A kNN Query Processing Algorithm using a Tree Index Structure on the Encrypted Database

Hyeong-Il Kim
Dept. of Computer Engineering
Chonbuk National University
Jeonju, Republic of Korea
melipion@jbnu.ac.kr

Hyeong-Jin Kim
Dept. of Computer Engineering
Chonbuk National University
Jeonju, Republic of Korea
yeon_hui4@jbnu.ac.kr

Jae-Woo Chang*
Dept. of Computer Engineering
Chonbuk National University
Jeonju, Republic of Korea
jwchang@jbnu.ac.kr

***Abstract*— With the adoption of cloud computing, database outsourcing has emerged as a new platform. Due to the serious privacy concerns in the cloud, database need to be encrypted before being outsourced to the cloud. Therefore, various kNN query processing techniques have been proposed over the encrypted database. However, the existing schemes are either insecure or inefficient. So, we, in this paper, propose a new secure kNN query processing algorithm. Our algorithm guarantees the confidentiality of both the encrypted data and a user's query record. To achieve the high query processing efficiency, we also devise an encrypted index search scheme which can perform data filtering without revealing data access patterns. We show from our performance analysis that the proposed scheme outperforms the existing scheme in terms of a query processing cost while preserving data privacy.**

Keywords- Database outsourcing; Database encryption; Encrypted index structure; Data Privacy; kNN query processing;

I. INTRODUCTION

Cloud computing has recently emerged as a new platform for deploying, managing, and provisioning large-scale services through an Internet-based infrastructure. Successful examples include Amazon EC2, Google App Engine, and Microsoft Azure. In the cloud computing, the data owner outsources his/her data to the cloud. Because the data are private assets of the data owner, they should be protected against the cloud. With the adoption of cloud computing, data owners can acquire huge economic benefits by outsourcing their data to the cloud.

Due to the serious privacy concerns in the cloud, sensitive data, such as financial or medical records, should be encrypted before being outsourced to the cloud server. In addition, the queries which are sent to the cloud server might disclose the sensitive information of the users and should be protected against the cloud. Therefore, a vital concern in the cloud computing is to protect both data privacy and query privacy among the data owner, the users, and the cloud.

During query processing, the cloud can derive sensitive information about the actual data items by observing the data access patterns even if the data and the query are encrypted [1]. Using encryption as a way to achieve data confidentiality may cause another issue during the query processing in the cloud. In general, it is very difficult to process encrypted data without having to decrypt it. For this, various techniques related to query processing over encrypted data have been proposed, including range queries [2, 3] and aggregation queries [4, 5]. However, these techniques are not applicable or inefficient to solve advanced queries, such as the secure processing of k-nearest neighbor query (SkNN) in the cloud.

In the past few years, various techniques have been proposed to address the SkNN problem [6-9]. However, the existing methods in [6], [7] are insecure because they are vulnerable to chosen and known plaintext attacks. Another work in [9] returns non-accurate kNN result to the end-user. Furthermore, the end-user involves in heavy computations during the query processing [6, 8, 9]. Additionally, the existing SkNN methods do not protect data access patterns from the cloud. For this, a recent method in [10] has been proposed to guarantee the confidentiality of the encrypted data, the confidentiality of a user's query record and hiding data access patterns. However, the recent work has a disadvantage of high query processing cost.

To solve the problem, we propose, in the paper, a new secure k-NN query processing algorithm. Our algorithm first guarantees the confidentiality of both the encrypted data and a user's query record. Our algorithm also hides data access patterns and provides efficient performance on SkNN. For this, we devise an encrypted index search scheme which can perform data filtering without revealing data access patterns.

Our contributions can be summarized as follows:

- We present a framework for outsourcing both the encrypted database and the encrypted index.
- We design secure protocols (e.g., SBN, SCMP, SPE) to support secure *k*NN query processing.
- We propose an encrypted index search scheme which hides the data access patterns. To the best of our knowledge, this is the first work that searches an index without revealing the data access patterns.
- We propose a new *k*NN query processing algorithm that conceals the data access patterns while supporting efficient query processing.
- We also present an extensive experimental evaluation of our scheme under the various parameter settings.

The rest of the paper is organized as follows. Section 2 introduces the existing kNN query processing algorithms over the encrypted data. Section 3 presents a system architecture and various secure protocols. Section 4 proposes a kd-tree based encrypted index search scheme and a new secure kNN query processing algorithm. Section 5 presents the performance analysis of our secure kNN query processing algorithm. Finally, Section 6 concludes this paper with some future research directions.

II. Related Work

In this section, we first review the existing privacy preserving kNN query processing schemes in outsourced databases. Then, we describe preliminaries for our work.

A. *Privacy preserving kNN query processing schemes*

The typical kNN query processing schemes on encrypted databases are as follows. Wong et al. [7] proposed the ASPE scheme that preserves scalar product between a given query and data. Because a distance comparison is available on the encrypted data, the ASPE scheme can support kNN query processing. However, the ASPE is vulnerable to chosen-plaintext attacks [9] and leaks the data access pattern to the cloud. Yiu et al. [11] proposed the CRT (cryptographic transformation) scheme that supports a kNN query processing by using the R-tree index encrypted by AES [12]. However, the CRT has a drawback that the most of the computation is performed at the user side rather than the cloud. In addition, data access pattern is not preserved as the user hierarchically requests the required R-tree nodes to the cloud. Hu et al. [6] proposed a kNN query processing scheme by using the provably secure privacy homomorphism encryption method [13] which supports modular addition and multiplication over encrypted data. However, the scheme is known to be vulnerable to chosen-plaintext attacks [9]. In addition, because a user has the encrypted index in the scheme, the user is in charge of index traversal during the query processing. Moreover, the scheme leaks the data access pattern as the identifiers of qualified objects are revealed. Zhu et al. [8] proposed a kNN query processing scheme by considering untrusted users. Because a user does not hold an encryption key, a data owner should participate in the query processing step to encrypt the query. In addition, the cloud can be aware of identifiers of the query result which implies the leakage of the data access pattern. Elmehdwi et al. [10] proposed a kNN query processing scheme (SkNN_m) over the encrypted database based on Paillier cryptosystem. To the best of our knowledge, this is the first work that not only guarantees the data privacy and the query privacy, but also conceals the data access pattern at the same time. Because a data owner and a user do not participate in the query processing step, this coincides with the goal of the database outsourcing. Another advantage of this scheme is that it retrieves the exact query result without false-positives. However, the query processing cost of this scheme is very high because it performs computations with all the encrypted data to hide the data access pattern. Most recently, Kim et al. [14] proposed a kNN query processing scheme using the Hilbert-curve order based index. The scheme can preserve the query privacy because a data group generated based on the Hilbert-curve order is considered as a query processing unit. However, a user is in charge of index traversal during the query processing step. In addition, the scheme leaks the data access pattern because the identifiers of the retrieved index are revealed to the cloud. Moreover, the query result may contain false-positives because data groups are returned as the result.

B. *preliminaries*

Paillier crypto system. The Paillier cryptosystem [15] is an additive homomorphic and probabilistic asymmetric encryption scheme for public key cryptography. The public key *pk* for encryption is given by (N, g), where N is a product of two large prime numbers p and q, and g is in $Z^*_{N^2}$. The secret key *sk* for decryption is given by (p, q). Let $E(\ )$ denote the encryption function and $D(\ )$ denote the decryption function. The Paillier crypto system has the following properties. i) Homomorphic addition: The product of two ciphertexts $E(m_1)$ and $E(m_2)$ results in the encryption of the sum of their plaintexts m_1 and m_2 (e.g., $E(m_1+m_2) = E(m_1)*E(m_2) \bmod N^2$). ii) Homomorphic multiplication: The b^{th} power of ciphertext $E(m_1)$ results in the encryption of the product of b and m_1 (e.g., $E(m_1*b) = E(m_1)^b \bmod N^2$). iii) Semantic security: Encrypting the same plaintexts with the same public key results in distinct ciphertexts. So, an adversary cannot infer any information about the plaintexts.

Adversarial models. There are two main types of adversaries, *semi-honest* and *malicious* [16]. In our system, we assume that the clouds act as adversaries. In the *semi-honest* adversarial model, the clouds correctly follow the protocol specification, but try to use the intermediate data to learn more information that are not allowed to them. Meanwhile, in the *malicious* adversarial model, the clouds can arbitrarily deviate from the protocol specification, according to the adversary's instructions. However, protocols against malicious adversaries are too inefficient to be used in practice. However, protocols under the *semi-honest* adversaries are efficient in practice and can be used to design protocols against malicious adversaries. Therefore, by following the work done in [10, 17], we also consider the semi-honest adversarial model in this paper.

III. SYSTEM ARCHITECTURE AND SECURE PROTOCOLS

In this section, we first describe the motivation of the paper and explain the system architecture of the proposed scheme. Next, we present generic secure protocols used for our kNN query processing algorithm.

A. *Motivation*

Our key motivation is that there is no work that not only preserves data privacy and query privacy, but also conceals data access pattern, while guaranteeing the efficient query processing. In particular, the only work [10] that preserves the data access pattern suffers from high query processing cost because it should consider all the data to process a kNN query. Especially, they compare the distances of all the data to find the nearest neighbor from a query and the algorithm repeats k times to find kNN result.

To solve the problem, we first design an encrypted index search scheme to filter out the irrelevant data to the query while hiding the data access patterns. By extracting the data relevant to the query, we can greatly enhance the efficiency

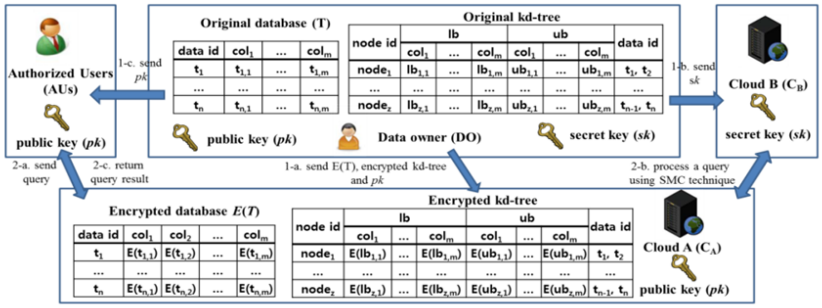


Figure 1. The overall system architecture

of the query processing. We also propose the kNN query processing algorithm including a query verification step to guarantee the accuracy of a query result. Consequently, our algorithm can provide efficient query processing while satisfying the above three privacy requirements.

B. System Architecture

Figure 1 shows the overall system architecture for our query processing scheme on the encrypted database. The system consists of data owner (*DO*), authorized user (*AU*), and two clouds (C_A and C_B). The *DO* owns the original database (T) of n records. A record t_i ($1 \leq i \leq n$) consists of m attributes and j^{th} attribute value of t_i is denoted by $t_{i,j}$. To provide the indexing on T, the *DO* partitions T by using kd-tree. If we retrieve the tree structure in hierarchical manner, the access pattern can be disclosed. So, we only consider the leaf nodes of the kd-tree and all the leaf nodes are retrieved once during the query processing. Let h denote the level of the constructed kd-tree and F be the fanout of each leaf node. The total number of leaf nodes is 2^{h-1}. From now on, a node means a leaf node. The region information of each node is represented as the lower bound $lb_{z,j}$ and the upper bound $ub_{z,j}$ ($1 \leq z \leq 2^{h-1}$, $1 \leq j \leq m$). Each node stores the identifiers (*id*) of data being located inside the node region.

To preserve the data privacy, the *DO* encrypts T attribute-wise using the public key (*pk*) of the Paillier cryptosystem [15] before outsourcing the database. So, the *DO* generates $E(t_{i,j})$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. The *DO* also encrypts the region information of all the kd-tree nodes to support efficient query processing. The *lb* and the *ub* of each node are encrypted attribute-wise, so $E(lb_{z,j})$ and $E(ub_{z,j})$ are generated, for $1 \leq z \leq 2^{h-1}$ and $1 \leq j \leq m$.

We assume that C_A and C_B are non-colluding and semi-honest (or honest-but-curious) clouds. So, they correctly perform the given protocols, but an adversary may try to obtain additional information from the intermediate data during executing his/her own protocol. This assumption is not new as mentioned in [10, 17] and has been used in the related problem domains (e.g., [18]). Especially, because most of the cloud services are provided by renowned IT companies, such as Amazon and Google, collusion between them which will damage their reputation is improbable [10].

To support kNN query processing over the encrypted database, a secure multiparty computation (SMC) is required between C_A and C_B. For this, the *DO* outsources the encrypted database and its encrypted index to the C_A with *pk* but sends the *sk* to the different cloud, C_B. The encrypted index includes the region information of each node in cipher-text and the *ids* of data that are located inside the node in plain-text. The *DO* also sends *pk* to *AU*s to enable them to encrypt a query. At query time, an *AU* first encrypts a query attribute-wise. The encrypted query is denoted by $E(q_j)$ for $1 \leq j \leq m$. C_A processes the query with the help of C_B, and returns a query result to the *AU*.

As an example, assume that an *AU* has 8 data in two-dimensional space (e.g., x-axis and y-axis) as depicted in Figure 2. The data are partitioned into 4 nodes (e.g., $node_1$~$node_4$) for a kd-tree. To clarify the relationship between data and nodes, in this example we suppose that there is no data on the boundary of a node. To outsource the database, the *DO* encrypts each data and region of each node attribute-wise. For example, t_1 is encrypted as $E(t_1) = \{E(2), E(1)\}$and the encrypted index is shown in Table 1.

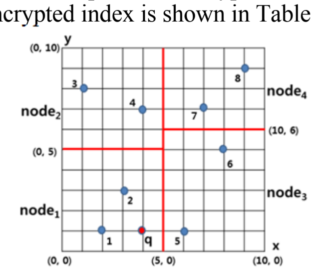


Figure 2. An example in two-dimensional space

TABLE I. AN EXAMPLE OF AN ENCRYPTED INDEX

Node id	*lb* (lower bound)		*ub* (upper bound)		Data id
	x	*y*	*x*	*y*	
$node_1$	E(0)	E(0)	E(5)	E(5)	1, 2
$node_2$	E(0)	E(5)	E(5)	E(10)	3, 4
$node_3$	E(5)	E(0)	E(10)	E(6)	5, 6
$node_4$	E(5)	E(6)	E(10)	E(10)	7, 8

C. Secure Protocols

Our encrypted index search scheme and kNN query processing algorithm are constructed using several secure protocols. In this section, all the protocols except SBN protocol are performed through the SMC technique between C_A and C_B. SBN protocol can be executed by C_A alone. Due to the space limitation, we first briefly introduce six secure protocols that we adopt from the literatures [10, 19]. SM (Secure Multiplication) protocol computes the encryption of $a \times b$, i.e., $E(a \times b)$, when two encrypted data $E(a)$ and $E(b)$ are given as inputs. SSED (Secure Squared Euclidean Distance) protocol computes $E(|X\text{-}Y|^2)$ when two encrypted vectors $E(X)$ and $E(Y)$ are given as inputs. Here, X and Y consist of m attributes. SBD (Secure Bit-Decomposition) protocol computes the encryptions of binary representation of the encrypted input $E(a)$. The output is $[a] = \langle E(a_1), \ldots, E(a_l)\rangle$ where a_1 and a_l denote the most and least significant bits of a, respectively. In this section, we use symbol $[a]$ to denote the encryptions of binary representation. SMIN (Secure Minimum) protocol returns the $[min]$ between two inputs $[u]$ and $[v]$. SMIN_n protocol returns $[min]$ among $[d_i]$ for $1 \leq i \leq n$. SMIN_n protocols finds a result by performing SMIN protocol n-1 times. SBOR (Secure Bit-OR) protocol performs the bit-or operation when two encrypted bits are given as inputs. Next, we propose our new secure protocols.

SBN protocol. SBN (Secure Bit-Not) protocol performs the bit-not operation when an encrypted bit $E(a)$ is given as input. The output of SBN $E(\sim a)$ is computed by $E(a)^{N-1}\times E(1)$. Note that "-1" is equivalent to "N-1" under Z_N.

SCMP protocol. When $[u]$ and $[v]$ are given as inputs, SCMP (Secure Compare) protocol returns $E(1)$ if $u \leq v$, $E(0)$ otherwise. We devise SCMP by modifying SMIN. The variables generated during SMIN can be categorized into two folds. One set of the variables include hints about what minimum value is. Another set of the variables are used to securely extract the minimum value. Because we need the information about whether u is smaller or not, we only compute the former (e.g., W, G, H, Φ, L, L'). The goal of designing SCMP is to make the returned value from C_B be exactly opposite for the same inputs, based on the functionality selected by C_A. SMIN can achieve this goal when two inputs have different values. However, if two values are the same, SMIN fails to attain the goal.

To solve this problem, we design SCMP as follows. Algorithm 1 shows the overall procedure of SCMP. First, C_A appends $E(0)$ to the least significant bits of $[u]$ and $E(1)$ to the least significant bits of $[v]$ (line 1). This makes $u \leftarrow u\times 2$ and $v \leftarrow v\times 2+1$. By doing so, SCMP makes u smaller than v only when two values are the same. This can bring two advantages. SCMP can solve the security problem of SMIN that C_B can notice whether the input values are same or not. In SMIN, only if two inputs have the same values, $D(L')$ does not include 1 or 0. In addition, by taking $u<v$ for inputs with the same values, we can take advantage of SMIN.

Secondly, C_A randomly chooses one functionality between $F_0{:}u{>}v$ and $F_1{:}v{>}u$. The selected functionality is oblivious to C_B (line 2). Then, C_A computes $E(u_i\times v_i)$ using SM and W_i, depending on the selected functionality (line 3~6). In particular, if $F_0{:}u{>}v$ is selected, C_A computes $W_i{=}E(u_i)\times E(u_i\times v_i)^{N-1}{=}E(u_i\times(1\text{-}v_i))$. If $F_1{:}v{>}u$ is selected, C_A computes $W_i{=}E(v_i)\times E(v_i\times u_i)^{N-1}{=}E(v_i\times(1\text{-}u_i))$. For $F_0{:}u{>}v$, $W_i{=}E(1)$ when $u_i{>}v_i$, and $W_i{=}E(0)$ otherwise. Similarly, for $F_1{:}v{>}u$, $W_i{=}E(1)$ when $v_i{>}u_i$, and $W_i{=}E(0)$ otherwise. Thirdly, C_A performs bit-xor between $E(u_i)$ and $E(v_i)$ and stores the result into G_i (line 7). C_A computes $H_i{=}(H_{i-1})^{ri}\times G_i$ and $\Phi_i{=}E(\text{-}1)\times H_i$ where $H_0{=}E(0)$ (line 8). Here, r_i is a random number in Z_N. When j is the index of the first appearance of $E(1)$ in G_i, j means the first position where the minimum value between u and v can be determined. Fourthly, C_A computes $L_i{=}W_i\times\Phi_i^{ri}$ (line 9) where L_i involves the information about which value is smaller between u and v at j. C_A generates L' by permuting L using a random permutation function π_1 and sends L' to C_B (line 10). Fifthly, C_B decrypts L' attribute-wise and checks whether there exists 0 in L_i' for $1 \leq i \leq l$. If so, C_B sets α as 1, and 0 otherwise. After encrypting α, C_B sends $E(\alpha)$ to C_A (line 11~12). Table 2 shows the values of α returned by C_B. The returned values are exactly opposite with the selected functionalities for every case, which coincides with the goal of SCMP.

Algorithm 1: SCMP

Input : [u], [v]
Output : $E(1)$ when $[u] \leq [v]$, $E(0)$ otherwise
C_A : **01:** Append $E(0)$ to []u;   Append $E(1)$ to [v]
02: Randomly choose F_0 or F_1
03: for $1 \leq i \leq l$
04: $E(u_i\times v_i) \leftarrow SM(E(u_i), E(v_i))$
05: if $F_0 : u > v$ is chosen then $W_i \leftarrow E(u_i)\times E(u_i\times v_i)^{N-1}$
06: else $W_i \leftarrow E(v_i)\times E(v_i\times u_i)^{N-1}$
07: $G_i \leftarrow E(u_i)\times E(v_i)\times E(u_i\times v_i)^{N-2}$ // XOR
08: $H_i \leftarrow (H_{i-1})^r\times G_i$ and $H_0 \leftarrow E(0)$; $\Phi_i \leftarrow E(-1)\times H_i$
09: $L_i \leftarrow W_i\times\Phi_i^{ri}$
10: $L' \leftarrow \pi_1(L)$;   send L' to C_B
C_B : **11:** if 0 exists in $D(L')$ then $\alpha \leftarrow 0$ otherwise $\alpha \leftarrow 1$
12: Send $E(\alpha)$ to C_A
C_A : **13:** if $F_0 : u > v$ then $E(\alpha) \leftarrow SBN(E(\alpha))$
14: return $E(\alpha)$

TABLE II. VALUE OF $E(\alpha)$ RETUREND BY C_B

		Selected functionality	
		F_0 : u > v	**F_1 : v > u**
Actual relationship between u and v	**u > v**	E(1)	E(0)
	v > u	E(0)	E(1)
	u = v	E(0)	E(1)

Finally, C_A performs $E(\alpha)$=SBN($E(\alpha)$) only when the selected functionality is $F_0{:}u{>}v$ and returns the $E(\alpha)$ (line 13~14). So, the final $E(\alpha)$ is $E(1)$ when $u \leq v$, regardless of the selected functionality. Note that the only information decrypted during SCMP is L' which is seen by C_B. However, C_B cannot obtain an additional information from $D(L')$ because the selected functionality is oblivious to C_B. Therefore, SCMP is secure under the semi-honest model.

SPE protocol. When the encryptions of binary representation of a point $[p]$ as well as region information $[lb]$ and $[ub]$ are given as inputs, SPE (Secure Point Enclosure) protocol returns $E(1)$ when p is inside the region or on a boundary of the region, $E(0)$ otherwise. Assuming that p, lb and ub consist of m attributes, a point p is inside the region only if two following conditions are satisfied; i) $p_i \leq E(ub_i)$ for $1 \leq i \leq m$ and ii) $E(lb_i) \leq p_i$ for $1 \leq i \leq m$. SPE determines the conditions by using our SCMP.

Algorithm 2 shows the overall procedure of SPE. C_A initializes $E(\alpha)$ as $E(1)$ (line 1). C_A obtains $E(\alpha')$ by performing SCMP($[p_i]$, $[ub_i]$) and updates $E(\alpha)$ by executing SM($E(\alpha)$, $E(\alpha')$). C_A repeats this step for $1 \leq i \leq m$ (line 2~3). Similarly, using $[p_i]$ and $[lb_i]$, C_A computes $E(\alpha')$ by performing SCMP([lb_i], [p_i]) and updates $E(\alpha)$ by executing SM($E(\alpha)$, $E(\alpha')$) for all attribute values (line 4~5). Only when all conditions are satisfied, the value of $E(\alpha)$ is $E(1)$. Finally, C_B returns the final $E(\alpha)$. Note that no decryption is performed during SPE except performing SCMP and SM protocols. SPE is secure under the semi-honest model because the securities of both SCMP and SM are verified.

Algorithm 2: SPE

Input : [p], [region] = ([lb], [ub])
Output : E(1) when *p* is inside the region or on a boundary of the region
$\mathbf{C_A}$: **01:** $E(\alpha) \leftarrow E(1)$
02: for $1 \le i \le m$
03: $E(\alpha') \leftarrow$ SCMP([p_i], [ub_i]) then $E(\alpha) \leftarrow$ SM($E(\alpha)$, $E(\alpha')$)
04: for $1 \le i \le m$
05: $E(\alpha') \leftarrow$ SCMP([lb_i], [p]) then $E(\alpha) \leftarrow$ SM($E(\alpha)$, $E(\alpha')$)
06: return $E(\alpha)$

IV. kNN Query Processing Algorithm

In this section, we present our kNN query processing algorithm. Our kNN query processing algorithm consists of three steps; encrypted kd-tree search step, kNN retrieval step, and result verification step.

A. Step 1 : Encrypted kd-tree search step

The S*k*NN scheme that hides the data access pattern [10] requires high computation cost because it performs encryption-based operations for all data. To tackle the problem, we design the encrypted index search scheme. Our scheme does not reveal the retrieved nodes of the index for query processing while extracting data in the nodes which the query is located in. In this paper, we consider *k*d-tree as an index structure because it is suitable for indexing multi-dimensional data. However, we emphasize that our index search scheme can be applied to any other indices whose entities (e.g., nodes) stores region information.

The procedure of the encrypted *k*d-tree search step is shown in Algorithm 3. First, C_A computes [q_j] for $1 \le j \le m$ by using SBD (line 1~2). C_A also computes {([$node_z.lb_j$], [$node_z.ub_j$] | $1 \le z \le num_{node}$, $1 \le j \le m$} by using SBD where num_{node} is the total number of *k*d-tree leaf nodes (line 3~6). Then, C_A securely finds the node relevant to the query by executing $E(\alpha_z) \leftarrow$ SPE([q], [$node_z$]) for $1 \le z \le num_{node}$ (line 7). Note that the nodes with $E(\alpha_z)=E(1)$ contains the query, but both C_A and C_B cannot know whether the value of each $E(\alpha_z)$ is $E(1)$. This is because Paillier encryption provides a semantic security property. Secondly, C_A generates $E(\alpha')$ by permuting $E(\alpha)$ using a random permutation function π and sends $E(\alpha')$ to C_B (line 8). For example, SPE returns $E(\alpha)$={$E(1)$, $E(0)$, $E(0)$, $E(0)$} in Figure 2 as the q is located inside the $node_1$. Assuming that π permutes data in reverse way, C_A sends the $E(\alpha')$={$E(0)$, $E(0)$, $E(0)$, $E(1)$} to C_B.

Thirdly, upon receiving the $E(\alpha')$, C_B obtains α' by decrypting the $E(\alpha')$ and counts the number of α'=1 and stores it into *c*. Here, *c* means the number of nodes that the query is related to (line 9). Fourthly, C_B creates *c* number of node groups (e.g., *NG*). C_B assigns to each *NG* a node with α'=1 and num_{node}/c-1 nodes with α'=0. Then, C_B computes *NG′* by randomly shuffling the *ids* of nodes in each *NG* and sends *NG′* to C_A (line 10~14). For example, C_B can know

Algorithm 3: Encrypted kd-tree search

Input : E(q), E(node)
Output : E(cand) // all the data inside nodes related to a query
$\mathbf{C_A}$: **01:** for $1 \le j \le m$
02: [q_j] $\leftarrow$ SBD(E(q_j))
03: for $1 \le z \le num_{node}$ // $num_{node} = 2^{h-1}$ (h : level of the kd-tree)
04: for $1 \le j \le m$
05: [$node_z.lb_j$] $\leftarrow$ SBD(E($node_z.lb_j$))
06: [$node_z.ub_j$] $\leftarrow$ SBD(E($node_z.ub_j$))
07: $E(\alpha) \leftarrow$ SPE([q], [$node_z$])
08: $E(\alpha') \leftarrow \pi(E(\alpha))$; send $E(\alpha')$ to C_B
$\mathbf{C_B}$: **09:** $\alpha' \leftarrow D(E(\alpha'))$; c $\leftarrow$ the number of ′1′ in α'
10: create c number of Group
11: for each *NG*
12: assign a node with α'=1 and num_{node}/c–1 nodes with α'=0
13: *NG′* $\leftarrow$ shuffle the *ids* of nodes
14: send *NG′* to CA
$\mathbf{C_A}$: **15:** *cnt* $\leftarrow$ 0
16: NG^* $\leftarrow$ permute node *ids* using π^{-1} for each *NG′*
17: for each NG^*
18: for $1 \le s \le F$
19: for $1 \le i \le num$ (# nodes in the selected NG^*)
20: $E(t'_{i,j}) \leftarrow$ SM($node_i.t_{s,j}$, $E(\alpha_z)$) for $1 \le j \le m$
21: for $1 \le j \le m$
22: $E(cand_{cnt,j}) \leftarrow \prod_{i=1}^{num} E(t'_{i,j})$
23: *cnt* $\leftarrow$ *cnt*+1
24: send E(*cand*)

that only $node_4$ is related to the query because α'={0, 0, 0, 1} contains one at the fourth position. However, C_B cannot correctly point out the node containing the query because the values in α' were permutated by C_A. As one node group is required, C_B assigns all nodes to a single node group and randomly shuffles the *ids* of the nodes like NG_1'={2, 1, 3, 4}.

Fifthly, C_A obtains NG^* by permuting the *ids* of nodes using π^{-1} in each *NG′* (line 16). In each NG^*, there exists only one node corresponding to the query, but C_A cannot know that node because the *ids* of the nodes in NG^* are shuffled by C_B. Sixthly, C_A gets access to one datum in each node (e.g., $node_z$) for each NG^* and performs $E(t'_{i,j}) \leftarrow$ SM($node_z.t_{s,j}$, $E(\alpha_z)$) for $1 \le s \le F$ and $1 \le j \le m$ *where* α_z is the returned values corresponding to the $node_z$ from SPE (line 17~20). So, the data in the nodes corresponding to the query is not affected by SM because the $E(\alpha_z)$ values of the nodes are $E(1)$. However, the data in other nodes become $E(0)$ by performing SM because the $E(\alpha_z)$ values of the nodes are $E(0)$. If a node has the less number of data than *F*, it performs SM by using $E(max)$, instead of using $node_z.t_{s,j}$, where $E(max)$ is the largest value in the domain. If the node is not related to the query, the result of SM becomes $E(0)$. If the node is related to the query, the result of SM becomes $E(max)$, which is not the real value in the database. Note that the data is safely pruned at the later query processing step.

When C_A accesses one datum from every node in a NG^*, C_A performs $E(cand_{cnt,j}) \leftarrow \prod_{i=1}^{num} E(t'_{i,j})$ where *num* means the total number of nodes in the selected NG^* (line 21~22). By doing so, a datum in the node related to the query is securely extracted without revealing the data access patterns. By repeating these steps, all the data in the nodes are safely stored into the $E(cand_{cnt,j})$. Finally, *cnt* means the total number of data extracted during the index search. As an example, C_A obtains NG_1^*={3, 4, 2, 1} by permuting the

NG_1'={2, 1, 3, 4} by using π^{-1}. C_A accesses $E(t_5)$ in $node_3$, $E(t_7)$ in $node_4$, $E(t_3)$ in $node_2$, and $E(t_1)$ in $node_1$. The results of SM using $E(t_5)$, $E(t_7)$, and $E(t_3)$, e.g., $E(t'_1)$, $E(t'_2)$ and $E(t'_3)$, are $E(0)$ for every attribute because the $E(\alpha)$ values of the corresponding nodes are $E(0)$. However, the results of SM using $E(t_1)$, e.g., $E(t'_4)$, become $E(2)$ and $E(1)$ for x and y dimension, respectively. So, the results of the attribute-wise homomorphic addition of $E(t'_1)$, $E(t'_2)$, $E(t'_3)$, and $E(t'_4)$ are $E(2)$ and $E(1)$ for x and y dimension, respectively. Thus, one datum $E(t_1)$ in $node_1$ is securely extracted. Similarly, the encrypted kd-tree search step can extract all the data stored in $node_1$ (e.g., $E(t_1)$ and $E(t_2)$).

B. Step 2 : kNN retrieval step

In kNN retrieval step, we retrieve k closest data from the query by executing the $SkNN_m$ scheme [10]. However, while the original $SkNN_m$ considers all the encrypted data, we only consider $E(cand_i)$ for $1 \leq i \leq cnt$ returned from the step 1. Due to the space limitation, we briefly explain the kNN retrieval step with an example.

Using SSED, C_A calculates the squared Euclidean distances $E(d_i)$ between a query and $E(cand_i)$ for $1 \leq i \leq cnt$ and computes $[d_i]$ of $E(d_i)$ using SBD. Then, C_A performs $SMIN_n$ to find the minimum value $[d_{min}]$ among $[d_i]$ where $1 \leq i \leq cnt$. Secondly, C_A converts $[d_{min}]$ into $E(d_{min})$ based on the homomorphic encryption property and calculates τ_i, i.e., the difference between the $E(d_{min})$ and $E(d_i)$ for $1 \leq i \leq cnt$. C_A generates τ'_i by multiplying a random value and τ_i. Note that only the τ_i corresponding to the $E(d_{min})$ has a value of $E(0)$. C_A obtains β by shuffling τ_i using a random permutation function π and sends β to the C_B. Thirdly, C_B sets U_i=$E(1)$ by decrypting β if $D(\beta_i)$=0, and sets U_i=$E(0)$ otherwise. After C_B sends U to C_A, C_A obtains V by permuting U using π^{-1}. By computing $E(t'_{s,j}) \leftarrow \prod_{i=1}^{cnt} E(V_i, E(cand_{i,j}))$ for $1 \leq j \leq m$, C_A can securely extract the datum corresponding to the $E(d_{min})$. To prevent the result from being selected in later phase, C_A securely updates the distance of the selected result as $E(max)$ by using SBD. This procedure is repeated for k rounds to find the kNN result. For example, in the first round, $E(t_1)$ with distance $E(4)$ is securely selected as the NN result among $cand$={$E(t_1)$ and $E(t_2)$}. As the distance of $E(t_1)$ is updated into $E(max)$, $E(t_2)$ is selected as the 2NN result in the second round.

C. Step 3 : Result verification step

The result of the step 2 is not accurate because the query is processed on the partial data that are extracted in the step 1. Therefore, it is required to verify if the current query result is correct or not. Assuming that the squared Euclidean distance between the k^{th} closest result $E(t'_k)$ and the query is $dist_k$, the kd-tree nodes located within $dist_k$ need to be searched. For this, we define the shortest point of a node as follows.

Definition 1 (**shortest point**) A shortest point (*sp*) is a point in a given node whose distance is shortest to a given query point p than the other points in the node.

Assume that there are three regions and one point in 1-dimensional space like Figure 3. The *sp* is determined according to the relationship between a region and p. i) If both the lower bound (*lb*) and the upper bound (*ub*) of the region are lesser than p, the *ub* becomes the *sp* of the region (e.g., $region_1$). ii) If both the *lb* and the *ub* of the region are greater than p, the *lb* becomes the *sp* of the region (e.g., $region_2$). iii) If p is between the *lb* and the *ub* of the region, p is the *sp* of the region (e.g., $region_3$).

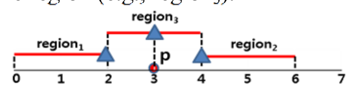


Figure 3. An example of the shortest point

We expand this property to the multi-dimensional space to find kd-tree nodes for the query result verification. The procedure of the result verification step is shown in Algorithm 4. First, C_A initializes $[ndist_z]$ for $1 \leq z \leq num_{node}$ by storing $E(\alpha_z)$, the result of SPE in step 1, in each bit (line 1~2). So, $[ndist_z]$ of the nodes related to the query have $E(1)$ for all bits, while $[ndist_z]$ of the other nodes have $E(0)$ for all bits. C_A also computes the squared Euclidean distance (e.g., $dist_k$) between the k^{th} closest result $E(t'_k)$ and the query and computes $[dist_k]$ using SBD (line 3). Second, for each $node_z$, C_A performs SCMP by using $[q_j]$ and $[node_z.lb_j]$ for $1 \leq j \leq m$ and stores the result in ψ_1. In addition, C_A performs SCMP using $[q_j]$ and $[node_z.ub_j]$ for $1 \leq j \leq m$ and stores the result in ψ_2. When the value of $[q_i]$ is equal to or less than the *lb* (*ub*), the ψ_1 (ψ_2) has the value of $E(1)$. Then, C_A obtains ψ_3 by carrying out a bit-xor operation using ψ_1 and ψ_2 (line 4~8). Thirdly, C_A securely obtains the $sp_{z,j}$ for each node by the equation (1).

$$sp_{z,j} = \psi_3 \times E(q_j) + (1-\psi_3) \times (\psi_1 \times E(node_z.lb_j) + (1-\psi_1) \times E(node_z.ub_j)) \quad (1)$$

Algorithm 4: Result verification

Input : E(q), [q], E(node), [node], E(t´), k
Output : result
C_A : 01: for $1 \leq z \leq num_{node}$
02: $[ndist_z] \leftarrow E(\alpha_z)$ for all bits // $E(\alpha_z)$: output of SPE in step 1
03: $dist_k$ = SSED(E(q), E(t'_k)); $[dist_k]$ = SBD($dist_k$)
04: for $1 \leq z \leq num_{node}$
05: for $1 \leq j \leq m$
06: $\psi_1 \leftarrow$ SCMP($[q_i]$, $[node_z.lb_i]$)
07: $\psi_2 \leftarrow$ SCMP($[q_i]$, $[node_z.ub_i]$)
08: $\psi_3 \leftarrow$ XOR(ψ_1, ψ_2)
09: E(temp) $\leftarrow$ SM(ψ_1, E($node_z.lb_i$))
10: E(temp) $\leftarrow$ E(temp) × SM(SBN(ψ_1), E($node_z.ub_i$))
11: E(temp) $\leftarrow$ SM(E(temp), SBN(ψ_3))
12: $sp_{z,i} \leftarrow$ E(temp) × SM(SBN(ψ_3), E(q))
13: $spdist_z \leftarrow$ SSED(E(q), $sp_{z,i}$)
14: $[ndist_z] \leftarrow$ SBOR(SBD($spdist_z$), $[ndist_z]$)
15: $E(\alpha_z) \leftarrow$ SCMP($[ndist_z]$, $[dist_k]$)
16: E(t´´) $\leftarrow$ perform 8~24 lines of algorithm 3
17: E(t´) $\leftarrow$ append E(t´´) to E(t´)
18: E(result) $\leftarrow$ perform $skNN_m$ algorithm (step 2)
19: for $1 \leq i \leq k$ and for $1 \leq j \leq m$
20: $E(\gamma_{i,j}) \leftarrow E(result_{i,j}) * E(r_{i,j})$
21: send $E(\gamma_{i,j})$ to C_B and $r_{i,j}$ to AU
C_A : 22: for $1 \leq i \leq k$ and for $1 \leq j \leq m$
23: $\gamma_{i,j} \leftarrow D(E(\gamma_{i,j}))$; send $\gamma_{i,j}$ to AU
AU: 24: for $1 \leq i \leq k$ and for $1 \leq j \leq m$
25: $result_{i,j} = \gamma_{i,j} - r_{i,j}$

In the equation, (1-ψ_1) and (1-ψ_3) can be executed by SBN and a multiplication can be performed by SM (line 9~12). Fourthly, C_A calculates $spdist_z$, the squared Euclidean distances between the query and sp_z for $1 \le z \le num_{node}$ (line 13). Fifthly, C_A updates [$ndist_z$] by performing SBOR using encryptions of binary representation of $spdist_z$ (e.g., SBD($spdist_z$)) and [$ndist_z$] (line 14). So, [$ndist_z$] of the nodes related to the query have $E(1)$ for all bits, while [$ndist_z$] of the other nodes becomes SBD($spdist_z$). Then, C_A performs SCMP by using [$ndist_z$] and [$dist_k$] (line 15). If the $ndist_z$ is less than $dist_k$, the corresponding $node_z$ are assigned $E(\alpha)=E(1)$. The nodes with $E(\alpha)=E(1)$ need to be retrieved for query result verification. Sixthly, C_A securely extracts the data stored in the nodes with $E(\alpha)=E(1)$, by performing the 9~24 lines of the algorithm 3. C_A appends the extracted data $E(t'')$ to the $E(t')$. Then, C_A executes the skNN$_m$ algorithm with the $E(t')$ and obtains $E(result_i)$ for $1 \le i \le k$ (line 16~18). Eighthly, C_A computes $E(\gamma_{i,j})=E(result_i)\times E(r_{i,j})$ for $1 \le i \le k$ and $1 \le j \le m$ by generating a random value $r_{i,j}$. Then, C_A sends $E(\gamma_{i,j})$ to C_B and $r_{i,j}$ to AU, respectively (line 19~21). Ninthly, C_B decrypts $E(\gamma_{i,j})$ and sends the decrypted value to AU (line 22~23). Finally, AU computes the actual kNN result by computing $\gamma_{i,j}$ - $r_{i,j}$ in a plaintext (line 24~25).

V. Performance Analysis

A. Performance Environment

In this section, we compare our SkNN$_I$ (secure kNN query processing scheme with the secure index) with the SkNN$_m$ [10] which is known to the only work to hide data access patterns. We do the performance analysis of both schemes in terms of query processing time with different parameters. We used the Paillier cryptosystem to encrypt a database for both schemes. We implemented both schemes by using C++. Experiments were performed on a Linux machine with an Intel Xeon E3-1220v3 4-Core 3.10GHz and 32GB RAM running Ubuntu 14.04.2. To examine the performance under various parameters, we randomly generated synthetic datasets by following [10, 17]. We used the parameters shown in Table 3.

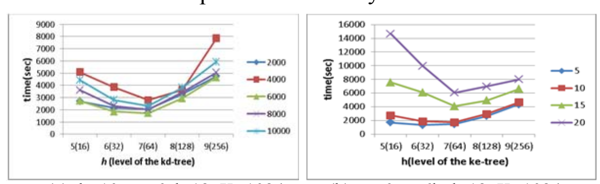
Table 3. Experimental parameters

Parameters	Values	Default value
Total number of data (n)	2k, 4k, 6k, 8k, 10k	6k
Level of kd-tree (h)	5, 6, 7, 8, 9	7
Required k (k)	5, 10, 15, 20	10
# of attributes (m)	3, 6, 9	6
Domain size (l)	9, 12, 15, 18	12
Encryption key size (K)	1024	1024

B. Performance of SkNN$_I$ with Varying the Level of kd-tree

In Figure 4, we measure the performance of our SkNN$_I$ for varying the level of kd-tree because SkNN$_m$ does not use the kd-tree. The performance of SkNN$_I$ by varying h and n is shown in Figure 4(a). Regardless of n, the query processing time is decreased as h changes from 5 to 7 while the query processing time increase as h changes from 7 to 9. This result comes from the following properties. The total number of leaf nodes grows as h increases. So, as h increases, the more computation cost is required for SPE to find a node corresponding to the query. However, the number of data in a node decreases as h increases. So, as h increases, the less computation cost is required when calculating the distance among the query and data.

The performance of SkNN$_I$ by varying h and k is shown in Figure 4(b). Regardless of k, the query processing time is decreased as h changes from 5 to 7 while the query processing time increases as h changes from 7 to 9. This shows the similar trend with Figure 4(a) due to the same reason we mentioned. In addition, the query processing time increases as k increases. However, the query processing time does not increase much as k changes from 5 to 10. This is because a node expansion is hardly done for the smaller k, in the result verification step, since the node related to the query includes the enough number of data. However, when k changes from 15 to 20, the query processing time is relatively great, especially for h=5. The reason is that a node expansion is required for the larger k and so it takes more time to retrieve the k nearest neighbors in multiple nodes. When varying the level of h, the overall performance of SkNN$_I$ depends on the kd-tree level. When h=7, the best performance is achieved for the most of the cases and thus we use h=7 for our performance analysis.



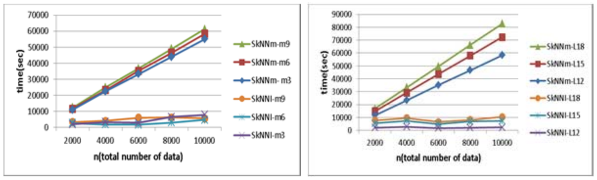
(a) k=10, m=6, l=12, K=1024 (b) m=6, n=6k, l=12, K=1024

Figure 4: Query processing time for varying h

C. Comparison of SkNN$_I$ and SkNN$_m$

Figure 5 shows the performance of SkNN$_I$ and SkNN$_m$ for varying the n. The performance of both schemes by varying n and m is shown in Figure 5(a). As the n becomes larger, the query processing time of SkNN$_m$ linearly increases for all cases because it needs to perform secure protocols using all the data. However, when the m increases, the query processing time slightly increases. This is because only the SSED is affected by m while the other secure protocols including the most time-consuming SMIN$_n$ are not affected by the m. In case of SkNN$_I$, the query processing time is not much affected by the m due to the same reason with SkNN$_m$. However, as n increases, the query processing time does not show the distinct pattern because we fix the h as 7. This coincides with the performance shown in Figure 4.

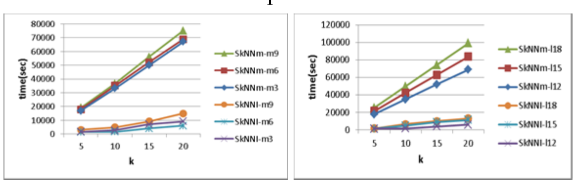
The performance of both schemes by varying n and l is shown in Figure 5(b). As the n becomes larger, the query processing time of SkNN$_m$ linearly increases for all cases because it needs to perform secure protocols using all the data. The query processing time of SkNN$_m$ also increases as the l increases. However, the change is slightly bigger than that of the Figure 5(a). This is because SBD, SBOR, and SMIN$_n$ are affected by l. Meanwhile, our SkNN$_I$ shows almost same performance with the performance shown in Figure 5(a) because the h is fixed in this performance. Overall, our SkNN$_I$ shows about 7 times better performance than SkNN$_m$.



(a) k=10, l=12 (b) k=10, m=6

Figure 5: Query processing time for varying n

Figure 6 shows the performance of $SkNN_I$ and $SkNN_m$ for varying the *k*. The performance of both schemes by varying *k* and *m* is shown in Figure 6(a) while the performance by varying *k* and *l* is shown in Figure 6(b). As the *k* becomes larger, the query processing time of both schemes linearly increase for all cases because the schemes should perform $SMIN_n$ *k* times. Meanwhile, when the *m* increases, the query processing time of both schemes slightly increases. This is because only the SSED is affected by *m*. The query processing time of $SkNN_m$ also increases as the *l* increases. However, the change is slightly bigger than that of the Figure 6(a). This is because SBD, SBOR, and $SMIN_n$ are affected by *l*. Meanwhile, our $SkNN_I$ shows almost same performance with the performance shown in Figure 6(a) because the h is fixed in this performance.



(a) k=10, l=12 (b) k=10, m=6

Figure 6: Query processing time for varying k

VI. Conclusion

Database outsourcing has emerged as a new platform in cloud computing. Due to the privacy concerns, database needs to be encrypted before being outsourced to the cloud. So, various kNN query processing techniques have been proposed over the encrypted database. However, most of the existing schemes disclose data access patterns during the query processing. This problem is critical because the cloud can derive sensitive information about the actual data items by observing the data access patterns even if the data and the query are encrypted. The only work that hides the data access patterns has a disadvantage of high query processing cost [10]. So, we proposed a new secure k-NN query processing algorithm which guarantees the confidentiality of both the encrypted data and the user's query record. By devising an encrypted index search scheme which supports data filtering without revealing data access patterns, our method can provide efficient query processing. We showed from our performance analysis that our algorithm outperformed the existing one in terms of a query processing cost, while preserving data privacy.

As a future work, we will improve the efficiency of our method by processing a query in parallel. We also plan to expand our work to support other query types, such as Top-k and skyline queries.

Acknowledgments

This work was supported by the Human Resource Training Program for Regional Innovation and Creativity through the Ministry of Education and National Research Foundation of Korea (NRF-2014H1C1A1065816). This research was also supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number 2013R1A1A4A01010099).

References

[1] S. De Capitani di Vimercati, S. Foresti, and P. Samarati, "Managing and accessing data in the cloud: Privacy risks and approaches", in CRiSIS, 2012, pp. 1-9.

[2] E. Shi, J. Bethencourt, T. H. Chan, D. Song, and A. Perrig, Multidimensional range query over encrypted data, in Symposium on Security and Privacy, 2007, pp. 350-364.

[3] B. Hore, S. Mehrotra, M. Canim, M. Kantarcioglu, Secure multidimensional range queries over outsourced data, VLDB Journal, vol. 21, issue 3, June 2012, pp. 333-358.

[4] H. Hacıgumus, B. Iyer, and S. Mehrotra, Efficient execution of aggregation queries over encrypted relational databases, in Database Systems for Advanced Applications, 2004, pp. 125-136.

[5] E. Mykletun and G. Tsudik, Aggregation queries in the database-as-aservice model, in Data and Applications Security, 2006, pp. 89-103.

[6] H. Hu, J. Xu, C. Ren, B. Choi, Processing private queries over untrusted data cloud through privacy homomorphism, in ICDE, 2011, pp. 601-612.

[7] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, Secure knn computation on encrypted databases, in SIGMOD, 2009, pp. 139-152.

[8] Y. Zhu, R. Xu, T. Takagi, Secure k-nn computation on encrypted cloud data without sharing key with query users, in Security in cloud computing, 2013, pp. 55-60.

[9] B. Yao, F. Li, X. Xiao, Secure nearest neighbor revisited, in ICDE, 2013, pp. 733-744.

[10] Y. Elmehdwi, B. K. Samanthula, W. Jiang, Secure k-nearest neighbor query over encrypted data in outsourced environments, in ICDE, 2014, pp. 664-675.

[11] M. L. Yiu, G. Ghinita, C. S. Jensen, P. Kalnis, Enabling search services on outsourced private spatial data, VLDB Journal, vol. 19, issue 3, 2010, pp. 363-384.

[12] F. P. Miller, A. F. Vandome, J. McBrewster, Advanced encryption standard. Proceedings of the Advanced Encryption Standard, 2009.

[13] J. Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism, in ISC, 2002, pp. 471-483.

[14] H. Kim, S. Hong, and J. Chang, Hilbert curve-based cryptographic transformation scheme for spatial query processing on outsourced private data, in Big Data and Smart Computing, 2014, pp. 77-82.

[15] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in EUROCRYPT, 1999, pp. 223-238.

[16] H. Carmit, Y. Lindell, Efficient secure two-party protocols, Techniques and constructions, 2010.

[17] A. Liu, K. Zheng, L. Li, G. Liu, L. Zhao, X. Zhou, Efficient Secure Similarity Computation on Encrypted Trajectory Data, in ICDE, 2015, pp. 66-77.

[18] S. Bugiel, S. Nürnberger, A. Sadeghi, T. Schneider, Twin clouds: Secure cloud computing with low latency, in CMS, 2011, pp. 32-44.

[19] B. K. Samanthula, H. Chun, W. Jiang, An efficient and probabilistic secure bit-decomposition, in ASIACCS, 2013, pp. 541-546.