A Secure and Efficient kNN Classification Algorithm Using Encrypted Index Search and Yao's Garbled Circuit over Encrypted Databases

Hyeong-Jin Kim, Jae-Hwan Shin, and Jae-Woo Chang(✉)

Department of Computer Engineering, Chonbuk National University, Jeonju, South Korea
{yeon_hui4,djtm99,jwchang}@jbnu.ac.kr

Abstract. Database outsourcing has been popular according to the development of cloud computing. Databases need to be encrypted before being outsourced to the cloud so that they can be protected from adversaries. However, the existing *k*NN classification scheme over encrypted databases in the cloud suffers from high computation overhead. So we proposed a secure and efficient *k*NN classification algorithm using encrypted index search and Yao's garbled circuit over encrypted databases. Our algorithm not only preserves data privacy, query privacy, and data access pattern. We show that our algorithm achieves about 17x better performance on classification time than the existing scheme, while preserving high security level.

Keywords: Database outsourcing · Data privacy · Query protection
Hiding data access pattern · kNN classification algorithm · Cloud computing

1 Introduction

Research on preserving data privacy in outsourced databases has been spotlighted with the development of a cloud computing. Since a data owner (*DO*) outsources his/her databases and allows a cloud to manage them, the *DO* can reduce the cost of data management by using the cloud's resources. However, because the data are private assets of the *DO* and may include sensitive information, they should be protected against adversaries including a cloud server. Therefore, the databases should be encrypted before being outsourced to the cloud. A vital challenge in the cloud computing is to protect both data privacy and query privacy. Meanwhile, during query processing, the cloud can derive sensitive information from the actual data items and users by observing data access patterns even if the data and the query are encrypted [1].

Meanwhile, a classification has been widely adopted in various fields such as marketing and scientific applications. Among various classification methods, a *k*NN classification algorithm is used in various fields because it does not require a time consuming learning process while guaranteeing good performance with moderate *k* [2]. When a query is given, a *k*NN classification first retrieves the *k*NN results for the query. Then, it determines the majority class label (or category) among the labels of *k*NN results. However, since the intermediate *k*NN results and the resulting class label are

T. K. Dang et al. (Eds.): FDSE 2018, LNCS 11251, pp. 21–38, 2018.
https://doi.org/10.1007/978-3-030-03192-3_3

closely related to the query, the queries should be more cautiously dealt to preserve the privacy of the users.

However, to the best of our knowledge, a *k*NN classification scheme proposed by Samanthula [3] is the only work that performs classification over the encrypted data in the cloud. The scheme preserves data privacy, query privacy, and intermediate results throughout the query processing. The scheme also hides data access pattern from the cloud. To achieve this, they adopt S*k*NN$_m$ [4] scheme among various secure *k*NN schemes [4–7] when retrieving k relevant records to a query. However, the scheme suffers from high computation overhead because it considers all the encrypted data during the query processing.

To solve the problem, in this paper, we propose a secure and efficient *k*NN classification algorithm over encrypted databases. Our algorithm can preserve data privacy, query privacy, the resulting class labels, and data access patterns from the cloud. To enhance the performance of our algorithm, we adopt the encrypted index scheme proposed in our previous work [7]. For this, we also propose efficient and secure protocols based on the Yao's garbled circuit [8] and a data packing technique.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 presents our overall system architecture and various secure protocols. Section 4 proposes our *k*NN classification algorithm over encrypted databases. Section 5 presents the performance analysis. Finally, Sect. 6 concludes this paper with some future research directions.

2 Background and Related Work

2.1 Background

Paillier Crypto System. The Paillier cryptosystem [9] is an additive homomorphic and probabilistic asymmetric encryption scheme for public key cryptography. The public encryption key pk is given by (N, g), where N is a product of two large prime numbers p and q, and g is in $Z^*_{N^2}$. Here, $Z^*_{N^2}$ denotes an integer domain ranging from 0 to N^2. The secret decryption key sk is given by (p, q). Let $E()$ and $D()$ denote the encryption and decryption functions, respectively. The Paillier crypto system provides 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. (ii) Homomorphic multiplication: The b^{th} power of ciphertext $E(m_1)$ results in the encryption of the product of b and m_1. (iii) Semantic security: Encrypting the same plaintexts using the same encryption key does not result in the identical ciphertexts. Therefore, an adversary cannot infer any information about the plaintexts.

Yao's Garbled Circuit. Yao's garbled circuits [8] allows two parties holding inputs x and y, respectively, to evaluate a function $f(x, y)$ without leaking any information about the inputs beyond what is implied by the function output. One party generates an encrypted version of a circuit to compute f. The other party obliviously evaluates the output of the circuit without learning any intermediate values. Therefore, the Yao's garbled circuit provides high security level. Another benefit of using the Yao's garbled

circuit is that it can provide high efficiency if a function can be realized with a reasonably small circuit.

Adversarial Models. There are two main types of adversarial models, *semi-honest* and *malicious* [10, 11]. In this paper, we assume that clouds act as insider adversaries with high capability. In the *semi-honest* adversarial model, the clouds honestly follow the protocol specification, but try to use the intermediate data in malicious way to learn forbidden information. In the *malicious* adversarial model, the clouds can arbitrarily deviate from the protocol specification. Protocols against malicious adversaries are too inefficient to be used in practice while protocols under the *semi-honest* adversaries are acceptable in practice. Therefore, by following the work done in [4, 10], we also consider the semi-honest adversarial model in this paper.

2.2 Secure kNN Classification Schemes

To the best of our knowledge, Samanthula proposed a *k*NN classification scheme (PP*k*NN) [3], which is the only work that performs classification over the encrypted data. The scheme performs S*k*NN$_m$ [4] scheme to retrieve k relevant records to a query and determines the class label of the query. The scheme can preserve both data privacy and query privacy while hiding data access pattern. However, the scheme suffers from the high computation overhead because it directly adopts the S*k*NN$_m$ scheme.

3 System Architecture and Secure Protocols

In this section, we explain our overall system architecture and present generic secure protocols used for our kNN classification algorithm.

3.1 System Architecture

We provide the system architecture of our scheme, which is designed by adopting that of our previous work [7]. Our previous work has a disadvantage that comparison operations cause high overhead by using encrypted binary arrays [7]. To solve this problem, we propose an efficient query processing algorithm that performs comparison operations through yao's garbled circuits [8]. Figure 1 shows the overall system architecture and Table 1 summarizes common notations used in this paper. The system consists of four components: data owner (*DO*), authorized user (*AU*), and two clouds (C_A and C_B). The *DO* stores the original database (T) consisting of n records. A record $t_i (1 \leq i \leq n)$ consists of $(m + 1)$ attributes and $t_{i,j}$ denotes the j^{th} attribute value of t_i. A class label of t_i is stored in $(m + 1)^{\text{th}}$ attribute, i.e., $t_{i,m+1}$. We do not consider $(m + 1)^{\text{th}}$ attribute when making an index using T. Therefore, the *DO* indexes on T by using a kd-tree, based on $t_{i,j} (1 \leq i \leq n$ and $1 \leq j \leq m)$. The reason why we utilize a kd-tree (k-dimensional tree) as a space-partitioning data structure is that it not only can evenly partition data into each node, but also is useful for organizing points in a k-dimensional space [14]. When we visit the tree in a hierarchical manner, access patterns can be disclosed. Consequently,

we only consider the leaf nodes of the kd-tree and all of the leaf nodes are retrieved once during the query processing step. Let h denote the level of the kd-tree and F be a fan-out which is the maximum number of data to be stored in each node. The total number of leaf nodes is 2^{h-1}. Henceforth, a node refers to a leaf node. The region information of each node is represented as both the lower bound $lb_{z,j}$ and the upper bound $ub_{z,j}\left(1 \le z \le 2^{h-1}, 1 \le j \le m\right)$. Each node stores the identifiers (id) of data located in the node region. Although we consider the kd-tree in this paper, another index structure whose nodes store region information can be applied to our scheme.

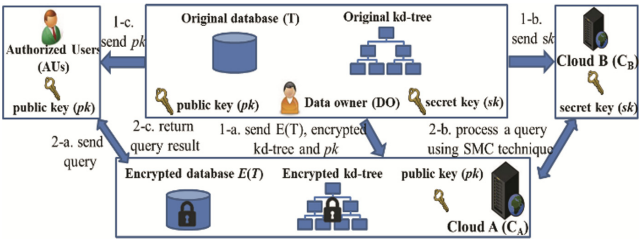


Fig. 1. The overall system architecture

Table 1. Common notations

Notations	Description
$E()$, $D()$	Encryption function and decryption function
t_i, $t_{i,j}$	i^{th} record and j^{th} attribute value of i^{th} record
t'_i	i^{th} extracted record during the index search
q, q_j	a query of a user and j^{th} attribute value of a query q
$node_z$	z^{th} node of the kd-tree
$node_z.t_{s,j}$	j^{th} attribute of s^{th} record stored in z^{th} node of the kd-tree
$lb_{z,j}$, $ub_{z,j}$	j^{th} attribute value of lower/upper bound of z^{th} kd-tree node
r	Random integers

To preserve data privacy, the DO encrypts T attribute-wise by using the public key (pk) of the Paillier cryptosystem [9] before outsourcing the database. Thus, the DO generates $E(t_{i,j})$ for $1 \le i \le n$ and $1 \le j \le m$. The DO also encrypts the region information of all kd-tree nodes to support efficient query processing. Specifically, $E(lb_{z,j})$ and $E(ub_{z,j})$ are generated with $1 \le z \le 2^{h-1}$ and $1 \le j \le m$ by encrypting lb and ub of each node attribute-wise. Assuming that C_A and C_B are non-colluding and semi-honest (or honest-but-curious) clouds, they correctly execute the assigned protocols, but an adversary may try to obtain additional information from the intermediate data while executing the assigned protocol. This assumption is not new and has been considered in earlier work [4, 10]. Specifically, because most cloud services are provided by renowned IT companies, collusion between them that would blemish their reputations is improbable [4].

To process *k*NN classification algorithm over the encrypted database, we utilize a secure multiparty computation (SMC) between C_A and C_B. To do this, the *DO* outsources both the encrypted database and its encrypted index to a cloud with *pk*, C_A in this case, but it sends *sk* to a different cloud, C_B in this case. In addition, the DO outsources the list of encrypted class labels denoted by $E(label_i)$ for $1 \leq i \leq w$ to C_A. The encrypted index includes the region information of each node in cipher-text and the *ids* of data located in the node in plaintext. The *DO* also sends *pk* to *AU*s to allow them to encrypt a query. At query time, an *AU* 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 sends the query result to the *AU*.

As an example, assume that an *AU* has eight data instances as depicted in Fig. 2. Each data t_i is depicted with its class label (e.g., 3 in case of t_6). The data are partitioned into four nodes (e.g., node_1– node_4) for a kd-tree. The *DO* encrypts each data instance and the region of each node attribute-wise. For example, t_6 is encrypted as $E(t_6) = \{E(8), E(5), E(3)\}$ because the values of *x*-axis and *y*-axis are 8 and 5, respectively, and the class label of t_6 is 3. Meanwhile, the $node_1$ is encrypted as $\{\{\text{E}(0), \text{E}(0)\}, \{\text{E}(5), \text{E}(5)\}, \{1, 2\}\}$ because the *lb* and *ub* of $node_1$ are $\{0, 0\}$ and $\{5, 5\}$, respectively, and the $node_1$ stores both t_1 and t_2.

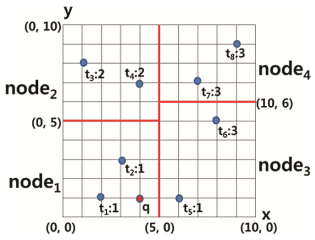


Fig. 2. An example in two-dimensional space

3.2 Secure Protocols

Our *k*NN classification algorithm is constructed using several secure protocols. In this section, all of the protocols except the SBN are performed with the SMC technique between C_A and C_B. The SBN can be solely executed by C_A. Due to space limitations, we briefly introduce five secure protocols found in the literature [3, 4, 7, 10]. (i) SM (Secure Multiplication) [4] 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. (ii) SBN (Secure Bit-Not) [7] performs a bit-not operation when an encrypted bit $E(a)$ is given as an input. (iii) CMP-S [10] returns 1 if $u < v$, 0 otherwise, when $-r_1$ and $-r_2$ are given from C_A as well as $u + r_1$ and $v + r_2$ are given from C_B. (iv) SMS_n (Secure Minimum Selection) [10] returns the minimum value among the inputs by performing the CMP-S for $n - 1$ times when $E(d_i)$ for $1 \leq i \leq n$ are given as inputs. (v) SF (Secure Frequency) [3] returns $E(f(label_j))$, the number of occurrence of each $E(label_j)$ in $E(c_i)$, when both $E(c_j)$ for $1 \leq i \leq k$ and $E(label_j)$ for $1 \leq j \leq w$ are given as inputs.

Meanwhile, we propose new secure protocols, i.e., ESSED, GSCMP, and GSPE. Contrary to the existing protocols, the proposed protocols do not take the encrypted binary representation of the data, like *E(0)* or *E(1)*, as inputs. Therefore, our protocols can provide a low computation cost. Next, we propose our new secure protocols.

ESSED Protocol. ESSED (Enhanced Secure Squared Euclidean Distance) computes $E(|X-Y|^2)$ when two encrypted vectors $E(X)$ and $E(Y)$ are given as inputs, where X and Y consist of m attributes. To enhance the efficiency, we pack λ number of σ-bit data instances to generate a packed value. The overall procedure of ESSED is as follows. First, C_A generates random numbers r_j for $1 \le j \le m$ and packs them by computing $R = \sum_{j=1}^{m} r_j \times 2^{\sigma(m-j)}$. Then, C_A generates $E(R)$ by encrypting R. Second, C_A calculates $E(x_j - y_j)$ attribute-wise and packs them by computing $E(v) = \prod_{j-1}^{m} E(x_j - y_j)^{2^{\sigma(m-j)}}$. Then, C_A computes $E(v) = E(v) \times E(R)$ and sends $E(v)$ to C_B. Third, assuming that w_j denotes $x_j - y_j + r_j (1 \le j \le m)$, C_B acquires $v = [w_1|\ldots|w_m]$ by decrypting $E(v)$. C_B obtains w_j for $1 \le j \le m$ by unpacking v through $v \times 2^{-\sigma(m-j)}$. Here, each instance of w_j represents the randomized distance of two input vectors for each attribute. C_B also calculates w_j^2 attribute-wise and stores their sum into d. C_B encrypts d and sends $E(d)$ to C_A. Finally, C_A obtains $E(|X-Y|^2)$ by eliminating randomized values using the following Eq. (1).

$$E(|X-Y|^2) = E(d) \times \prod_{j=1}^{m} \left(E(x_j - y_j)^{-2rj} \times E(r_j^2)^{-1} \right) \tag{1}$$

Our ESSED achieves better performance than the existing distance computation protocol, DPSSED [10], in two aspects. First, our ESSED requires only one encryption operation on the C_B side while DPSSED needs m times. Second, our ESSED calculates the randomized distance in plaintext on the C_B side while DPSSED computes the sum of the squared Euclidean distances among all attributes over ciphertext on the C_A side. Therefore, the number of computations on encrypted data in our ESSED can be reduced greatly.

GSCMP Protocol. When $E(u)$ and $E(v)$ are given as inputs, GSCMP (Garbled Circuit based Secure Compare) protocol returns 1 if $u \le v$, 0 otherwise. The main difference between GSCMP and CMP-S is that GSCMP receives encrypted data as inputs while CMP-S receives the randomized plaintext. The overall procedure of the GSCMP is as follows. First, C_A generates two random numbers r_u and r_v, and encrypts them. C_A computes $E(m_1) = E(u)^2 \times E(r_u)$ and $E(m_2) = E(v)^2 \times E(1) \times E(r_v)$. Second, C_A randomly selects one functionality between $F_0{:}u > v$ and $F_1{:}v > u$. The selected functionality is oblivious to C_B. Then, C_A sends data to C_B, depending on the selected functionality. If $F_0{:}u > v$ is chosen, C_A sends $<E(m_2), E(m_1)>$ to C_B. If $F_1{:}u < v$ is chosen, C_A sends $<E(m_1), E(m_2)>$ to C_B. Third, C_B obtains $<m_2, m_1>$ by decrypting $<E(m_2), E(m_1)>$ if $F_0{:}u > v$ is chosen. If $F_1{:}u < v$ is chosen, C_B obtains $<m_1, m_2>$ by decrypting $<E(m_1), E(m_2)>$. Fourth, C_A generates a garbled circuit consisting of two ADD circuits and one CMP circuit. Here, ADD circuit takes two integers u and v as input, and outputs $u + v$ while CMP circuit takes two integers u and v as input, and

outputs 1 if $u < v$, 0 otherwise. If F_0:$u > v$ is selected, C_A puts $-r_v$ and $-r_u$ into the 1st and 2nd ADD gates, respectively. If F_1:$u < v$ is selected, C_A puts $-r_u$ and $-r_v$ into the 1st and 2nd ADD gates. Fifth, if F_0:$u > v$ is selected, C_B puts m_2 and m_1 into the 1st and 2nd ADD gates, respectively. If F_1:$u < v$ is selected, C_B puts m_1 and m_2 into the 1st and 2nd ADD gates. Sixth, the 1st ADD gate adds two input values and puts the output $result_1$ into CMP gate. Similarly, the 2nd ADD gate puts the output $result_2$ into CMP gate. Seventh, CMP gate outputs $\alpha = 1$ if $result_1 < result_2$ is true, $\alpha = 0$ otherwise. The output of the CMP is returned to the C_B. Then, C_B encrypts α and sends $E(\alpha)$ to C_A. Finally, only when the selected functionality is F_0:$u > v$, C_A computes $E(\alpha) = \mathrm{SBN}(E(\alpha))$ and returns the final $E(\alpha)$. If $E(\alpha)$ is E(1), u is less than v.

GSPE Protocol. GSPE (Garbled circuit based Secure Point Enclosure) protocol returns $E(1)$ when p is inside a *range* or on a boundary of the *range*, $E(0)$ otherwise. GSPE takes an encrypted point $E(p)$ and an encrypted range $E(range)$ as inputs. Here, the *range* consists of the $E(lb_j)$ and the $E(ub_j)$ for $1 \le j \le m$. If $E(p_j) \le E(range.ub_j)$ and $E(p_j) \ge E(range.lb_j)$, the p is inside a *range*. The overall procedure of the GSPE is as follows. First, C_A generates two random numbers ra_j and rb_j for $1 \le j \le 2m$. C_A obtains packed values RA and RB by packing ra_j and rb_j, respectively, using the following Eq. (2) for $1 \le j \le 2m$.

$$RA = \sum_{j=1}^{2m} ra_j \times 2^{\sigma(2m-j)}, RB = \sum_{j=1}^{2m} rb_j \times 2^{\sigma(2m-j)} \tag{2}$$

Here, σ means the bit length to represent a data. Then, C_A generates $E(RA)$ and $E(RB)$ by encrypting RA and RB. Second, C_A computes $E(\mu_j) = E(p_j)^2$ and $E(\omega_j) = E(range.lb_j)^2$ for $1 \le j \le m$. C_A also computes $E(\delta_j) = E(p_j)^2 \times E(1)$ and $E(\rho_j) = E(range.ub_j)^2 \times E(1)$ for $1 \le j \le m$. Third, C_A randomly selects one functionality between F_0:$u > v$ and F_1:$v > u$. Then, C_A performs data packing by using the $E(\mu_j)$ and $E(\rho_j)$, depending on the selected functionality.

– If F_0: $u > v$ is selected, compute

$$E(RA) = E(RA) \times E(\rho_j)^{2^{\sigma(2m-j)}}, E(RB) = E(RB) \times E(\mu_j)^{2^{\sigma(2m-j)}}$$

– If F_1: $v > u$ is selected, compute

$$E(RA) = E(RA) \times E(\mu_j)^{2^{\sigma(2m-j)}}, E(RB) = E(RB) \times E(\rho_j)^{2^{\sigma(2m-j)}}$$

In addition, C_A performs data packing by using the $E(\omega_j)$ and $E(\delta_j)$, depending on the selected functionality. Then, C_A sends packed values $E(RA)$ and $E(RB)$ to C_B.

– If F_0: $u > v$ is selected, compute

$$E(RA) = E(RA) \times E(\delta_j)^{2^{\sigma(2m-j)}}, E(RB) = E(RB) \times E(\omega_j)^{2^{\sigma(2m-j)}}$$

– If F_1: $v > u$ is selected, compute

$$E(RA) = E(RA) \times E(\omega_j)^{2^{\sigma(2m-j)}}, E(RB) = E(RB) \times E(\delta_j)^{2^{\sigma(2m-j)}}$$

Fourth, C_B obtains RA and RB by decrypting $E(RA)$ and $E(RB)$. C_B computes $ra_j + u_j \leftarrow RA \times 2^{-\sigma(2m-j)}$ and $rb_j + v_j \leftarrow RB \times 2^{-\sigma(2m-j)}$ for $1 \le j \le 2m$. Here, u_j (or v_j) is one of the μ_j, ρ_j, ω_j, and δ_j. Fifth, C_A generates CMP-S circuit and puts $-ra_j$ and $-rb_j$ into CMP-S while C_B puts $ra_j + u_j$ and $rb_j + v_j$ into CMP-S for $1 \le j \le 2m$. Once four inputs (i.e., $-ra_j$, $-rb_j$, $ra_j + u_j$ and $rb_j + v_j$) are given to CMP-S, the output α'_j is returned to C_B. Then, C_B encrypts α' and sends $E(\alpha')$ to C_A. Sixth, C_A performs $E\left(\alpha'_j\right) = \mathrm{SBN}\left(E\left(\alpha'_j\right)\right)$ for $1 \le j \le 2m$ only when the selected functionality is F_0:$u > v$. Then, C_A computes $E(\alpha) = \mathrm{SM}\left(E(\alpha), E\left(\alpha'_j\right)\right)$ where the initial value of $E(\alpha)$ is $E(1)$. Only when all of the $E\left(\alpha'_j\right)$ for $1 \le j \le 2m$ are $E(1)$, the value of $E(\alpha)$ remains $E(1)$. Finally, GSPE outputs the final $E(\alpha)$. The p is inside the *range* if the final $E(\alpha)$ is $E(1)$.

$\mathrm{SXS_n}$ Protocol. $\mathrm{SXS_n}$ (Secure Maximum Selection) returns the maximum value among the inputs when $E(d_i)$ for $1 \le i \le n$ are given as inputs. $\mathrm{SXS_n}$ can be realized by converting the logic of $\mathrm{SMS_n}$ in opposite way. Therefore, we omit the detailed procedure of $\mathrm{SXS_n}$ due to the space limitation.

4 KNN Classification Algorithm

In this section, we present our *k*NN classification algorithm ($\mathrm{S}k\mathrm{NNC_G}$) which uses the Yao's garbled circuit. Our algorithm consists of four steps; encrypted kd-tree search step, *k*NN retrieval step, result verification step, and majority class selection step.

4.1 Step 1: Encrypted kd-Tree Search Step

In the encrypted kd-tree search phase, the C_A securely extracts all of the data from a node containing a query point while hiding the data access patterns. To obtain high efficiency, we redesign the index search scheme proposed in our previous work [7]. Specifically, our algorithm does not require operations related to the encrypted binary representation which causes high computation overhead. In addition, we utilize our newly proposed secure protocols based on Yao's garbled circuit.

Algorithm 1: 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 $1 \le z \le num_{node}$ // $num_{node} = 2^{h-1}$ (h : level of the kd-tree)
02: $E(\alpha_z) \leftarrow GSPE(E(q), E(node_z))$
03: $E(\alpha') \leftarrow \pi(E(\alpha))$; send $E(\alpha')$ to C_B
$\mathbf{C_B}$: **04:** $\alpha' \leftarrow D(E(\alpha'))$; c ← the number of '1' in α'
05: create c number of Group
06: for each *NG*
07: assign a node with $\alpha'=1$ and $num_{node}/c-1$ nodes with $\alpha'=0$
08: *NG'* ← shuffle the *ids* of nodes
09: send *NG'* to CA
$\mathbf{C_A}$: **10:** *cnt* ← 0
11: NG^* ← permute node *ids* using π^{-1} for each *NG'*
12: for each NG^*
13: for $1 \le s \le F$
14: for $1 \le i \le num$ (# nodes in the selected NG^*)
15: $z = id$ of i^{th} node of NG^*
16: $E(t'_{i,j}) \leftarrow SM(node_z.t_{s,j}, E(\alpha_z))$ for $1 \le j \le m+1$
17: for $1 \le j \le m+1$
18: $E(cand_{cnt,j}) \leftarrow \prod_{i=1}^{num} E(t'_{i,j})$
19: *cnt* ← *cnt*+1
20: return E(*cand*)

The procedure of the encrypted *k*d-tree search step is shown in Algorithm 1. First, C_A securely finds nodes which include a query by executing $E(\alpha_z) \leftarrow \mathrm{GSPE}(E(q), E(node_z))$ for $1 \le z \le num_{node}$ where num_{node} means the total number of *k*d-tree leaf nodes (lines 1–2). Note that the nodes with $E(\alpha_z) = E(1)$ are related to the query, but both C_A and C_B cannot know whether or not the value of each $E(\alpha_z)$ is $E(1)$, because the Paillier encryption provides semantic security. Then, we partially perform the index search algorithm in [7]. Specifically, C_A generates $E(\alpha')$ by permuting $E(\alpha)$ using a random permutation function π and then sends $E(\alpha')$ to C_B (line 3). For example, the output of GSPE is $E(\alpha) = \{E(1), E(0), E(0), E(0)\}$ in Fig. 2 because the q is given 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.

Third, C_B obtains α' by decrypting $E(\alpha')$ and counts the number of $\alpha' = 1$ and stores it into c. Here, c means the number of nodes that the query is related to (line 4). Fourth, C_B creates c number of node groups. Assuming that *NG* denotes a node group, C_B assigns to each *NG* both a node with $\alpha' = 1$ and $num_{node}/c - 1$ nodes with $\alpha' = 0$. Then, C_B obtains NG' by randomly shuffling the *ids* of nodes in each *NG* and sends NG' to C_A (lines 5–9). For example, C_B can obtain $\alpha' = \{0, 0, 0, 1\}$ which contains one at the fourth

position. Because one node group is required, C_B assigns all nodes to one node group and randomly shuffles the *ids* of the nodes, i.e., $NG'_1 = \{2, 1, 3, 4\}$.

Fifth, C_A obtains NG^* by permuting the *ids* of nodes using π^{-1} in each NG' (line 11). Six, C_A gets access to one datum in a node for each NG^* and executes $\mathrm{E}\left(t'_{i,j}\right) = \mathrm{SM}\left(\mathrm{E}\left(node_z.t_{s,j}\right), \mathrm{E}\left(\alpha_z\right)\right)$ for $1 \le s \le F$ and $1 \le j \le m+1$ where $\mathrm{E}\left(\alpha_z\right)$ is the result of GSPE corresponding to $node_z$ (line 12–16). As a result, SM results in $\mathrm{E}\left(node_z.t_{s,j}\right)$ only for the data inside the nodes related to the query because their $E\left(\alpha_z\right)$ values are $E(1)$; otherwise SM results in $E(0)$. If a node has the less number of data than F, it performs SM by using $\mathrm{E}(max)$, instead of using $\mathrm{E}\left(node_z.t_{s,j}\right)$. Here, $E(max)$ is the largest value in the domain. When C_A accesses one datum from every node in a NG^*, C_A performs $E\left(cand_{cnt,j}\right) \leftarrow \prod_{i=1}^{num} E(t'_{i,j})$ where *num* means the total number of nodes in the selected NG^* (line 17–18). As a result, a datum in the nodes related to the query is securely extracted without revealing the data access patterns because the searched nodes are not revealed. By repeating these steps, all of the data in the nodes are safely stored into the $E\left(cand_{cnt,j}\right)$ for $1 \le i \le cnt$ and $1 \le j \le m+1$ where *cnt* means the total number of data extracted during the index search. For example, C_A obtains $NG^*_1 = \{3, 4, 2, 1\}$ by permuting the $NG'_1 = \{2, 1, 3, 4\}$ using π^{-1}. C_A gains access to $E(t_5)$ 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 $E(t_3)$ are $E(0)$ for all attributes because $E(\alpha_z)$ for the corresponding nodes are $E(0)$. The results are stored into $E\left(t'_1\right), E\left(t'_2\right)$ and $E\left(t'_3\right)$, respectively. However, the results of SM using $E(t_1)$ become $\{E(2), E(1), \mathrm{E}(1)\}$ because the values of x-axis and y-axis are 2 and 1, respectively, and the class label of t_1 is 1. The results are stored into $E\left(t'_4\right)$. Thus, the final attribute-wise homomorphic addition of $E(t_i)$ for $1 \le i \le 4$ are $\{E(2), E(1), E(1)\}$. Accordingly, one datum $E(t_1)$ in $node_1$ is securely extracted. By repeating this, the encrypted kd-tree search step can extract all of the data in $node_1$ (e.g., $E(t_1)$ and $E(t_2)$) and finally stores them into $E(cand)$.

4.2 Step 2: kNN Retrieval Step

In the *k*NN retrieval phase, we retrieve the k closest data from the query by partially utilizing the S*k*NN$_{\mathrm{m}}$ scheme [4]. However, we only consider $E\left(cand_i\right)$ for $1 \le i \le cnt$, which are extracted in the index search phase, whereas the S*k*NN$_{\mathrm{m}}$ considers all the encrypted data. In addition, we utilize our efficient protocols which require relatively low computation costs, instead of using the existing expensive protocols. The procedure of the *k*NN retrieval step is shown in Algorithm 2.

Algorithm 2: *k*NN retrieval

Input : E(q), E($cand$), k

Output : E(t′) // candidate kNN result

$\mathbf{C_A}$: **01:** for $1 \le i \le cnt$

02: E(d_i) ← ESSED(E(q), E($cand_i$))

03: for $1 \le s \le k$

04: E(d_{min}) ← $\mathrm{SMS_n}$(E(d_1), …, E(d_{cnt}))

05: for $1 \le i \le cnt$

06: E(τ_i) ← E(d_{min})×E(d_i)$^{N-1}$; E(τ'_i) ← E(τ_i)^{r_i}.

07: E(β) ← π(τ'); send E(β) to C_B

$\mathbf{C_B}$: **08:** for $1 \le i \le cnt$

09: if D(β_i)=0 then E(U_i) ← E(1); else E(U_i) ← E(0)

10: send E(U) to $\mathrm{C_A}$

$\mathbf{C_A}$: **11:** E(V) ← π^{-1}(U)

12: for $1 \le i \le cnt$

13: E($V'_{i,j}$)←SM(E(V_i),E($cand_{i,j}$)) for $1 \le j \le m+1$

14: E($t'_{s,j}$) = $\prod_{i=1}^{cnt} E(V'_{i,j})$ for $1 \le j \le m+1$

15: if $s < k$

16: E(d_i)←SM($E(V_i)$, E(max))×SM(SBN(E(V_i)), E(d_i))

17: return E(t')

First, using our proposed ESSED, C_A securely calculates the squared Euclidean distances $E(d_i)$ between a query and $E(cand_i)$ for $1 \le i \le cnt$ (lines 1–2). Then, instead of using the inefficient $\mathrm{SMIN_n}$, C_A performs $\mathrm{SMS_n}$ to find the minimum value $E(d_{min})$ among $E(d_i)$ for $1 \le i \le cnt$. Second, C_A calculates $\mathrm{E}(\tau_i) = E(d_{min}) \times E(d_i)^{N-1}$, i.e., the difference between the $E(d_{min})$ and $E(d_i)$, for $1 \le i \le cnt$. Then, C_A computes $\mathrm{E}(\tau'_i) = \mathrm{E}(\tau_i)^{r_i}$ (lines 3–6). Note that only the $E(\tau'_i)$ corresponding to the $E(d_{min})$ has a value of $E(0)$. C_A obtains $E(\beta)$ by shuffling $E(\tau')$ using a random permutation function π and then sends $E(\beta)$ to the C_B (line 7). For example, because $E(cand) = \{E(t_1), \mathrm{E}(t_2)\}$ is given from the index search phase, $E(d_1) = E(4)$ and $E(d_2) = E(5)$. By performing $\mathrm{SMS_n}$, $E(d_{min})$ is set as $E(4)$. Then, $E(\tau')$ is computed as $\{E(0), E(-r)\}$. The $E(\tau'_i)$ with $E(0)$ corresponds to the $E(d_{min})$, i.e., $E(t_1)$. Assuming that π permutes data in reverse way, C_A sends the $E(\beta) = \{E(-r), E(0)\}$ to C_B. Third, after decrypting $E(\beta)$, C_B sets $E(U_i) = E(1)$ if $E(\beta_i) = 0$, and sets $E(U_i) = E(0)$ otherwise. After C_B sends $E(U)$ to C_A, C_A obtains $E(V)$ by permuting $E(U)$ using π^{-1} (line 8–11). Then, C_A performs SM protocol by using $E(V_i)$ and $E(cand_{i,j})$ to obtain $E(V'_{i,j})$. By computing $E(t'_{s,j}) = \prod_{i=1}^{cnt} E(V'_{i,j})$ for $1 \le j \le m+1$, C_A can securely extract the datum corresponding to the $E(d_{min})$ (line 12–14). For example, C_B sends $E(U) = \{E(0), E(1)\}$ because the $\beta_2 = 1$. Then, C_A obtains $E(V) = \{E(1), E(0)\}$ by permuting $E(U)$ using π^{-1}. For the x-attribute, C_A performs $\mathrm{SM}(\mathrm{E}(cand_{1,1}), \mathrm{E}(V_1)) = \mathrm{E}(2)$ and $\mathrm{SM}(\mathrm{E}(cand_{2,1}), \mathrm{E}(V_2)) = \mathrm{E}(0)$. By adding the two values, the x-attribute value of $\mathrm{E}(t_1)$, i.e., E(2), is

securely calculated. Similarly, we can compute E(1), the y-attribute value of $\mathrm{E}(t_1)$. Therefore, we can store $\mathrm{E}(t_1) = \{\mathrm{E}(2), \mathrm{E}(1)\}$ into $\mathrm{E}(t'_1)$ without revealing data access patterns. Finally, to prevent the selected result from being selected in later phase, C_A securely updates the distance of the selected result as $E(max)$ by performing $E(d_i) = \mathrm{SM}(E(V_i), E(max)) \times \mathrm{SM}(\mathrm{SBN}(E(V_i)), E(d_i))$ (line 15-16). 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 1NN result and $E(t_2)$ with $E(d_2) = E(5)$ is selected in the second round as the 2NN result.

4.3 Step 3: Result Verification Step

The result of the step 2 may not be accurate because they are retrieved over the partial data being extracted in the step 1. Therefore, the result verification is essential to confirm the correctness of the current query result. Specifically, assuming that $dist_k$ denotes the squared Euclidean distance between the k^{th} closest result, i.e., $E(t'_k)$, and the query, the neighboring nodes located within $dist_k$ in the kd-tree need to be searched. For this reason, we use the concept of shortest point (*sp*) defined in [7]. The *sp* is a point in a given node whose distance is closest to a given point p as compared with the other points in the node. To find an *sp* in each node, we use the following properties. (i) If both the lower bound (*lb*) and the upper bound (*ub*) of the node are lesser than p, the *ub* is the *sp*. (ii) If both the *lb* and the *ub* of the region are greater than p, the *lb* is the *sp*. (iii) If p is between the *lb* and the *ub* of the region, p is the *sp*. To enhance the efficiency of the result verification algorithm in the previous work [7], we use our newly proposed protocols instead of using the existing expensive protocols.

Algorithm 3: Result verification

Input : E(q), E(node), E(t′), k
Output : E(c)
$\mathbf{C_A}$: **01:** E($dist_k$) = ESSED(E(q), E(t'_k))
02: for $1 \le z \le num_{node}$
03: for $1 \le j \le m$
04: E(ψ_1) ← GSCMP(E(q_j), E($node_z.lb_j$))
05: E(ψ_2) ← GSCMP(E(q_j), E($node_z.ub_j$))
06: E(ψ_3) ← E(ψ_1) × E(ψ_2) × SM(E(ψ_1), E(ψ_2))$^{N-2}$ // bit-xor
07: E($temp$) ← SM(E(ψ_1), E($node_z.lb_j$))
08: E($temp$) ← E($temp$)×SM(SBN(E(ψ_1)), E($node_z.ub_j$))
09: E($temp$) ← SM(E($temp$), SBN(E(ψ_3)))
10: E($sp_{z,j}$) ← E($temp$) × SM(E(ψ_3), E(q_j))
11: E($spdist_z$) ← ESSED(E(q), E(sp_z))
12: E($spdist_z$) = SM(E(α_z), E(max)) × SM(SBN(E(α_z)), E($spdist_z$))
13: E(α_z) ← GSCMP(E($spdist_z$), E($dist_k$))
14: E(t′′) ← perform 4~20 lines of Algorithm 1
15: E(t′) ← append the E(t′′) to E(t′)
16: E(result) ← perform Algorithm 2
17: for $1 \le i \le k$
18: E(c_i) ← E($\text{result}_{i,m+1}$)
19: return E(c)

The procedure of the result verification step is shown in Algorithm 3. First, C_A computes $\mathrm{E}(dist_k) = \mathrm{ESSED}(\mathrm{E}(q), E(t'_k))$ to calculate the squared Euclidean distance between the query and the k^{th} closest result among E(t'), i.e., the output of the kNN retrieval step (line 1). Second, C_A performs GSCMP by using $E(q_j)$ and $E(node_z.lb_j)$ for $1 \le z \le num_{node}$ and $1 \le j \le m$ and then stores the result in $E(\psi_1)$. C_A also performs GSCMP by using $E(q_j)$ and $E(node_z.ub_j)$ for $1 \le z \le num_{node}$ and $1 \le j \le m$ and then stores the result into $E(\psi_2)$. In addition, C_A calculates $E(\psi_3)$ by executing $E(\psi_1) \times E(\psi_2) \times \mathrm{SM}(E(\psi_1), E(\psi_2))^{N-2}$ to obtain the result of bit-xor operation between $E(\psi_1)$ and $E(\psi_2)$ (lines 3–6). Note that "−2" is equivalent to "$N - 2$" under Z_N. Third, C_A securely obtains the shortest point of each node, i.e., $E(sp_{z,j})$, by executing $\mathrm{SM}(E(\psi_3), E(q_j)) \times \mathrm{SM}(\mathrm{SBN}(E(\psi_3)), f(E(lb_{z,j}), E(ub_{z,j})))$ for $1 \le z \le num_{node}$ and $1 \le j \le m$, where $f(E(lb_j), E(ub_j))$ means $\mathrm{SM}(E(\psi_1), E(lb_{z,j})) \times \mathrm{SM}(\mathrm{SBN}(E(\psi_1)), E(ub_{z,j}))$ (lines 7–10). For example, assuming that the required k is 2, $\mathrm{E}(dist_2) = E(5)$ because E(t_2) is the current 2NN. Meanwhile, in Fig. 2, the shortest point of $node_3$ (i.e., sp_3) to the $E(q)$ is computed as follows. Because the x-value of the q is less than the x-values of both lb and ub of $node_3$, the x-value of $E(sp_3)$ is calculated by $E(sp_{3,1}) = E(0) \times E(4) + E(1) \times (E(1) \times E(5) + E(0) \times E(10)) = E(5)$. Similarly, the y-value of $E(sp_3)$ is computed as $E(sp_{3,2}) = E(1)$.

Fourth, C_A calculates $\mathrm{E}(spdist_z)$, the squared Euclidean distances between the query and $E(sp_z)$ for $1 \le z \le num_{node}$ by using ESSED. In addition, C_A securely updates the

$E(spdist_z)$ of the retrieved nodes into $E(max)$ by computing $E(spdist_z) = \mathrm{SM}(E(\alpha_z), E(max)) \times \mathrm{SM}(\mathrm{SBN}(E(\alpha_z)), E(spdist_z))$ (lines 11–12). Here, $E(\alpha_z)$ is the output of GSPE computed in index search step. Then, C_A performs $E(\alpha_z) = \mathrm{GSCMP}(E(spdist_z), E(dist_k))$ (line 13). The nodes with $E(\alpha_z) = E(1)$ need to be retrieved for query result verification. For example, the initial value of E(*spdist*) is ($E(0)$, $E(16)$, $E(1)$, $E(26)$) for each node in Fig. 2, and E(*spdist*) is updated as ($E(max)$, $E(16)$, $E(1)$, $E(26)$). Therefore, the result of GSCMP becomes $\mathrm{E}(\alpha) = (E(0), E(0), E(1), E(0))$ because $E(dist_k) = E(5)$. Fifth, C_A securely extracts the data stored in the nodes with $E(\alpha) = E(1)$ by performing the 4–20 lines of the Algorithm 1 and appends them to $E(t')$. Then, C_A executes the *k*NN retrieval step (Algorithm 2) based on $E(t')$ to obtain the $E(result_i)$ for $1 \le i \le k$ (lines 14–16). Finally, C_A stores $E(result_{i,m+1})$ into $E(c_i)$ for $1 \le i \le k$ to extract the class labels of the *k*NN results (line 18–19). For example, the final result becomes $E(result) = \{E(t_1), E(t_5)\}$. Because the class labels of both $E(t_1)$ and $E(t_5)$ are 1 in Fig. 2, the final $E(c)$ becomes ($E(1)$, $E(1)$).

4.4 Step 4: Majority Class Selection Step

We securely determine the majority class label among the output of the result verification step, i.e., $E(label)$. The procedure of the result verification step is shown in Algorithm 4. First, C_A performs SF using $E(label_j)$ for $1 \le j \le w$ and $E(c_i)$ for $1 \le i \le k$ to obtain $E(f(label_j))$. Then, C_A finds the maximum value, i.e., $E(f_{max})$, among $E(f(label_j))$ for $1 \le j \le w$ by using $\mathrm{SXS_n}$ (line 1–2). Second, C_A securely obtains the class label $E(output)$ corresponding to the $E(f_{max})$ by using the logic similar to 5–10 lines of Algorithm 2. Due to the space limitation, we briefly describe this procedure. C_A calculates $\mathrm{E}(\tau_i) = E(f_{max}) \times E(f(label_j))^{N-1}$ for $1 \le i \le w$. Then, C_A computes $\mathrm{E}(\tau'_i) = \mathrm{E}(\tau_i)^{r_i}$ and obtains $E(\beta)$ by shuffling $E(\tau')$ by using π and then sends $E(\beta)$ to the C_B (line 3–5). After decrypting $E(\beta)$, C_B sets $E(U_i) = E(1)$ if $E(\beta_i) = 0$, and sets $E(U_i) = E(0)$ otherwise. After C_B sends $E(U)$ to C_A, C_A obtains $E(V)$ by permuting $E(U)$ using π^{-1} (line 6–9). Then, C_A performs $E(output) = \prod_{j=1}^{w} \mathrm{SM}(\mathrm{E}(V_j), \mathrm{E}(label_j))$ for $1 \le j \le w$ to obtain the majority class label (line 10–12). For example, $E(output)$ is E(1) because the class label '1' has the maximum occurrence among $E(f(label)) = (\mathrm{E}(2), \mathrm{E}(0), \mathrm{E}(0))$. Third, C_A returns the decrypted result to AU in cooperation with C_B to reduce the computation overhead at the AU side. To do this, C_A computes $E(output) \times E(r)$ by generating a random value r, and then sends the result of $E(output + r)$ to C_B and r to AU (lines 14). C_B decrypts the data sent from C_A and sends the decrypted value (e.g., $output + r$) to AU (lines 15). Finally, AU computes the actual class label by computing $(output + r) - r$ in plaintext (lines 16–17).

Algorithm 4: Majority class selection

Input : E(*label*), E(*c*)
Output : *output*
C_A : **01:** E(*f*(*label*))← SF(*E*(*label*), *E*(*c*))
02: $E(f_{max}) \leftarrow SXS_n(E(f(lable_1)), \ldots, E(f(lable_w)))$
03: for $1 \le j \le w$
04: $E(\tau_j) \leftarrow E(f_{max}) \times E(f(label))^{N-1}$; $E(\tau'_j) \leftarrow E(\tau_j)^{r_j}$.
05: $E(\beta) \leftarrow \pi(\tau')$; send $E(\beta)$ to C_B
C_B : **06:** for $1 \le j \le w$
07: if $D(\beta_j)=0$ then $E(U_j) \leftarrow E(1)$; else $E(U_j) \leftarrow E(0)$
08: send E(*U*) to C_A
C_A : **09:** $E(V) \leftarrow \pi^{-1}(U)$
10: for $1 \le j \le k$
11: $E(V'_j) \leftarrow SM(E(V_j), E(label_j))$
12: $E(output) = \prod_{j=1}^{w} SM(E(V_j), E(label_j))$
14: send $E(output) \times E(r)$ to C_B; send *r* to AU
C_B : **15:** decrypt $E(output+r)$; send *output*+*r* to AU
AU: **16:** $output = (output + r) - r$
17: return *output*

5 Performance Analysis

In this section, we compare our S*k*NNC_G (secure *k*NN classification algorithm using the Yao's garbled circuit) with PP*k*NN [3] that is the only existing work to perform classification over encrypted databases in the cloud. To measure the performance gains of using our newly proposed protocols, we also compare our scheme with S*k*NNC_I (secure *k*NN classification algorithm with secure index) that performs classification based on the existing expensive secure protocols, instead of using our newly proposed protocols. Therefore, we can see that the performance gap between S*k*NNC_I and PP*k*NN comes from the use of secure index search scheme. We implemented three schemes by using C++ and evaluate their performances in terms of classification time under different parameters settings. The parameters used for our performance analysis are shown in Table 2. We used the Paillier cryptosystem to encrypt a database for all of the schemes. Our experiments were performed on a Linux machine running Ubuntu 14.04.2 with an Intel Xeon E3-1220v3 4-Core 3.10 GHz and 32 GB RAM. We conducted performance analysis by using the real Chess dataset because it is considered as an appropriate dataset for classification [15]. It consists of 28,056 records with six attributes and their class labels.

Table 2. Experimental parameters

Parameters	Values	Default value
Total number of data (n)	4k, 8k, 12k, 16k, 20k, 24k, 28k	28k (28,056)
Level of kd-tree (h)	5, 6, 7, 8, 9	7
Required k (k)	5, 10, 15, 20	10
Encryption key size (K)	512	512

In Fig. 3, we measure the performance of SkNNC$_{\mathrm{I}}$ and our SkNNC$_{\mathrm{G}}$ by varying the level of kd-tree because PPkNN does not use the secure index. The classification times of both schemes are decreased as h changes from 5 to 7 while the classification time increase as h changes from 7 to 9. This is because as h increases, the total number of leaf nodes grows, thus requiring more GSPE and SPE [7] executions for SkNNC$_{\mathrm{G}}$ and SkNNC$_{\mathrm{I}}$, respectively. Whereas, as h increases, the number of data in the node decreases, thus requiring less computation cost for distance calculation. However, our SkNNC$_{\mathrm{G}}$ outperforms SkNNC$_{\mathrm{I}}$ because our scheme uses both efficient secure protocols based on the Yao's garbled circuit and the data packing technique.

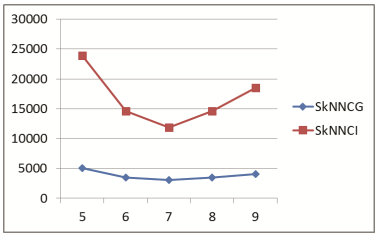
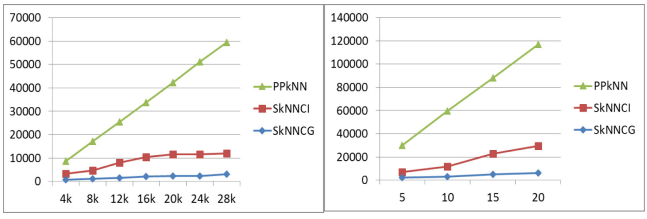


Fig. 3. Performance of varying h

Figure 4(a) shows the performance of three schemes by varying the n. As the n becomes larger, the query processing time of PPkNN linearly increases because it considers all of the data. Although the overall query processing times of SkNNC$_{\mathrm{I}}$ and SkNNC$_{\mathrm{G}}$ are increased as the n increases, they are less affected by n than PPkNN. Overall, our SkNNC$_{\mathrm{G}}$ shows 17.1 and 4.7 times better performance than PPkNN and SkNNC$_{\mathrm{I}}$, respectively. Due to the index-based data filtering, both SkNNC$_{\mathrm{G}}$ and SkNNC$_{\mathrm{I}}$ shows better performance than PPkNN. However, our SkNNC$_{\mathrm{G}}$ outperforms



(a) Performance of varying n (b) Performance of varying k

Fig. 4. Classification time for varying n

SkNNC$_I$ because our algorithm can reduce the computation cost by using the Yao's garbled circuit and the data packing technique.

Figure 4(b) shows the performance of three schemes by varying the k. As the k becomes larger, the query processing times of three schemes increase because the larger k requires more executions of expensive protocols, e.g., SMS$_n$ in case of our SkNNC$_G$ and SMIN$_n$ in case of both PPkNN and SkNNC$_I$, to retrieve the more kNN results. Overall, our SkNN$_G$ shows 17.7 and 4.2 times better performance than PPkNN and SkNNC$_I$, respectively, due to the same reasons described for Fig. 3.

6 Conclusion

Databases need to be encrypted before being outsourced to the cloud, due to its privacy issues. However, the existing kNN classification scheme over encrypted databases in the cloud has a problem that it suffers from high computation overhead. Therefore, in this paper we proposed a new secure and efficient kNN classification algorithm over encrypted databases. Our algorithm not only preserves data privacy and query privacy, but also conceals resulting class labels and data access pattern. In addition, our algorithm can support efficient kNN classification by using an encrypted index search scheme, the Yao's garbled circuit and a data packing technique. We showed from our performance analysis that the proposed algorithm showed about 17 times better performance on classification time than the existing PPkNN scheme, while preserving high security level.

As a future work, we plan to expand our algorithm to the distributed cloud computing environment. We also plan to study on data clustering and association rule mining over encrypted database for cloud computing.

Acknowledgment. This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (grant number 2016R1D1A3B03935298). This work was partly supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. R0113-17-0005, Development of an Unified Data Engineer-ing Technology for Large-scale Transaction Processing and Real-time Complex Analytics).

References

1. Vimercati, S., Foresti, S., Samarati, P.: Managing and accessing data in the cloud: privacy risks and approaches. In: CRiSIS, pp. 1–9 (2012)
2. Riley, J.W., Alfons, C., Fredäng, E., Lind, P.: Nearest Neighbor Classifiers (2009)
3. Samanthula, B., Elmehdwi, Y., Jiang, W.: K-nearest neighbor classification over semantically secure encrypted relational data. TKDE **27**(5), 1261–1273 (2015)
4. Elmehdwi, Y., Samanthula, B.K., Jiang, W.: Secure k-nearest neighbor query over encrypted data in outsourced environments. In: ICDE, pp. 664–675 (2014)
5. Hu, H., Xu, J., Ren, C., Choi, B.: Processing private queries over untrusted data cloud through privacy homomorphism. In: ICDE, pp. 601–612 (2011)

6. Zhu, Y., Xu, R., Takagi, T.: Secure k-NN computation on encrypted cloud data without sharing key with query users. In: Security in cloud computing, pp. 55–60 (2013)
7. Kim, H., Kim, H., Chang, J.: A kNN query processing algorithm using a tree index structure on the encrypted database. In: Big Data and Smart Computing (BigComp), pp. 93–100 (2016)
8. Yao, A.C.C.: How to generate and exchange secrets. In: Foundations of Computer Science, pp. 162–167 (1986)
9. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT, pp. 223–238 (1999)
10. Liu, A., Zhengy, K., Liz, L., Liu, G., Zhao, L., Zhou, X.: Efficient secure similarity computation on encrypted trajectory data. In: ICDE, pp. 66–77 (2015)
11. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security, vol. 201, no. 1 (2011)
12. Domingo-Ferrer, J.: A provably secure additive and multiplicative privacy homomorphism*. In: Chan, A.H., Gligor, V. (eds.) ISC 2002. LNCS, vol. 2433, pp. 471–483. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45811-5_37
13. Samanthula, B.K., Chun, H., Jiang, W.: An efficient and probabilistic secure bit-decomposition. In: ASIACCS, pp. 541–546 (2013)
14. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann (2006)
15. http://archive.ics.uci.edu/ml/