two non-colluding servers, and refining the resultant data by the query user, etc. In this work, we try to overcome those limitations and find a new approach which meets the needs of privacy preserving and searchability simultaneously. Moreover, unlike most of existing methods that search the K-NN results linearly through the entire dataset, our approach leverages a tree structure, named R-tree [1], to store the data, making the search complexity faster than the linear ones. In short, the contributions of our work can be summarized as follows:

a. We build a new encryption scheme by combining the ideal secure order-preserving encryption [2] with the well-known Paillier cryptosystem [3] and show that the resultant scheme is still ideally secure.
b. We propose a PPkNN search scheme based on the new encryption method described above, which can perform comparison and addition operations in the encrypted domain, and use an R-tree structure to achieve sub-linear search complexity.
c. Benefiting from the new encryption method, the construction of an R-tree can also be done by the cloud server on the encrypted domain, which can reduce the computational overhead of data owners and enhance the query user's security by preventing data owners from knowing the data access pattern.
d. We address and overcome the limitations of related works and make a comparison of our work with them.
e. Experimental results over synthetic datasets show that our method is practical and efficient. Moreover, the workloads on user ends are very lightweight.

The rest of this paper is organized as follows. In Section 2, we survey some related works and discuss their limitations. The preliminary backgrounds to understand our scheme are introduced in Section 3 and the proposed work is addressed in Section 4. The security guarantees, such as the ideal security definition, the data privacy, and the query privacy, are analyzed in Section 5. In Section 6, we evaluate the performances of our work by complexity analyses and demonstrate them by the experimental results. Finally, Section 7 concludes this work.

2. Related Works

2.1. A Brief Survey of the Access Control Mechanisms

As mentioned by one of the anonymized reviewers, cloud computing provides the on demand and scalable services, therefore, the corresponding environment is highly dynamic. In addition to seeking for the assistance of cryptographic techniques, applying the access control mechanism is another fundamental and important choice that can meet the security requirements in the cloud environment.

The main goal of applying access control mechanism is to restrict the users from performing any unauthorized activity to protect the sensitive information.

As summarized in [4], there are wide variety of models, methods, and policies proposed for designing an access control system. Each access control system has its own attributes, methods, and functions based on set of policies. For example, the Mandatory Access Control model (MAC) [5] is an access control policy in which a subject or request initiator can perform some sort of operation on a particular object or resource. When a subject or user attempts to access an object or resources, an authorization rule is enforced to determine whether the access can take place by examining the security attributes. Discretionary Access Control model (DAC) [6] is another well-known access control policy which determines the owner of an object. The owner decides who is allowed to access the service based on users' identities. Some researchers analyze the dynamic requirements for cloud environment and introduce the Role Based Access Control model (RBAC) [7] to the cloud environment. RBAC is an access control policy determined by the systems rather than by the owner itself. RBAC model can only be applied within a closed network and it is based on identification. It only checks the user identity and roles assigned to users and based on this role it checks if the user is authorized or not.

However, as commented also in [4], both MAC and DAC can only be applied to specific environments like operating and database management systems, while RBAC model fails to check the malicious activity done by the authorized users. In facing of the above-mentioned shortages, the trust-based mechanisms (TBMs) [8] and the context-aware RBAC (CA-RBAC) [9–12], which well fit today's dynamic environments provided by cloud and fog computing, are arisen. Both TBMs and CA-RBAC do effectively secure the sensitive information by authorizing legal users and protecting the cloud resources from the malicious activities. However, the focus of this work is to find cryptography-based security solution for Cloud-based K-NN search problem, and a complete survey of TBMs and AC-RBAC cannot be provided within such a limited space. Interested readers can find the corresponding details in the listed references and the references therein.

2.2. A General Survey of Privacy-Preserving K-NN Problem

As mentioned in Section 1, many works have been proposed to solve the secure K-NN search problem over the encrypted outsourced dataset in the cloud. In [13], a PPkNN approach was proposed to provide the privacy of dataset, input query, kNN result, and data access pattern, but it is vulnerable to collusion attacks. In other words, it assumed that there is neither collusion among cloud servers nor collusion between any data owner and cloud server. In [14], a privacy preserving medical diagnosis system using E-health was proposed, in which the cloud server worked in a privacy preserving manner even though the medical datasets are owned by multiple data owners. As a building block of the proposed diagnosis system, the authors designed a deterministic K-NN based privacy preserving protocol for finding the k data with the highest similarity to a queried symptom, which reduces the average running time by 35% compared to that of a previous probabilistic-behaved work [15]. Basically, this PPkNN protocol is constructed on the basis of multiparty computation (MPC) based on secret sharing, and therefore works in a distributed manner without any trusted server. A survey about existing works related to PPkNN can be found in [14]. Similarly, in [16], the authors focused on solving the clustering problem over encrypted cloud data. In particular, they proposed a privacy-preserving k-means clustering technology over encrypted multi-dimensional cloud data by leveraging the scalar-product-preserving encryption primitive, called PPK-means. The proposed technique is able to achieve efficient multi-dimensional data clustering as well as preserve the confidentiality of the outsourced cloud data. The authors claimed that their work is the first one to explore the privacy-preserving multi-dimensional data clustering in the cloud computing environment. Extensive experiments in simulation data-sets and real-life data-sets demonstrate that the proposed PPK-means is secure, efficient, and practical.

2.3. A Survey of the Most Related Works

In this subsection, we briefly describe the most related works and their corresponding characteristics such as the scheme structure, the main idea of searchable encryption, the security definition and the corresponding performances.

Wong et al. [17] presented an asymmetric scalar-product preserving encryption (ASPE) scheme to perform the secure K-NN computation on encrypted databases. The word asymmetric indicates the fact that the data owner and the query user perform their encryption in different ways. The scalar-product between encrypted query and data points is preserved and the encryption equations built by ASPE allow the user to find out the K closest points among all the encrypted data. The data owner is forced to share the decryption key with users so that they can recover the encrypted results after receiving them. We consider that as a risk of revealing more unexpected private information to users. In addition, the method simply uses linear scan to find the results, but the requirement of complexity we consider is a sub-linear one.

Hu et al. [18] discussed the general problem of query processing over untrusted data cloud including K-NN query and range query. Unlike most of the other methods which simply outsourced the data to cloud server, they restructured their data based on an R-tree and sent the associated indexes to users. The query processing procedure is executed by the cloud server, who has the decryption key, and the user. They leverage a privacy homomorphism encryption method to make the secure version of K-NN best first search (BFS) algorithm over R-tree executable. However, most of the computational cost associated with the processing procedure is endured at the user end which is against the original intention of using the cloud. Moreover, the indexes of R-tree need to be stored which consumes additional memories from the user. Furthermore, the scheme was proved not secure under probing attack [19]. After sending an adequate number of query requests, the user is able to recover the plaintext owned by the data owner.

Some security issues of [17,18] have been reported by Yao et al. in their work [19]. They revisited the above two methods and proved that they are insecure under chosen plaintext attack by introducing a new attack model. Another secure nearest neighbor finding scheme using a secure Voronoi Diagram (SVD) was presented in [19]. The SVD scheme induces a partition of the dataset and the data owner encrypts those partitions and sends them to the server with their identifiers. The query user finds the nearest neighbor of his query by asking server to return the partitions relative to the results. However, the above-mentioned limitations such as the need of sharing decryption key with users and the additional storage consumption (to store the descriptions of partitioned data) still exist in this scheme. Users also need to provide extra computational resources to retrieve the accurate results among the returned partitions. Worst of all, the construction of SVD on data with higher than two dimensions needs large computational cost.

Considering the security level more strictly, Elmehdwi et al. [20] proposed a fully secure K-NN search protocol over encrypted datasets in outsourced environment. They introduced several basic security protocols, e.g., secure multiplication, secure squared Euclidean distance, secure minimum, and secure bit-or with Paillier cryptosystem to build the whole secure scheme, which not only preserve the data privacy and query privacy but also hide the data access pattern from the cloud server. Unfortunately, the protocols are based on the setup of two non-colluding cloud servers and the corresponding complicated protocols between the two servers leads to a long query response time. Those characteristics are regard as the disadvantages of this fully secure scheme which may handicap its value in practical usage. Our approach tries to relax the security guarantee of this approach by preventing the data access pattern revealed to server into considerations so as to enhance the practicability.

Wang et al. [21] proposed a practical method using the ideal secure order preserving encryption and the R-tree structure to solve the similar problem on large-scale data. In their work, the search procedure includes two interaction communications between the user and the server. In the first round of interaction, the server narrows down the candidate results by identifying which minimum bounding

rectangle (MBR) contains the query and sends the points within MBR back to the user. Then the user creates a search box according to the nearest point in the returned set and sends it to the server, and the server outputs all points in that box as the resulting set. One can see that this scheme, the same as [19], does not return accurate results, and extra distance calculations must be operated by the user. The problem structure of this work is also different from the ordinary ones, for example, in [21] a user plays the roles of both the data owner and the query user. That is, if we want to allow a third-party user to make a query request, the data owner needs to share out the decryption key, after all.

In addition, a few new works have been proposed to solve the same problem, but most of them did not meet our needs as well. Those works either need two non-colluding servers as model-assumption [22] or only return the indexes of search results [23,24]; moreover, decryption key shared from the data owner is a must, if the user wants to obtain the unencrypted results.

In summary, plenty of privacy preserving K-NN search schemes have been proposed but all of them have certain shortages and put up barriers on their usage in practice. On the other hand, our approach achieves the relaxed security level, which preserves the data privacy of the data owner and the query privacy of the query user but reveals the data access pattern to the cloud server. Notice that the leakage of data access pattern is not an issue when the stored data has been encrypted. In other words, without the decryption key, no damage occurs even if the server knows what ciphertext the user has accessed. More importantly, the limitations of the other works have been released somewhat by our scheme. We summarize the characteristics of the proposed scheme as follows and also show the comparisons with some related works in Table 1:

a. The search complexity of our scheme is faster than linear.
b. The data owner does not have to share the decryption key to the cloud server or other query users.
c. Our approach does not need extra local storage at users' end and consumes extremely small computational resources.
d. The cloud server returns neither the approximate results nor the indexes of resultant data points, but the accurate K-NN values.
e. Not only suitable for two dimensional spatial tuples, our approach can be extended to high dimensional dataset, directly.
f. Comparing with some methods in which two non-colluding servers are required, one honest-but-curious server is good enough to our scheme.

Table 1. The comparison between related works and the proposed scheme.

	[13]	[14]	[15]	[16]	[17]	Our Scheme
Sub-linear search complexity	×	✓	✓	×	✓	✓
No need to share private key with other	×	×	×	×	×	✓
No need to store local data at user end	✓	×	×	✓	✓	✓
Return only accurate K-NN set	✓	✓	×	✓	×	✓
Applicable to high dimensional data	✓	✓	×	✓	✓	✓
Single Sever	✓	✓	✓	×	✓	✓

3. Preliminaries

For the ease of explanation, the three most important components of the proposed scheme: Paillier Cryptosystem, Order-preserving Encryption, and R-tree will be briefly reviewed in this section.

3.1. Paillier Cryptosystem

Paillier cryptosystem is a probabilistic public key cryptographic system proposed by Pascal Paillier in 1999 [3]. Since it is a secure and efficient cryptography along with the additive homomorphic property, it has been utilized in many existing applications such as E-voting. The Paillier encryption scheme, denoted as $(\text{KeyGen}, \text{Enc}, \text{Dec})_{paillier}$, is defined as follows:

a. $\text{KeyGen}_{paillier}$

The key generation procedure is responsible for generating a pair of keys $(pk,\ sk)$. Public key pk is for data encryption and available to all public users in the system. Secret key sk is for decryption and only the privilege member, e.g., official manager has the right to access it. A simplified variant of generation steps [25] can be described as follows:

(1) Randomly select two large primes p and q, such that pq and $(p-1)(q-1)$ are relatively prime, i.e., $\gcd(pq,\ (p-1)(q-1)) = 1$.
(2) Compute $n = pq$ and $\lambda = (p-1)(q-1)$.
(3) Compute $g = n+1$ and $\mu = \lambda^{-1} \bmod n$.
(4) Public key pk is (n, g) and secret key sk is $(\lambda,\ \mu)$.

b. $\text{Enc}_{paillier}$

The encryption procedure takes the plaintext message $m \in \mathbb{Z}_n$ and pk as input, while the output is the encrypted ciphertext c. Notation $\text{E}_{pk}(\cdot)$ is used to represent the encryption procedure with the encryption key pk. The following two steps complete the corresponding encryption process.

(1) Select a random number $r \in \mathbb{Z}_n$
(2) $\text{c} = \text{E}_{pk}(\text{m}) = g^{m} \cdot r^{n} \bmod n^2$

c. $\text{Dec}_{paillier}$

The decryption procedure takes the ciphertext $c \in \mathbb{Z}^*_{n^2}$ and sk as input, while the output is the decrypted plaintext message m. Notation $\text{D}_{sk}(\cdot)$ is used to represent the decryption procedure with the decryption key sk. The decryption procedure can be represented by the following formulas.

$$m = \text{D}_{sk}(c) = \text{L}\left(c^{\lambda} \bmod n^2\right) \cdot \mu \bmod n \tag{1}$$

where function $\text{L}(\text{x}) = \frac{x-1}{n}$.

The most well-known feature of Paillier cryptosystem is its additive homomorphic property, which enables users to do additive operation directly in the encrypted domain without decrypting the ciphertext first. The additive homomorphic property of Paillier cryptosystem, from the decryption point of view, includes:

(1) The decryption of the product of two ciphertexts c_1 and c_2 can be realized by the addition of the two corresponding plaintexts m_1 and m_2 directly, that is $\text{D}_{sk}(c_1 \cdot c_2) = \text{D}_{sk}\left(\text{E}_{pk}(m_1) \cdot \text{E}_{pk}(m_2) \bmod n^2\right) = \text{E}_{pk}(m_1 + m_2 \bmod n)$.
(2) The decryption of a ciphertext c_1 raised to the power of a plaintext m_2 can be realized by computing the product of the corresponding plaintexts m_1 and m_2 directly, that is $\text{D}_{sk}({c_1}^{m_2}) = \text{D}_{sk}\left(\text{E}_{pk}(m_1)^{m_2} \bmod n^2\right) = m_1 \cdot m_2 \bmod n$.

3.2. Order-Preserving Encryption

Order-preserving encryption or order-preserving encoding (OPE) [26] is a special kind of encryption scheme in which the order relationship between plaintext messages is preserved after the encryption is done. The security of an ideal OPE should be defined with indistinguishability under ordered chosen plaintext attacks (IND-OCPA) [27]. That means the ciphertexts of an OPE should reveal nothing more but the ordering of the plaintexts. Popa et al. [2] presented the first ideal-security protocol for OPE. They showed that the mutability of some ciphertexts is required for ideal-security OPE even if the encryption model is already stateful and interactive. The proposed mutable order preserving encoding (mOPE) protocol is conducted between a client (or data owner) and an OPE server. A basic mOPE protocol scheme, denoted as $(\text{KeyGen}, \text{InitState}, \text{Enc}, \text{Dec}, \text{Order})_{mOPE}$, can be understood as follows:

a. KeyGen_{mOPE}

The client takes a security parameter κ as input and generates the secrete key sk by the key generation module of any symmetric deterministic encryption scheme (SDES), which obeys the pseudo random function security property [28].

b. InitState_{mOPE}

The server initializes its state st by creating a binary search tree, or a b-ary B-tree, as OPE tree and a table as OPE table. The nodes on an OPE tree after encryption contain the ciphertexts and the values of the decrypted ciphertexts in the tree's left subtree nodes are smaller than those in the right subtree ones. The OPE table records a mapping from each ciphertext to its corresponding OPE value.

c. Enc_{mOPE}

The encryption algorithm of mOPE is run interactively by a client and a server. The client takes sk as input to encrypt the original plaintext m. After running the algorithm, the server state st is updated. The full encryption scheme can be addressed as follows:

(1) Client: Encrypt m by SDES and send the ciphertext c to the server.
(2) Server: If OPE table already contains c, do nothing.
(3) Client $\leftrightarrow$ Server: Otherwise the client helps the server insert c to OPE tree by starting a traversal from the root of OPE tree:

   (a) The server sends back the ciphertext c' to the client when it encounters a node.
   (b) The client decrypts c' to m'. If $m' > m$, the client tells the server to go left, otherwise go right.
   (c) Repeat steps a and b until the traversal goes to an empty leaf node and then puts c in there.
   (d) Rebalance OPE tree if needed and update the OPE table.

A ciphertext value in OPE table is computed according to a path from the root to the node and the position of the ciphertext in that node is obtained based on Equation (2):

$$\text{OPE value} = [\text{path}][\text{pos}]0 \ldots 0, \tag{2}$$

where [path] is the concatenation of the binary strings of path pointer indexes from root to the node, where [pos] is the binary string of one plus position index of the ciphertext in that node. The total length of both of them depends on b, which is the maximum ciphertext number in a node. Figure 1 shows an example of the pre-described encryption algorithm. The texts in each node represent a plaintext-ciphertext pair. The blue strings represent the path string from the root to the target node. The red strings represent the ciphertext positions in that node.

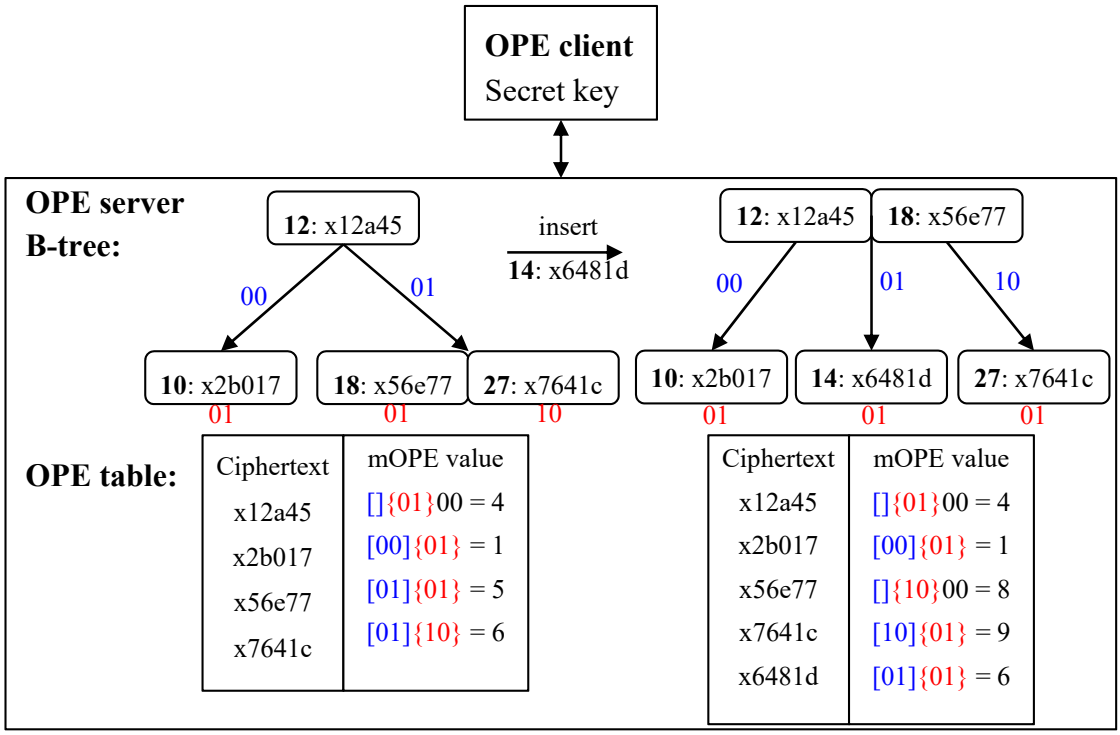

Figure 1. An example of a mutable order preserving encoding (mOPE) scheme using 3-ary B-tree.

Suppose that there are five integers: 12, 10, 18, 27, and 14 to be encrypted by the client. The state of the 3-ary B-tree and the content of OPE table after the first four insertions are on the left of Figure 1. After the insertion of the fifth number 14, the B-tree on the right rebalances its structure and the server updates the OPE table. Take the number 27 as an example. Before the insertion, its path index from the root was 1 so [path] = "01" and one plus its position index becomes $1 + 1 = 2$ so [pos] = "10". After the encryption of number 14, the path index of number 27 becomes 2 so [path] = "10" and one plus the position index becomes $1 + 0 = 1$ so [pos] = "01". The OPE value of number 27 changed from 6 to 9 but the relative ordering did not change. The padding zeros at the end of Equation (2) are used to let all binary strings of OPE values have the same length.

d. Dec_{mOPE}

The decryption of mOPE is just the same as the decryption of SDES. There are no additional operations needed for calculating OPE value or OPE binary string.

e. Order_{mOPE}

Since OPE table stores the ordered OPE values for each ciphertext, by taking ciphertexts as input, we can compare which corresponding plaintext is larger by using the ordering function $\mathrm{Ord}(\cdot)$. Take numbers 10 and 14 in Figure 1 as an example, since $10 < 14$, $\mathrm{Ord}(\text{"x2b017"}) = 1 < \mathrm{Ord}(\text{"x6481d"}) = 6$.

3.3. R-Tree

R-tree is a tree data structure proposed by Antonin Guttman in 1984 [1]. It is widely used in many multi-dimensional data management tasks, such as *K*-NN or geometric search. Each node in an R-tree represents an MBR of all its children MBRs. At the leaf level, each MBR contains a specific number of spatial data points or data objects as its children. R-tree is also a balanced tree, like B-tree, so the average time complexity for searching a data point on it is $O(\log_M N)$, where M is the maximum number of children in each node and N is the total number of data points. Figure 2 shows an example of a two-dimensional R-tree, in which $M = 3$ and $N = 10$.

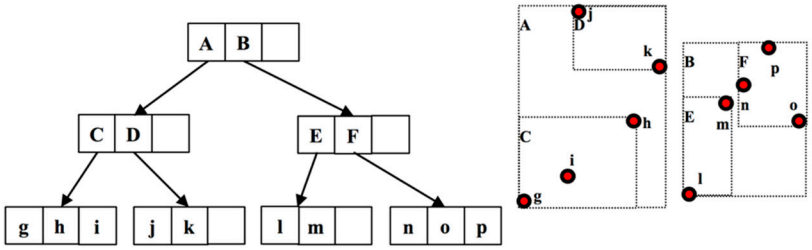

Figure 2. An R-tree example. The nodes with uppercase letters indicate the minimum bounding rectangles (MBRs), while the nodes with lowercase letters are the data points.

3.3.1. Bulk-Loading

Bulk-loading is a kind of construction method for R-tree which loads several data points to an R-tree at once. Apart from the normal construction method, where the data points are inserted one by one, bulk-loading provides a more efficient construction but needs to know the whole data points beforehand. The R-tree constructed by bulk-loading method also has better query performance since the overlap between MBRs can be reduced.

Leutenegger et al. [29] proposed an efficient R-tree bulk-loading algorithm in 1997. The leaf-MBRs of an R-tree constructed by this algorithm do not overlap at all and the rest MBRs in higher levels overlap just a little. The following is the construction algorithm for N D-dimensional data points, where M is the maximum children number in a node:

(1) Set $d = D$; if $N < M$, simply create the root-MBR and end the algorithm.
(2) Calculate the number of leaf-MBR pages, that is $P = \left\lceil \frac{N}{M} \right\rceil$, where $\lceil x \rceil$ denotes the ceiling function.
(3) Sort the data points according to the d-th coordinate and divide them into $S = \left\lceil P^{\frac{1}{d}} \right\rceil$ slices. Each slice has $M \cdot \left\lceil P^{\frac{d-1}{d}} \right\rceil$ data points.
(4) Recursively process each slice, by repeating steps 2 and 3, where N is the current data number in this slice, and $d = d - 1$ until each slice contains only M data points.
(5) Create an MBR for each slice with pointers pointing to all data points in that slice.
(6) Treat MBRs created in step 5 as data points and create higher level MBRs based on step 1.

3.3.2. K-NN Search over R-Tree

The structure of an R-tree can improve the performance of a K-NN search. By traversing a part of MBRs and the data points in them, there is no need to compute the distance between the query point and the whole data points. The idea of the best first search (BFS), proposed in [20] for K-NN search over R-tree, is to access the nearest MBRs or data points all the time by using a priority queue. The input of the algorithm is an R-tree, Rt, and a query point $\boldsymbol{q}$, and the output will be $\boldsymbol{R}$, the set of K data points nearest to $\boldsymbol{q}$. The algorithm can be understood as follows:

(1) Initialize a priority queue Q and the result set $\boldsymbol{R}$.
(2) Enqueue the root of Rt into Q.
(3) While Q is not empty
    (a) Dequeue element $\boldsymbol{e}$ from Q.
    (b) If $\boldsymbol{e}$ is a data point, put it in $\boldsymbol{R}$. If $|\boldsymbol{R}| = K$, return $\boldsymbol{R}$.
    (c) Otherwise
        i. Compute the distance from $\boldsymbol{q}$ to each of the children of $\boldsymbol{e}$.
        ii. Enqueue all children of $\boldsymbol{e}$ into Q using their distance to $\boldsymbol{q}$ as priority.

At the beginning, we enqueue the root of an R-tree to the priority queue. For every element dequeued from the priority queue, we check if it is a data point. If so, add it into the resulting set or recursively enqueue its children to the priority queue if it is still an MBR. The algorithm ends if the resulting set has K elements. Since the priority used in the priority queue is the distance to query point, we can always access to the best candidate containing the resultant points.

4. The Proposed Scheme

4.1. Overview

In this section, the structure of our proposed scheme and some notations will be described first, and the details of the scheme will be addressed in the remaining paragraphs. There are three characters (or players) involved in the proposed scheme, the data owner (*DO*) who owns the original data, the cloud server (*CS*) who provides the storage for data and shares the computation loads of K-NN search, and the query user (*QU*) who makes the K-NN search query request to *CS*, and obtains the final results. In order to preserve the privacy of the data, *DO* encrypts his data before outsourcing them to *CS*. On the other hand, *QU* wants to protect his own query privacy from revealing it to *CS* and *DO*, so he also encrypts the query. We assume that *CS* mentioned here is an honest-but-curious third party, i.e., it will follow the protocol to maintain its business credit but will try to know the data or query as much as it can during the execution of procedures. We leverage Paillier cryptosystem as the encryption method for *DO* and *QU* so that both of them can perform encryption with the same public key while the privilege of decryption retained to *DO*. The homomorphic properties of Paillier will help us shift some operations to *CS* on the encrypted domain. At the same time, the ordering between the encrypted data should be preserved if we want to outsource the construction of R-tree and the running-job of BFS algorithm as well. After executing BFS algorithm, the results returned to *QU* should be decrypted but without revealing them to *DO*; otherwise the approximate value of query will be known by *DO*. *CS* scrambles the results before sending them to *DO* for decryption and helps *QC* restore them, and then obtains the real data results. The structure of the proposed scheme and the detail steps of the proposed protocol are presented in Section 4.3.4.

In the next section, we describe the main encryption method used in the proposed scheme. Some security data management building blocks, used by server for protecting query privacy, are introduced in Section 4.3 and a newly proposed privacy preserving K-NN search method is described in Section 4.3.4.

4.2. Encryption Method

As mentioned above, the encryption method in the proposed protocol is a combination of Paillier cryptosystem and OPE. By utilizing both of their properties, *CS* is able to do additive operations without asking *DO* to decrypt the ciphertexts first and checking the order between them. First of all, we try to replace *SDES* in mOPE by Paillier cryptosystem. There are plenty of issues to be considered because Paillier is neither a symmetric nor a deterministic cryptosystem. In an mOPE, the order relation is preserved by OPE tree whose construction needs decryption. If we change the cryptosystem from symmetric to asymmetric ones, the whole members in that system are able to encrypt data using the public key but cannot obtain the order unless the accessibility to secret key is provided. Therefore, no matter how the other members encrypt their own new data by the public key, they are unable to find out the original data of *DO* through probing attack. More security analyses will be presented in Section 5. On the other hand, the original mOPE made use of a deterministic algorithm so the ciphertexts will be the same if there are two equal plaintexts. mOPE protocol can easily come to an end if a ciphertext already existed in OPE table. If we change *SDES* to Paillier, two equal plaintexts will be encrypted into two different ciphertexts and their OPE values will also be different. It is no big deal here, after we described the whole scheme, because the corresponding OPE values still adjacent to each other in ascending or descending order and the equal ciphertexts are always decrypted to the

same number. The homomorphic properties of Paillier also need to be considered in this combination. The situation that some system members, without having secret key, may create new ciphertexts with the aid of homomorphic addition is similar to that of the pre-described symmetric/asymmetric situation. The OPE value of the new ciphertext obtained from homomorphic addition will not be the addition of two OPE values. In order to obtain the real one, one has to access to secret key from *DO*, first. The full encryption scheme combining Paillier and mOPE is denoted as $(KeyGen, InitState, Enc, SmOPE, Dec, Order)$ in the rest of this paper.

a. KeyGen

The same as $\text{KeyGen}_{paillier}$, *DO* generates public key *pk* for *CS* and *QU* for encryption and keeps the secret key *sk* to himself for conducting the decryption job.

b. InitState

Similar to InitState_{mOPE}, *CS* initializes an OPE tree and an OPE table before *DO* encrypts any data. A node in OPE tree contains the ciphertexts of Paillier and OPE table gives the map between the ciphertexts of Paillier to the values of OPE table.

c. Enc

The same as $\text{Enc}_{paillier}$, the one with *pk* encrypts a plaintext to a ciphertext by Paillier cryptography.

d. SmOPE

SmOPE stands for the secure mOPE. First of all, we replace the encryption of SDES by taking the ciphertext from Enc in the first step of Enc_{mOPE}. Secondly, we do not need to check whether the ciphertext is already in OPE table (in step 2) or not because Paillier is a non-deterministic algorithm. The rest of the protocols will be described, as a security building blocks, in Section 4.3.

e. Dec

DO is the only one who owns *sk* in this scheme and the decryption of a ciphertext is the same as that of $\text{Dec}_{paillier}$.

f. Order

We can obtain the ordering between ciphertexts of Paillier because of the functionality of mOPE. However, the difference apart from Order_{mOPE} is that if m_1 and m_2 are two plaintexts and $m_1 = m_2$, there will be two different ciphertexts c_1 and c_2, where $c_1 \neq c_2$ and $\text{Ord}(c_2) < \text{Ord}(c_1)$ if c_1 is encrypted before c_2.

4.3. Security Building Blocks

In our scheme, only *DO* has the right to use *sk* to decrypt data. This concern is about not letting anybody except *DO* himself have the authority to obtain the original data. *DO* will be more willing to provide data to *QU* through *CS* if he has such a security guarantee. On the other hand, *QU* wants to access the data from *DO* by sending queries without revealing his own personal query to any other people. However, the query of *QU*, even encrypted by the scheme described in Section 4.2, can easily be revealed to *DO* through conducting Dec. Thus, *CS* plays an important role here to meet the needs of both. Some security building blocks are needed to be used by *CS* such that *DO* does not need to share his secret key with others and *QU* can keep his query privacy after decryption at the same time.

4.3.1. R-Tree Construction on the Encrypted Domain

In existing methods, which leveraged R-tree data structure to speed up their *K*-NN search, the construction steps are done by data owners in the plaintext domain. After data owners constructed the tree, they encrypted the plaintext values at the tree nodes and left the pointers between parents and children unencrypted and sent the tree structure to the server. Though the server would not obtain the

real plaintext value in each node, it obviously could know the data access pattern of the query since the pointers are unencrypted. Therefore, if we use similar construction procedure to the existing methods, the data access pattern will be revealed to *DO* when running BFS algorithm. That is unacceptable because *DO* is the one who built the tree in this situation. Once he obtains the access pattern, he can find out nearby data of the query and thus obtains an approximation of the personal query value of *QU*.

To make sure that the above-mentioned problem will not happen, we decided to shift the construction steps of R-tree from *DO* to *CS*. By doing this, we can hide the access pattern from *DO* for one thing and share the computation cost of construction for another. As we can see, the R-tree construction method introduced in the previous section, only needs to sort along with different dimensions and a few computations about the total data number N, dimension D, and the maximum number of children M in a node. The sorting procedure can be successfully done in the encrypted domain by $\mathrm{Ord}(.)$. Parameters N and D should be known by *CS* even if the data outsourced by *DO* are encrypted. In our method, we will let *CS* decide which dimension the sorting procedure will be followed up in addition to parameter M as a randomness of construction. The altered algorithm of R-tree construction on the encrypted domain, denoted as RtConstruct, is shown in Algorithm 1.

Algorithm 1. RtConstruct

Input: N D-dimensional encrypted data points
Output: Encrypted R-tree root pointer Rt

(1) Generate a proper number M and a random sequence $Rnd = \{r_1, r_2, \ldots, r_D\}$, which is a permutation of numbers from 1 to D.
(2) Set $d = D$ and $i = 1$; if $N < M$, simply create the root MBR; let Rt point to it and return.
(3) Calculate the number of the leaf MBR pages, $P = \left\lceil \frac{N}{M} \right\rceil$.
(4) Sort the encrypted data points by $\mathrm{Ord}()$ according to coordinate r_i and divide them into $S = \left\lceil P^{\frac{1}{d}} \right\rceil$ slices. Each slice has $M \cdot \left\lceil P^{\frac{d-1}{d}} \right\rceil$ data points.
(5) Recursively process on each slice by repeating steps 3 and 4; while N becomes the current data number of this slice, let $d = d - 1$ and $i = i + 1$ until each slice contains only M data points.
(6) Create an MBR for each slice with pointers pointing to all data points in that slice.

Treat MBRs created in step 6 as data points and create higher level MBRs based on step 2.

By adding two parameters decided by *CS* and the function $\mathrm{Ord}(.)$ to sort the ciphertexts, *DO* cannot directly know which MBRs or data points are nearby *QU*'s query during the execution of BFS algorithm. The data access pattern is revealed to *CS*, which is the same as the construction process of existing methods; but it does not matter since *CS* cannot perform the decryption to find the plaintext value out.

4.3.2. Secure Compare and Secure mOPE

CS keeps the structure of R-tree in order to prevent *DO* from finding out the query through data access pattern. However, *DO* already has a chance to know it before searching if we simply follow the original mOPE protocol. The problem is that mOPE requires pre-decryption by *DO* and that will directly reveal the query value, which is unacceptable. We need a secure protocol that can make the procedure mOPE more robust. *DO* should know nothing about query, including its plaintext value or the ordering to any of his data point. We leverage a secure comparison protocol, SCompare as shown in Algorithm 2, as a building block and propose the secure version of mOPE, denoted as SmOPE, shown in Algorithm 3. We also let *CS* decide the order of data points input to SmOPE once he obtains all the outsourced data. As a result, *DO* does not obtain the structure details of OPE tree (just like an R-tree); thus, he truly has no idea what the query value is.

Algorithm 2. SCompare

Input: two ciphertexts $E_{pk}(a)$ and $E_{pk}(b)$
Output: $a > b$?
CS :

(1) Generate a random number r and a Boolean flag f
(2) Encrypt r by Enc: $E_{pk}(r)$
(3) If $f == true$, swap $E_{pk}(a)$ and $E_{pk}(b)$.
(4) Add a random number r to both ciphertexts by homomorphic additions: $E_{pk}(a+r) = E_{pk}(a) \cdot E_{pk}(r)$, $E_{pk}(b+r) = E_{pk}(b) \cdot E_{pk}(r)$
(5) Send $E_{pk}(a+r)$ and $E_{pk}(b+r)$ to *DO*.

DO :

(6) Decrypt $E_{pk}(a+r)$ and $E_{pk}(b+r)$ to $a+r$ and $b+r$ by Dec.
(7) If $a+r > b+r$, send 1, otherwise send -1 to *CS* as "*result*".

CS :

(8) If $f == true$, return "–*result*", otherwise return " – *result*".

Algorithm 3. SmOPE

Input: ciphertext c
Output: the newest state st of OPE table
CS :

(1) Start a traversal from OPE tree root
(2) Obtain the ciphertext c' on encountering with a node.

CS $\leftrightarrow$ *DO* :

(3) Run SCompare, use c and c' as input.

CS :

(4) Receive *result*
(5) If result $= 1$, go right; otherwise go left.
(6) Repeat steps 1 to 3 until the traversal goes to an empty leaf node and put c in there.
(7) Rebalance OPE tree if needed and update the state st of OPE table.

4.3.3. Secure Square Euclidean Distance

The last security building block introduced in this section is the secure square Euclidean distance protocol, SSED. This protocol allows *CS* and *DO* to compute the Euclidean distance between two data points together without revealing the plaintext distance value to both of them. It comes up from the security protocols proposed in [20]. In this protocol, first, the distance between two encrypted data points is calculated through homomorphic subtraction. Each dimension of that difference vector is then used to compute the square of the difference vector through a secure multiplication protocol. Finally, the ciphertext of the square Euclidean distance is computed by homomorphically adding all the components of the square difference vector. We do not need to further compute the square root of the previous distance square because it will be used just for finding the priority in BFS algorithm. The secure multiplication, SM, is a multiplication protocol between *CS* and *DO* for two given ciphertexts, and it outputs the ciphertext of multiplication of two corresponding plaintexts without letting both of them know any one of the plaintext values. Details of SM and SSED are shown in Algorithms 4 and 5, respectively.

Algorithm 4. SM

Input: ciphertexts $\mathrm{E}_{pk}(a)$ and $\mathrm{E}_{pk}(b)$
Output: $\mathrm{E}_{pk}(ab)$
CS :

(1) Generate two random numbers r_a and r_b
(2) Encrypt r_a and r_b by Enc and obtain $\mathrm{E}_{pk}(r_a)$ and $\mathrm{E}_{pk}(r_b)$.
(3) Add random numbers to both ciphertexts by homomorphic addition:$\mathrm{E}_{pk}(a+r_a) = \mathrm{E}_{pk}(a)\cdot\mathrm{E}_{pk}(r_a)$, $\mathrm{E}_{pk}(b+r_b) = \mathrm{E}_{pk}(b)\cdot\mathrm{E}_{pk}(r_b)$.
(4) Send them to *DO*.

DO :

(5) Receive $\mathrm{E}_{pk}(a+r_a)$ and $\mathrm{E}_{pk}(b+r_b)$.
(6) Decrypt them to $a+r_a$ and $b+r_b$ by Dec.
(7) Compute $(a+r_a)\cdot(b+r_b) = ab + ar_b + br_a + r_ar_b$.
(8) Encrypt it by Enc and obtain $\mathrm{E}_{pk}(ab + ar_b + br_a + r_ar_b)$.
(9) Send it to *CS*.

CS :

(10) Receive $\mathrm{E}_{pk}(ab + ar_b + br_a + r_ar_b)$.
(11) Compute $\mathrm{E}_{pk}(a)^{r_b} = \mathrm{E}_{pk}(ar_b)$, $\mathrm{E}_{pk}(b)^{r_a} = \mathrm{E}_{pk}(br_a)$ and $\mathrm{E}_{pk}(r_a)^{r_b} = \mathrm{E}_{pk}(r_ar_b)$
(12) Remove the random numbers by computing $\mathrm{E}_{pk}(ab + ar_b + br_a + r_ar_b)\cdot\mathrm{E}_{pk}(ar_b)^{-1}\cdot\mathrm{E}_{pk}(br_a)^{-1}\cdot\mathrm{E}_{pk}(r_ar_b)^{-1} = \mathrm{E}_{pk}(ab)$.
(13) Return $\mathrm{E}_{pk}(ab)$.

Algorithm 5. SSED

Input: encrypted data points $\mathrm{E}_{pk}(\boldsymbol{X}) = \big(\mathrm{E}_{pk}(x_1), \mathrm{E}_{pk}(x_2), \ldots, \mathrm{E}_{pk}(x_D)\big)$ and $E_{pk}(\boldsymbol{Y})$
Output: encrypted square Euclidean distance $\mathrm{E}_{pk}(\| (\boldsymbol{X}-\boldsymbol{Y}) \|)^2$.
CS :

(1) Compute $\mathrm{E}_{pk}(\boldsymbol{X}-\boldsymbol{Y}) = \big(\mathrm{E}_{pk}(x_1-y_1), \mathrm{E}_{pk}(x_2-y_2), \ldots, \mathrm{E}_{pk}(x_D-y_D)\big)$, where $\mathrm{E}_{pk}(x_i-y_i) = \mathrm{E}_{pk}(x_i)\cdot\mathrm{E}_{pk}(y_i)^{-1}$, for $i = 1$ to D
(2) Compute $\mathrm{E}_{pk}\big((\boldsymbol{X}-\boldsymbol{Y})^2\big) = \big(\mathrm{E}_{pk}\big((x_1-y_1)^2\big), \ldots, \mathrm{E}_{pk}\big((x_D-y_D)^2\big)\big)$, where $\mathrm{E}_{pk}\big((x_i-y_i)^2\big)$ is the output of SM. And two $\mathrm{E}_{pk}(x_i-y_i)$, for $1 \le i \le D$, are used as input
(3) $\mathrm{E}_{pk}(\| (\boldsymbol{X}-\boldsymbol{Y}) \|)^2 = \mathrm{E}_{pk}\big((x_1-y_1)^2\big)\cdot\mathrm{E}_{pk}\big((x_2-y_2)^2\big)\cdot\ldots\cdot\mathrm{E}_{pk}\big((x_D-y_D)^2\big)$.
(4) Return $\mathrm{E}_{pk}(\| (\boldsymbol{X}-\boldsymbol{Y}) \|)^2$.

4.3.4. A Newly Proposed Privacy Preserving K-NN Search Scheme (PPKSS)

The setup of our scheme starts from the encryption of all data points by *DO* using *sk*. *DO* publishes the public key *pk* of Paillier and then outsources the encrypted data points to *CS*. Upon receiving the encrypted data, *CS* permutes them and runs SmOPE one ciphertext by another. The maximum ciphertexts number in the node of an OPE tree is also decided by *CS* so that *DO* cannot obtain the structure details of OPE tree. After *CS* processed the RtConstruct algorithm, we are ready for responding to any query request from *QU* now.

QU is able to encrypt his own query any time he wants using *pk*, broadcasted by *DO*, and sends the encrypted query to *CS*. In order to prevent *DO* from keeping himself online all the time (which is impractical), *QU* needs to notify *DO* every time he wants to make a query request. Upon receiving the request from *QU*, *DO* checks if he is a trusted query user and ready for participating the protocols starting from *CS*. The first protocol *CS* needs to start is SmOPE for inserting the query to OPE tree and keeping its order relation with other data points in OPE table. Secondly, *CS* runs the secure BFS algorithm, SBFS as shown in Algorithm 6, in the encrypted domain by using encrypted distance as

priority, calculating the encrypted distance by SSED and pushing it into priority queue by SCompare. Notice that the distance between the query and an MBR should be the distance from the query to the closest point in that MBR which can be found by using Ord(). Figure 3 shows two examples of finding the closest points to queries in an MBR with the aid of the pre-described order-preserving properties. The output returned by the algorithm is a set of encrypted data points which represents the K-NN search results of the query. *CS* then adds a random number to each dimensional component of each point by homomorphic addition and sends the resultant set to *DO*. *DO* decrypts the whole resultant set and returns it to *QU*. At the same time, *CS* would send the plaintexts of the random numbers matrix he adds on the encrypted results to *QU* to help *QU* recover the real plaintext results. The steps of the full privacy preserving *K*-NN search scheme, PPKSS, is shown in Algorithm 7 and the corresponding flow diagram is given in Figure 4.

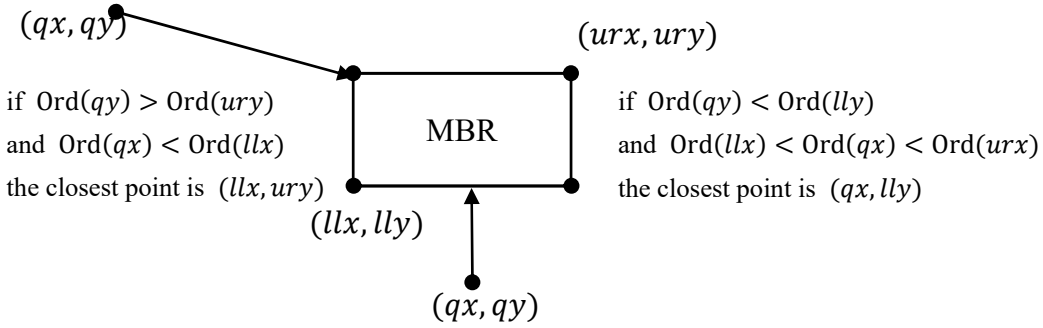

Figure 3. Two example queries for finding their closest points in an MBR by Ord(.).

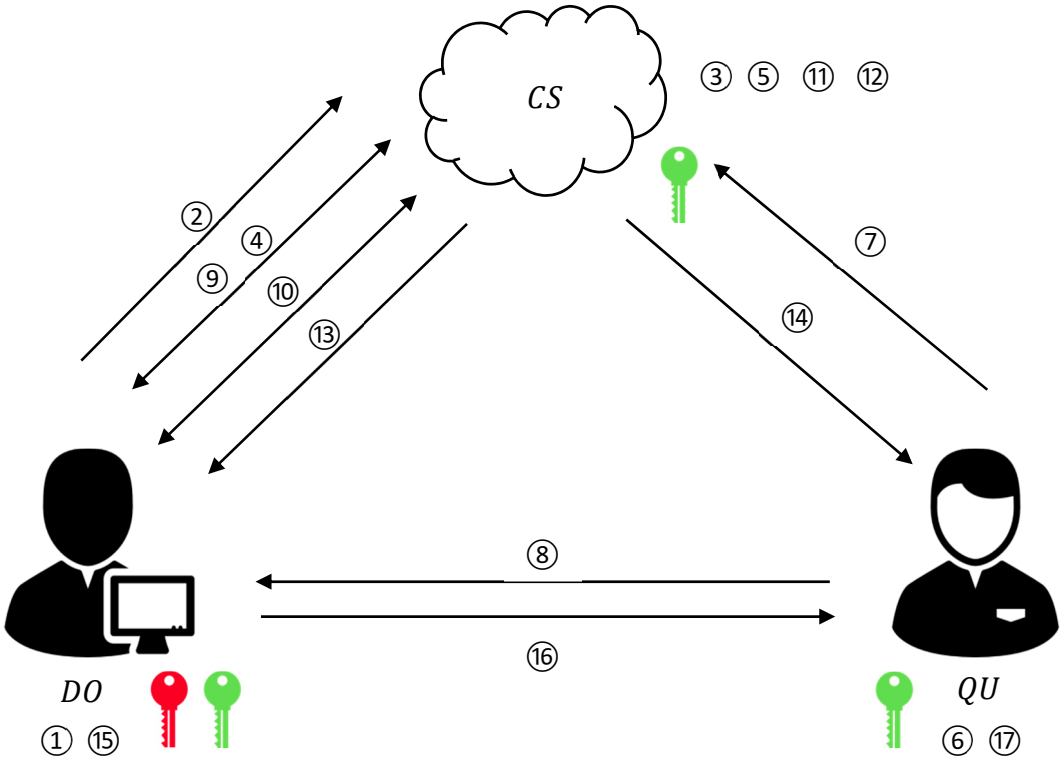

Figure 4. The structure and the flow diagram of the proposed secure K-nearest neighbor (*K*-NN) search scheme. The green key denotes the public key while the red one denotes the corresponding secret key.

Algorithm 6. SBFS

Input: encrypted query $\mathrm{E}_{pk}(\boldsymbol{q})$, R-tree Rt and K.
Output: encrypted resultant set $\mathrm{E}_{pk}(\boldsymbol{R})$
$CS \leftrightarrow DO$:

(1) Initialize an encrypted priority queue Q and the result set $\boldsymbol{R}$
(2) Enqueue the root of Rt into Q
(3) While Q is not empty
   (a) Dequeue element $\mathrm{E}_{pk}(\boldsymbol{e})$ from Q
   (b) If $\boldsymbol{e}$ is a data point, put it in $\boldsymbol{R}$; if $|\boldsymbol{R}| = K$, return $\boldsymbol{R}$
   (c) Otherwise,
      i. Compute the distance from $\mathrm{E}_{pk}(\boldsymbol{q})$ on the basis of all the children of $\mathrm{E}_{pk}(\boldsymbol{e})$ by SSED.
      ii. Enqueue all children of $\mathrm{E}_{pk}(\boldsymbol{e})$ into Q using their encrypted distance to $\mathrm{E}_{pk}(\boldsymbol{q})$ as priority and compare them by SCompare.

Algorithm 7. PP*K*SS

Input: DO : generate data points $\boldsymbol{P} = \left\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\right\}$, where $\boldsymbol{p}_i = \left(p_1^i, p_2^i, \ldots, p_D^i\right)$, $p_j^i \in \mathbb{Z}$; QU : make a query $\boldsymbol{q} = \left(q_1^i, q_2^i, \ldots, q_D^i\right)$), $q_j^i \in \mathbb{Z}$ and K
Results: QU : receive K-NN result $\boldsymbol{R} = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_K\}$

Setup

DO :

① Encrypt all dimensions of all points of $\boldsymbol{P}$ by Enc and obtain $\mathrm{E}_{pk}(\boldsymbol{P})$.
② Outsource $\mathrm{E}_{pk}(\boldsymbol{P})$ to CS.

CS :

③ InitState.

$CS \leftrightarrow DO$:

④ For each encrypted dimension of each point in $\mathrm{E}_{pk}(\boldsymbol{P})$, do SmOPE.

CS :

⑤ RtConstruct.

Search

QU :

⑥ Encrypt all dimensions of $\boldsymbol{q}$ by Enc and obtain $\mathrm{E}_{pk}(\boldsymbol{q})$.
⑦ Send $\mathrm{E}_{pk}(\boldsymbol{q})$ and K to CS.
⑧ Send a request to DO.

$CS \leftrightarrow DO$:

⑨ For each dimension of $\mathrm{E}_{pk}(\boldsymbol{q})$, do SmOPE.
⑩ Run SBFS algorithm, and obtain encrypted result set $\mathrm{E}_{pk}(\boldsymbol{R})$.

CS :

⑪ Generate a $K \times D$ random matrix M.
⑫ For $i = 1$ to K, $j = 1$ to D, add the random number $M(i, j)$ to result by $\mathrm{E}_{pk}(r_j^i) \cdot \mathrm{E}_{pk}(M(i, j)) = \mathrm{E}_{pk}\left(r_j^i + M(i, j)\right)$ and denote the result as $\mathrm{E}_{pk}(\boldsymbol{R'})$
⑬ Send $\mathrm{E}_{pk}(R')$ to DO.
⑭ Send M to QU.

DO :

⑮ Decrypt $\mathrm{E}_{pk}(\boldsymbol{R'})$ to $R^{'}$.
⑯ Send $\boldsymbol{R'}$ to QU.

QU :

⑰ For $i = 1$ to K, $j = 1$ to D, remove the random number $M(i, j)$ out. $r_j^i + M(i, j) - M(i, j) = r_j^i$ and obtain $\mathbf{R}$

5. Security Analysis

5.1. Indistinguishability under Order Chosen Plaintext Attack

The ideal security of OPE is founded on its indistinguishability under order chosen plaintext attack (IND-OCPA), defined by Boldyreva et al. [27]. Generally speaking, the ideal OPE method is IND-OCPA secure if the encryption procedure leaks nothing more than the ordering between ciphertexts. An IND-OCPA security game between *DO*, *CS*, and a malicious adversary (*MA*) can clearly describe the definition. Consider that there are two sequences $\boldsymbol{x} = \{x_1, x_2, \ldots, x_l\}$ and $\boldsymbol{y} = \{y_1, y_2, \ldots, y_l\}$ sent from an *MA* to *DO*, where l is the sequence length and the sequences have the same order relation, that is, $x_i < x_j$ if and only if $y_i < y_j$ for all $i,\ j \in \mathbb{Z}_l^*$. *DO* then randomly selects a sequence to encrypt and *MA* would attack the model by guessing which sequence has been chosen. The definition assumes that *MA* is able to check every ciphetexts of the chosen sequence and the corresponding state of the server if it is an interactive scheme. We say *MA* wins, Win_{MA}, if the right sequence randomly selected by *DO* is guessed and the satisfaction of Equation (3) realizes the IND-OCPA security. That is

$$\Pr[\mathrm{Win}_{MA}] \leq \frac{1}{2} + \mathrm{negl}(\kappa), \tag{3}$$

where Pr[A] denotes the probability of event A and negl(κ) is a negligible function with parameter κ. That means *MA* can only have a negligible advantage over random guessing if the scheme is IND-OCPA secure.

We prove that our modified mOPE combining with Paillier cryptosystem is IND-OCPA secure based on their primitive security guarantees. Paillier cryptosystem has been proved to provide semantic security against chosen plaintext attack (IND-CPA), so, by definition, it meets a higher security requirement than an OPE scheme. That is, the order relationships of the two sequences sent by *MA* are not necessary the same, if the scheme is IND-CPA secure. Therefore, we can prove the proposed encryption method provides IND-OCPA security merely by discussing that the information leaks from *CS* while running mOPE is indistinguishable between two sequences. We inductively prove it on the basis of the number of encrypted messages. The base case is at the beginning of the procedure, when no message is encrypted yet, *CS* initializes the server state, which leaks the same original information to *MA*. We assume that *MA* receives the same information no matter which sequence has been encrypted and cannot distinguish the results after conducting the i-th encryption. At the $(i+1)$-th encryption, the sustentation of the indistinguishability completes the proof. Since sequences $\boldsymbol{x}$. and $\boldsymbol{y}$ have the same order relation, the associated OPE trees have the same structure after the i-th encryption has been conducted. The traversal over the tree and the update of OPE table, at the $(i+1)$-th encryption, are also in the same way as before. Therefore, by observing the server state, *MA* can obtain nothing but the ordering between the ciphertexts, which is the same on both sequences. By mathematical induction, the procedure of mOPE reveals the same ordering information to *MA*. Additionally, owing to IND-CPA security of Paillier, we complete the proof that the encryption scheme can survive the order chosen plaintext attack (IND-OCPA).

5.2. Privacy Preserving of Data

We claim that the data privacy of our scheme is preserved if the following two statements can be justified:

(1) *CS* is prevented from knowing the actual plaintexts of the original data.
(2) *QU* receives only *K*-NN search results in every query request, and has no idea about the rest of the dataset.

Since the data are encrypted by Paillier cryptosystem before outsourcing and *sk* is held by *DO* all the time, *CS* cannot know the plaintexts directly. Therefore, our analyses focus on the interactions within mOPE. Despite assuming an honest-but-curious server, we consider the situation that *CS* takes

wrong ciphertexts while running SmOPE as launching a probing attack and tries to find the plaintexts of dataset out. *DO* can prevent this kind of attack by setting a reasonable threshold on the maximum number of *SCompare* taken by *CS* to complete the system setup. The exceeding amount of *SCompare* is regarded as a malicious behavior and *DO* could shut the comparison service down before the privacy exposes.

The second requirement for preserving the data privacy is achieved much simpler than the first one, since *QU* has no chance at all to directly contact with the encrypted dataset. The data involved by *K*-NN search are only shared by *CS* and *DO*, so *QU* has no idea about the dataset except the receiving results. Certainly, one should also setup a threshold for the maximum acceptable value of *K* to prevent *QU* from sending an extremely large *K* value and retrieving the entire dataset.

5.3. Privacy Preserving of Query

Apart from the preserving of data privacy, the proposed scheme preserves query privacy as well. Similarly, two required statements have to be justified to fulfill the privacy preserving of query.

(1) *CS* is prevented from knowing the actual value of query.
(2) While running the search scheme, *DO* must have no idea about the query, including the plaintext of it, the order relation with any other point in the dataset and the corresponding *K*-NN search result.

The first statement can be justified in a similar way to that of the previous section, i.e., the query can be treated as another new encrypted data point. By executing SmOPE protocol, the only information leaks to *CS* is the order relation. On the other hand, since our approach needs *QU* to make a request to *DO* at the same time, *CS* is unable to maliciously pretend as a fake query user and send a large sequence of values to probe for the plaintext data.

The issues of query privacy prevention against *DO* are more complicated because *DO* is the one has decryption key *sk*. We design the scheme carefully by bringing in randomness on every part of operations involving the decryption process. First of all, the structure of OPE tree is unknown to *DO* as the parameters are decided by *CS*. Under the above condition, SmOPE involves a sequence of comparisons of random values at *DO*'s aspect. The true value of the query and the order relation with data points in the dataset are hidden as a result. Secondly, the structure of R-tree is also unknown to *DO*, similar to the case of OPE tree. Therefore, *DO* has no choice but treats SBFS algorithm as an ordinary sequence of multiplications and comparisons. Moreover, the protocols used in SBFS, i.e., SSED and SCompare, always add random numbers to the ciphertexts, which then will be decrypted by Paillier homomorphic property, so that *DO* is performing operations on obscured data values. Finally, *DO* receives a set of encrypted results at the end of PPKSS; however, even if the decryption is performed upon all of them, *DO* still cannot obtain the *K*-NN search result associated with the query eventually, since they have been randomly permuted in advance.

6. Performance Evaluation

The effectiveness of our scheme, PPKSS, is demonstrated in the following two different ways. First, we analyze the complexity of the scheme in the next section. Secondly, we show the experimental results in Section 6.2. The full PPKSS can be separated into different parts and each part will be discussed independently.

6.1. Complexity Analysis

First of all, *DO* encrypts the dataset attribute-wise and outsources them to *CS*, which takes $O(N * D)$ encryptions. This part of tasks can be fully parallelized since the encryption of each data can be done separately. After the outsourcing, *CS* performs $O(N * D)$ SmOPE protocol to preserve the order relation of each ciphertext. We further disassemble SmOPE protocol and find that it is composed of $O(\log(N * D))$ SCompare and $O(1)$ OPE table update, where $\log(N * D)$ indicates the height of the OPE tree. The computation load of SCompare is shared by *DO* and *CS*, where they

need $O(1)$ Paillier decryption and plaintext comparison and $O(1)$ Paillier encryption and ciphertext multiplication, respectively. Additionally, a hash table is used to implement the OPE table, so we claim that the average update and access complexity takes $O(1)$ plaintext operations.

After the construction of OPE tree, we leverage the order relation store in OPE table to build the R-tree, RtConstruct, which is done simply based on sorting. The complexity of RtConstruct is $O(DN\log(N))$, and it can be derived as follows.

Since RtConstruct sorts data along with every dimension at each level of an R-tree, the comparison time using quick sort is the multiplication of D and the summation of sorting complexity at each level. That is,

$$\begin{gathered} D*\sum_{i=0}^{\log_M N}\frac{N}{M^i}\log\left(\frac{N}{M^i}\right) \\ = DN(\left(\log(N)*\sum_{i=0}^{\log_M N}\frac{1}{M^i}\right)-\left(\log(M)*\sum_{i=0}^{\log_M N}\frac{i}{M^i}\right)) \\ \approx DN(\log(N)-\left(\log(M)*\sum_{i=0}^{\log_M N}\frac{i}{M^i}\right))\ (\sin ce\ N\gg M) \\ \leq DN\log(N) \end{gathered}$$

QU enrolls the system after the setup steps were completed. We will discuss why the scheme is claimed to be with lightweight user loads, later. The complexity of query encryption is the same as the complexity of encryption of an individual data point, where the Paillier encryption is done by *QU*. We analyze the complexity of SBFS based on the original search algorithm over an R-tree in the plaintext domain [29]. The analysis is simplified under the assumption that the data in the dataset are uniformly distributed. Let $\boldsymbol{H}$ be the hypersphere centered at the query $\boldsymbol{q}$, with r (the distance from $\boldsymbol{q}$ to the K-th result $\boldsymbol{r}_K$), as radius. The search complexity or the total time SSED would take is the access number of MBRs from the root of an R-tree to the leaf-MBRs, which is $O(\log N)$, plus the total access number of data points of the leaf-MBRs (i) inside H, which is $O(K)$, or (ii) intersected by $\boldsymbol{H}$, which is the most complicated part to analyze and it is derived as follows.

For simplification, another assumption is made, that is, we assume each leaf-MBR intersected by $\boldsymbol{H}$ forms a hypercube with average occupancy of c data points. Since the data points are assumed to be uniformly distributed, the expected volume of search region is $\frac{K}{N}$ and the expected volume of each leaf-MBR is $\frac{c}{N}$. Moreover, the volume of $\boldsymbol{H}$ is proportional to $\pi^{\frac{D}{2}}r^D$, so if $\pi^{\frac{D}{2}}r^D=\frac{K}{N}$, $r=\sqrt[D]{\frac{K}{\pi^{\frac{D}{2}}N}}$. Similarly, the side volume of each leaf-MBR $s=\sqrt[D]{\frac{c}{N}}$. In addition, the number of leaf-MBRs intersected by $\boldsymbol{H}$ is the same as that intersected by the circumscribed cube of $\boldsymbol{H}$ [30,31], so the $2*D$ sides of a circumscribed cube intersect $\left\lfloor\frac{(2r)^{D-1}}{s}\right\rfloor\leq\frac{(2r)^{D-1}}{s}$ regions, where $\lfloor x\rfloor$ denotes the floor function. Thus, the total expected data access number of leaf-MBRs intersected by $\boldsymbol{H}$ is:

$$\begin{gathered} 2D*\frac{(2r)^{D-1}}{s}*c \\ = 2D*\frac{2^{D-1}\left(\frac{K}{\pi^{\frac{D}{2}}N}\right)^{\frac{D-1}{D}}}{\left(\frac{c}{N}\right)^{\frac{D-1}{D}}}*c\approx 2D*\sqrt[D]{cK} \end{gathered}$$

In summary, the total number operations of SSED is bounded by $O\left(\log N+K+DK^{\frac{D-1}{D}}\right)$ and $O\left(\log N+DK^{\frac{D-1}{D}}\right)$ is the expected node number in the priority queue.

Each sub-task of SSED is analyzed separately to see the workload in detail. From the pseudo-codes given in Section 4.3.3, it follows that SSED runs $O(D)$ SM protocols, each of which needs $O(1)$ decryption, plaintext multiplication and encryption on the site of *DO*, and $O(1)$ encryption, ciphertext multiplication, and exponentiation on the site of *CS*.

On the other hand, a heap is used to implement the priority queue. If the expected number of MBRs and data points in the priority queue is $|Q|$, which is about $O\left(\log N+DK^{\frac{D-1}{D}}\right)$ in total, then it takes $O(\log(|Q|))$ SCompare individually to enqueue or dequeue data from the queue. Moreover, the complexity of our enqueue and dequeue approximates to $O\left(\log N+K+DK^{\frac{D-1}{D}}\right)$.

In the end of PP*K*SS, *CS* takes $O(DK)$ Paillier encryptions and multiplications to randomly permute the results and *DO* takes $O(DK)$ decryptions. Finally, only extremely light, $O(DK)$, plaintext subtractions need to be done by *QU*.

We summarize our complexity analysis results in Table 2 by showing the executed complexities of different type of operations in different protocols of our scheme. Clearly, from the table, the operations in Paillier ciphertext domain, which cost most expensively, are performed by *CS*. As for *DO*, he or she focuses mainly on the decryption tasks and needs to do a few plaintext operations. Most importantly, a very lightweight workload is put on *QU*, which raises the possibility of extending our approach to a user with resource-limited mobile devices. Another highlight of our approach is that the search complexity, $O\left(\log N + K + DK^{\frac{D-1}{D}}\right)$, is faster than linear scan, whose complexity is $O(N)$, since N is much greater than D and K, in general.

Table 2. The complexity analysis results of the proposed scheme. The complexities represented by red, blue, and green colors indicate the computational loads of data owner (*DO*), cloud server (*CS*), and query user (*QU*), respectively.

		Enc/Dec	Paillier Operation	Plaintext Operation
Enc for Data		$O(ND)$		
SmOPE for Data		$O(ND\log(ND))$ $O(ND\log(ND))$	$O(ND\log(ND))$	$O(ND\log(ND))$ $O(ND)$
RtConstruct				$O(DN\log(N))$
Enc for Query		$O(D)$		
SmOPE for Query		$O(D\log(ND))$ $O(D\log(ND))$	$O(D\log(ND))$	$O(D\log(ND))$ $O(D)$
SBFS	**SSED**	$O\left(D\left(\log N + K + DK^{\frac{D-1}{D}}\right)\right)$ $O\left(D\left(\log N + K + DK^{\frac{D-1}{D}}\right)\right)$	$O\left(D\left(\log N + K + DK^{\frac{D-1}{D}}\right)\right)$	$O\left(D\left(\log N + K + DK^{\frac{D-1}{D}}\right)\right)$
	Access	$O\left(\log\lvert Q\rvert\left(\log N + K + DK^{\frac{D-1}{D}}\right)\right)$ $O\left(\log\lvert Q\rvert\left(\log N + K + DK^{\frac{D-1}{D}}\right)\right)$	$O\left(\log\lvert Q\rvert\left(\log N + K + DK^{\frac{D-1}{D}}\right)\right)$	$O\left(\log\lvert Q\rvert\left(\log N + K + DK^{\frac{D-1}{D}}\right)\right)$
Return Results		$O(DK)$ $O(DK)$	$O(DK)$	$O(DK)$

6.2. Experimental Results

The overall performance of our proposed scheme is also demonstrated by several simulation experiments. We realized the proposed scheme, including *DO*, *CS*, and *QU*, in C-language and executed them on a laptop running macOS Sierra 10.12.3, with 2.7 GHz Intel Core i5 and 8 GB 1867 MHz DDR3 memory. The test dataset for *DO* is randomly generated with different N and D. We use 4-ary B-tree as an OPE tree and $M = 5$ for R-tree. The key length of Paillier cryptosystem is set to 1024 bits. The rest of this section shows the executing time of each part of PP*K*SS except for the encryption operations of dataset and query, which are simply accomplished by applying the original Paillier encryption. In general, a Paillier encryption of a plaintext data or a query takes about 7.5 milliseconds (ms) under our settings. Specifically, the timing responses of the proposed scheme will be examined separately so as to represent the workload of each character, independently. Moreover, the communication cost is also listed because the proposed procedures always include interactions between two parties.

First of all, Figure 5 shows the timing cost of performing SmOPE on a single Paillier ciphertext, that is, the time required to insert one ciphertext into an OPE tree and update the corresponding OPE table. When the total number of encrypted attributes grows from 100,000 to 1,000,000, each insertion may take longer time, changing from 40 ms to 45 ms, since the height of the tree grows. Notice that it may take a long time to complete the order preserving setup, but that does not matter since it is a one-time job.

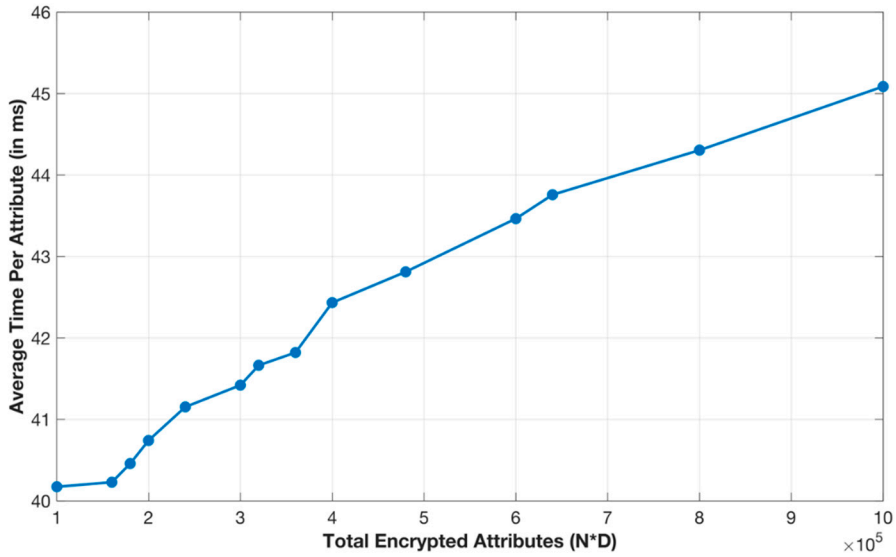

Figure 5. Average execution time for secure mOPE (SmOPE) per attribute with respect to different numbers of total attributes.

The second task of the setup is RtConstruct, which can totally be accomplished by *CS*. We illustrate the required timing for constructing the R-tree of the testing datasets with different sizes and dimensions in Figure 6. As the size of 2-dimensional dataset grows from 20,000 to 100,000, the construction costs changed from 8 s to 45 s. Since the required construction time is proportion to the dimension of the dataset, as expected, Figure 6 shows it grows linearly with the respect to the number of involved data attributes.

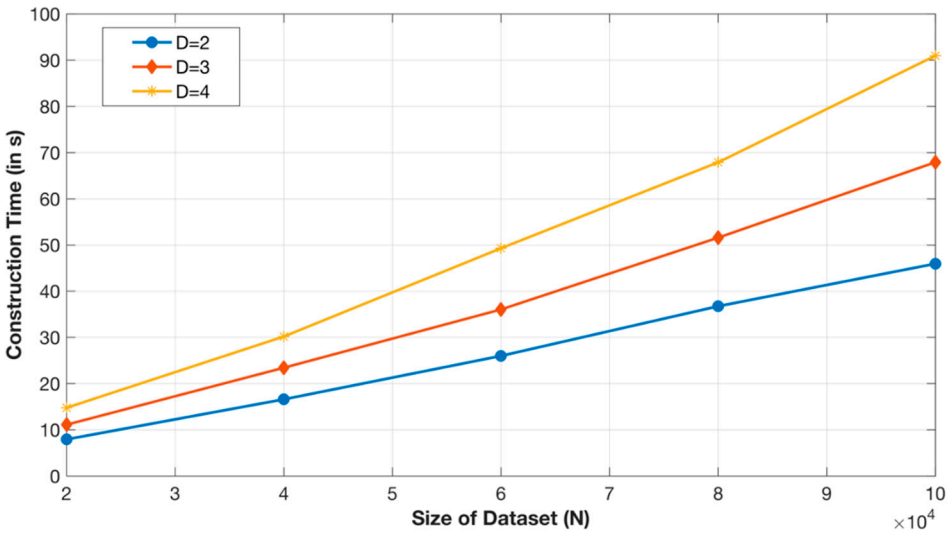

Figure 6. Construction timing cost of R-tree in *CS* with different numbers of data and dimensions.

We define the query search time of our scheme as a specific time-duration, which starts from the time after *CS* completed SmOPE for inserting the encrypted query into an OPE table, and ends after *QU* removed the random numbers out and got the unencrypted results correctly. The size of dataset N, the dimension of data D, and the range value K for a K-NN search are used as parameters to generate various sets of experiments. Each experimental result (corresponding to a different set of parameters) is obtained by running 100 queries on it and we record the resultant average search time. First, by fixing $D = 2$ and $K = 1$, that is, to find the nearest neighbors in a 2-dimensional dataset, we illustrate the obtained average search time, for datasets with sizes changed from 10,000 to 100,000, in Figure 7.

Since our approach utilizes R-tree structure to manage the dataset, the search time grows only a little while the size of dataset increases tremendously. Next, by setting $K = 1$ and $N = 20,000$, Figure 8 shows the search time behavior of our work over multi-dimensional dataset. The average search time grows apparently as the data dimension increases, for example, the search time changed from 3.52 s to 53.49 s when D changed from 2 to 6. The reasons for this phenomenon come from the relatively complicated structure of R-tree and the increase of the number of necessary visited data points in MBRs, if high dimensional datasets are involved. In the last experiment, we fix $N = 20,000$ and $D = 2$ and conduct K-NN searches by varying K, and the resulting search time is shown in Figure 9. As the value of K increases from 1 to 20, the search time grows linearly from 3.52 s to 9.17 s.

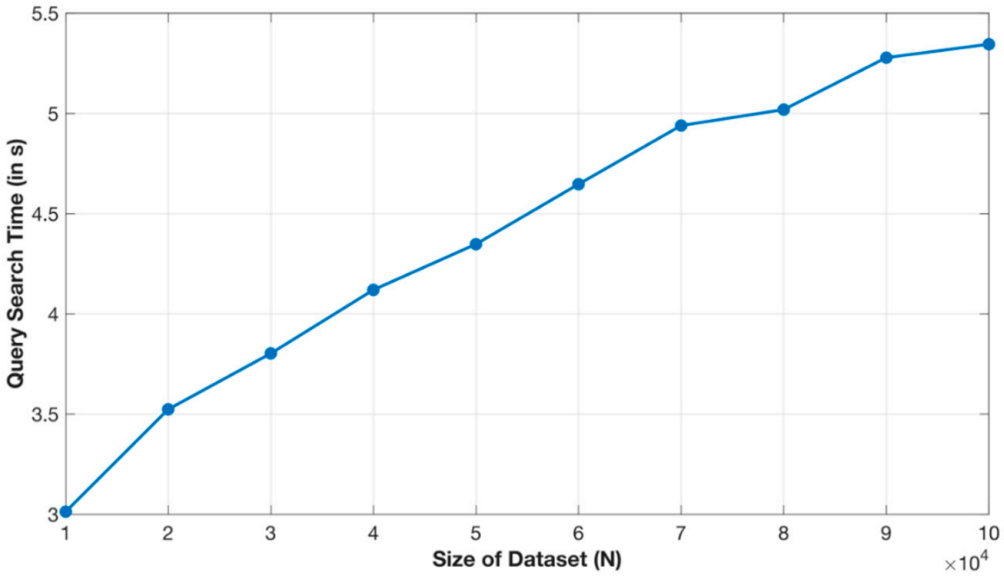

Figure 7. Average nearest neighbor query search time with respect to different sizes of two-dimensional datasets, in which $D = 2$ and $K = 1$.

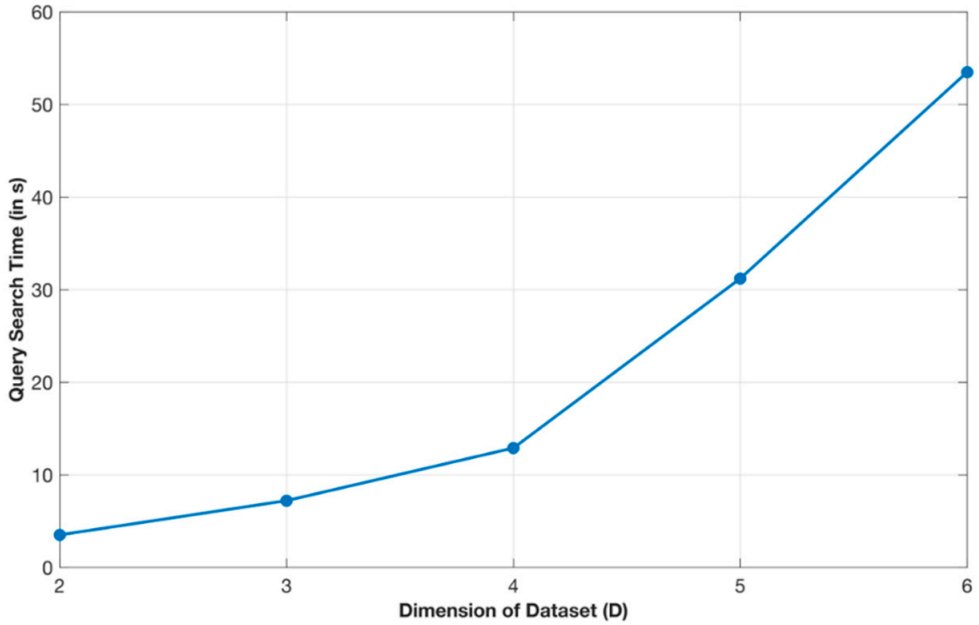

Figure 8. Average nearest neighbor query search time of multi-dimensional datasets, in which $N = 20,000$ and $K = 1$.

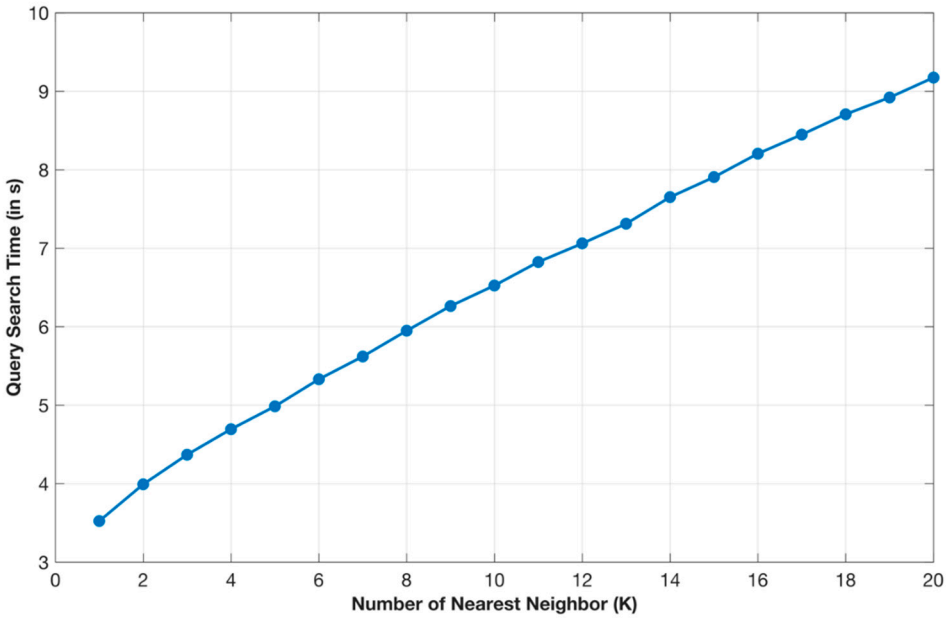

Figure 9. Average query search time with respect to different K, in which $N = 20,000$ and $D = 2$.

Finally, we separate the search time of our approach into the combination of *DO*'s CPU time, *CS*'s CPU time, and *QU*'s CPU time to clearly identify the workload of each character. The data transmission costs are also listed to evaluate the extra communication overhead of the proposed scheme. We take $N = 20,000$, $K = 20$ and different D as testing parameters and report the details of the composition of a search time in Table 3. One can see that although the search time is a little bit longer as the dimension of dataset is increased, about 80% of computational loads are outsourced to *CS*, and *DO* takes care of the rest 20%. At the *QU* site, the computation overhead is extremely lightweight and is negligible. This fact makes the proposed scheme more applicable and practical even if *DO* and *QU* are worked on resource-limited environments. Notice that there is a high correlation between the communication cost and the total query search time, which is quite reasonable.

Table 3. Details of the query search time of different D with N=20,000 and K=20. (Time is measured in seconds and communication cost is measured in Kb, Percentage abbreviated to Per.).

D	Total Query Time	*DO* Time	Per.	*CS* Time	Per.	*QU* Time	Per.	Communication Cost
2	9.17	2.05	22.3%	7.04	76.7%	0.000003	<0.01%	469.003
3	19.03	3.93	20.7%	14.91	78.4%	0.000003	<0.01%	978.568
4	40.24	7.85	19.5%	32.01	79.5%	0.000003	<0.01%	2091.073
5	77.8	14.52	18.7%	62.55	80.4%	0.000004	<0.01%	4030.649
6	135.06	24.41	18.1%	109.43	81.0%	0.000006	<0.01%	7059.019

7. Conclusions and Future Works

In this work, a privacy preserving K-NN search scheme is constructed based on a newly proposed encryption scheme, which preserves not only comparison but also addition operations in the encrypted domain. Moreover, some disadvantages of related works are addressed and released by our approach. The proposed scheme is realized on the bases of several security protocols among a data owner, a cloud server and a query user, and we manage the dataset with the structure of an R-tree. Security analysis shows that the proposed encryption scheme not only achieves IND-OCPA security but also preserves both the data privacy and query privacy. Furthermore, we justify that our approach runs in sub-linear time by complexity analysis. And finally, experimental results demonstrate its effectiveness

with respect to different sizes of datasets. Most of all, the impressive lightweight user loads enhance the applicability of our approach to resource-limited mobile platforms.

As an interesting extension of PPkNN, Wu et al. [32] presented the first solution to the so-called Group k-nearest neighbor (kGNN) search problem, which allows a group of n mobile users to jointly retrieve k points from a location-based service provider (LSP) that minimizes the aggregate distance to them, at the same time. The authors identified four protection objectives in the privacy-preserving kGNN (PPkGNN) search: (i) every user's location should be protected from LSP; (ii) the group's query and the query answer should be protected from LSP; (iii) LSP's private database information should be protected from users, i.e., the users cannot learn more information beyond the answer they requested; (iv) every user's location should be protected from the other users in the group. Since the encryption mechanism of our scheme is based on the Paillier cryptosystem, same as in [32], we might extend our PPkNN scheme to solve PPkGNN search problem, which is one of our future research directions. Furthermore, how to improve the overall system performance in regarding to high-dimensional datasets is another research direction.

One of the anonymized reviewers brings the following important and interesting issue to us: "How a data owner can be contributed in the process of securing/encrypting cloud dataset?", which is an important aspect of the dynamically changing environments, such as in Cloud-based servers. Although this topic is out of the main scope of our current work, this subject should be included in our future research directions, as suggested by the reviewer. Specific thanks to the same reviewer for mentioning the following useful reference [33] to us. As mentioned by the same reviewer: "Using more than one Cloud servers the processing and computational overheads can be further reduced." Of course, this interesting and challenging issue would be included in our future research topics. Thanks also to the reviewer for bringing the following useful references to us [34,35].

Our scheme would fail if a cloud server did not follow the pre-defined protocols and returned the wrong results to the query user. In other words, we think how to design a cloud-based privacy preserving *K*-NN search scheme which is robust to "Man-in-the-middle attack" is still an open problem.

Author Contributions: Conceptualization, Y.-C.H. and J.-L.W.; Data curation, Y.-C.H.; Formal analysis, Y.-C.H.; Investigation, Y.-C.H.; Methodology, Y.-C.H. and J.-L.W.; Software, Y.-C.H.; Supervision, J.-L.W.; Validation, Y.-C.H.; Visualization, Y.-C.H. and C.-H.H.; Writing—original draft, Y.-C.H. and J.-L.W.; Writing—review & editing, C.-H.H. and J.-L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Guttman, A. R-trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Boston, MA, USA, 18–21 June 1984; pp. 47–57.
2. Popa, R.A.; Li, F.H.; Zeldovich, N. An Ideal-Security Protocol for Order-Preserving Encoding. In Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 19–22 May 2013; pp. 463–477.
3. Paillier, P. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Prague, Czech Republic, 2–6 May 1999; pp. 223–238.
4. Behera, P.K.; Khilar, P.M. A Novel Trust Based Access Control Model for Cloud Environment. In Proceedings of the International Conference on Signal, Networks, Computing, and Systems, Delhi, India, 25–27 February 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 285–295.
5. "Mandatory Access Control." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. Available online: http://en.wikipedia.org/wiki/Mandatory_access_control (accessed on 21 December 2019).
6. Samarati, P.; De Vimercati, S.C. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 137–196.
7. Sandhu, R.S.; Coyne, E.J.; Feinstein, H.L.; Youman, C.E. Role-based access control models. *Computer* **1996**, *29*, 38–47. [CrossRef]

8. Guoyuan, L.; Wang, D.; Bie, Y.; Lei, M. MTBAC: A mutual trust-based access control model in Cloud computing. *China Commun.* **2014**, *11*, 154–162. [CrossRef]
9. Kayers, A.S.M.; Rahayu, W.; Dillon, T.; Chang, E.; Han, J. Context-aware access control with imprecise context characterization for cloud-based data resources. *Future Gener. Comp. Syst.* **2019**, *93*, 237–255.
10. Kwon, Y.; Seo, J.H.; Park, T.B. Dynamic Role-based User Service Authority Control and Management on Cloud Computing. *Univers. J. Electr. Electron. Eng.* **2019**, *6*, 79–93. [CrossRef]
11. Voloch, N.; Levy, P.; Elmakies, M.; Gudes, E. *An Access Control Model for Data Security in Online Social Networks Based on Role and User Credibility. The Third International Symposium on Cyber Security Cryptography and Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 156–168.
12. Kayes, A.S.M.; Rahayu, W.; Tharam, T. Critical situation management utilizing IoT-based data resources through dynamic contextual role modeling and activation. *Computing* **2019**, *101*, 743–772. [CrossRef]
13. Rong, H.; Wang, H.-M.; Liu, J.; Xian, M. Privacy-preserving k-nearest neighbor computation in multiple cloud environments. *IEEE Access* **2016**, *4*, 9589–9603. [CrossRef]
14. Park, J.; Lee, D.H. Privacy preserving k-nearest neighbor for medical diagnosis in e-health cloud. *J. Healthc. Eng.* **2018**, *2018*, 4073103. [CrossRef] [PubMed]
15. Burkhart, M.; Dimitropoulos, X. Privacy-preserving distributed network troubleshooting—bridging the gap between theory and practice. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **2011**, *14*, 31. [CrossRef]
16. Yin, H.; Zhang, J.; Xiong, Y.; Huang, X.; Deng, T. PPK-means: Achieving privacy-preserving clustering over encrypted multi-dimensional cloud data. *Electronics* **2018**, *7*, 310. [CrossRef]
17. Wong, W.K.; Cheung, D.W.-L.; Kao, B.; Mamoulis, N. Secure KNN Computation On Encrypted Databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Providence, RI, USA, 29 June–2 July 2009; pp. 139–152.
18. Hu, H.; Xu, J.; Ren, C.; Choi, B. Processing Private Queries over Untrusted Data Cloud Through Privacy Homomorphism. In Proceedings of the IEEE 27th International Conference on Data Engineering, Hannover, Germany, 11–16 April 2011; pp. 601–612.
19. Yao, B.; Li, F.; Xiao, X. Secure Nearest Neighbor Revisited. In Proceedings of the IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, Australia, 8–12 April 2013; pp. 733–744.
20. Elmehdwi, Y.; Samanthula, B.K.; Jiang, W. Secure K-nearest Neighbor Query over Encrypted Data in Outsourced Environments. In Proceedings of the IEEE 30th International Conference on Data Engineering, Chicago, IL, USA, 31 March–4 April 2014; pp. 664–675.
21. Wang, B.; Hou, Y.; Li, M. Practical and Secure Nearest Neighbor Search on Encrypted Large-scale Data. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9.
22. Xu, R.; Morozov, K.; Yang, Y.; Zhou, J.; Takagi, T. Privacy-preserving K-nearest Neighbour Query on Outsourced Database. In Proceedings of the Australasian Conference on Information Security and Privacy, Melbourne, Australia, 4–6 July 2016; Springer: Berlin/Heidelberg, Germany; pp. 181–197.
23. Zhou, L.; Zhu, Y.; Castiglione, A. Efficient k-NN query over encrypted Data in cloud with limited key-disclosure and offline data owner. *Comput. Secur.* **2017**, *69*, 84–96. [CrossRef]
24. Zhu, Y.; Huang, Z.; Takagi, T. Secure and controllable k-NN query over encrypted cloud data with key confidentiality. *J. Parallel Distrib. Comput.* **2016**, *89*, 1–12. [CrossRef]
25. Katz, J.; Lindell, Y. *Introduction to Modern Cryptography*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2014.
26. Agrawal, R.; Kiernan, J.; Srikant, R.; Xu, Y. Order Preserving Encryption for Numeric Data. In Proceedings of the ACM SIGMOD International Conference on Management of data, Paris, France, 13–18 June 2004; pp. 563–574.
27. Boldyreva, A.; Chenette, N.; Lee, Y.; O'neill, A. Order-Preserving Symmetric Encryption. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, 26–30 April 2009; pp. 224–241.
28. Goldreich, O. *Foundations of Cryptography: Volume 1, Basic Tools*; Cambridge University Press: Cambridge, UK, 2007.
29. Leutenegger, S.T.; Lopez, M.A.; Edgington, J. STR: A Simple and Efficient Algorithm for R-tree Packing. In Proceedings of the 13th International Conference on Data Engineering, Birmingham, UK, 7–11 April 1997; pp. 497–506.

30. Hjaltason, G.R.; Samet, H. Distance browsing in spatial databases. *ACM Trans. Database Syst. (TODS)* **1999**, *24*, 265–318. [CrossRef]
31. Henrich, A. A Distance Scan Algorithm for Spatial Access Structures. In Proceedings of the ACM-GIS, Gaithersburg, MD, USA, 1–2 December 1994; pp. 136–143.
32. Wu, Y.; Wang, K.; Zhang, Z.; Lin, W.; Chen, H.; Li, C. Privacy Preserving Group Nearest Neighbor Search. In Proceedings of the EDBT, Vienna, Austria, 26–29 March 2018; pp. 277–288.
33. Kayes, A.; Han, J.; Colman, A.; Islam, M.S. Relboss: A Relationship-Aware Access Control Framework for Software Services. In Proceedings of the OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Rhodes, Greece, 21–25 October 2014; pp. 258–276.
34. Khilar, P.M.; Chaudhari, V.; Swain, R.R. Trust-Based Access Control in Cloud Computing Using Machine Learning. In *Cloud Computing for Geospatial Big Data Analytics*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 55–79.
35. Kayes, A.S.M.; Rahayu, W.; Dillon, T.S.; Chang, E. Accessing Data from Multiple Sources Through Context-Aware Access Control. In Proceedings of the 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, New York, NY, USA, 1–3 August 2018; pp. 551–559.