



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

EasyChair preprints are intended for rapid  
dissemination of research results and are  
integrated with the rest of EasyChair.

June 20, 2018

# A Secure and Efficient $k$ NN Classification Algorithm using Encrypted Index Search and Yao's Garbled Circuit over Encrypted Databases

Hyeong-Jin Kim

Dept. of Computer Engineering  
Chonbuk National University  
Jeonju, South Korea  
yeon\_hui4@jbnu.ac.kr

Jae-Hwan Shin

Dept. of Computer Engineering  
Chonbuk National University  
Jeonju, South Korea  
djtm99@jbnu.ac.kr

Jae-Woo Chang\*

Dept. of Computer Engineering  
Chonbuk Nat'l Univ., Jeonju,  
South Korea, jwchang@jbnu.ac.kr  
\*Corresponding author

*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 can preserve data privacy, query privacy, and data access pattern. We show that our algorithm achieves about 17 times 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;  $k$ NN classification algorithm; Cloud computing;

## I. 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 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 B. 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  $SkNN_m$  [4] scheme among various secure  $k$ NN schemes [4, 5, 6, 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, Section 6 concludes this paper with some future research directions.

## II. BACKGROUND AND RELATED WORK

### A. 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  $b^{\text{th}}$  power of ciphertext  $E(m_i)$  results in the encryption of the product of  $b$  and  $m_i$ . 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 obviously 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.

### B. Secure $k$ NN classification schemes

To the best of our knowledge, B. 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 SkNN $_m$  [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 SkNN $_m$  scheme.

## III. SYSTEM ARCHITECTURE AND SECURE PROTOCOLS

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

### A. System Architecture

We provide the system architecture of our scheme, which is designed by adopting that of our previous work [7]. 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  $j^{\text{th}}$  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  $ub_{z,j}$  ( $1 \leq z \leq 2^{h-1}$ ,  $1 \leq j \leq m$ ). 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.

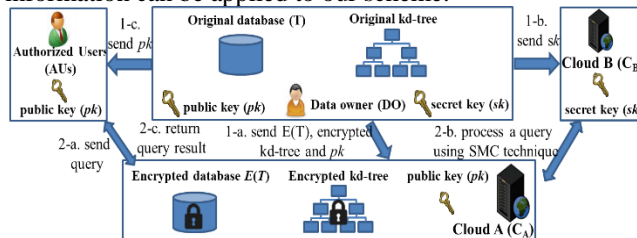


Figure 1. The overall system architecture

Table 1. Common notations

Notations	Description
$E(), D()$	encryption function and decryption function
$t_i, t_{i,j}$	$i^{\text{th}}$ record and $j^{\text{th}}$ attribute value of $i^{\text{th}}$ record
$t'_i$	$i^{\text{th}}$ extracted record during the index search
$q, q_j$	a query of a user and $j^{\text{th}}$ attribute value of a query $q$
$node_z$	$z^{\text{th}}$ node of the kd-tree
$node_{z,t_{s,j}}$	$j^{\text{th}}$ attribute of $s^{\text{th}}$ record stored in $z^{\text{th}}$ node of the kd-tree
$lb_{z,j}, ub_{z,j}$	$j^{\text{th}}$ attribute value of lower/upper bound of $z^{\text{th}}$ 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  $1 \leq i \leq n$  and  $1 \leq j \leq m$ . 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  $1 \leq z \leq 2^{h-1}$  and  $1 \leq j \leq m$  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  $AUs$  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 Figure 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.,  $node_1 \sim node_4$ ) 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  $\{\{E(0), E(0)\}, \{E(5), E(5)\}, \{1, 2\}\}$  because the  $lb$  and  $ub$  of  $node_1$  are  $\{0, 0\}$  and  $\{5, 5\}$ , respectively, and the  $node_1$  stores both  $t_1$  and  $t_2$ .

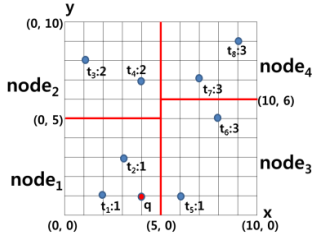


Figure 2. An example in two-dimensional space

## B. 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<sub>n</sub> (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, our proposed protocols do not take the encrypted binary representation of the data 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  $\lambda$  number of  $\sigma$ -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  $1 \leq j \leq m$  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  $x_j - y_j + r_j$  ( $1 \leq j \leq m$ ),  $C_B$  acquires  $v = [w_1 | \dots | w_m]$  by decrypting  $E(v)$ .  $C_B$  obtains  $w_j$  for  $1 \leq j \leq m$  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 equation (1).

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

Our ESSED achieves better performance than the existing distance computation protocol, DPSSSED [10], in two aspects. First, our ESSED requires only one encryption operation on the  $C_B$  side while DPSSSED needs  $m$  times. Second, our ESSED calculates the randomized distance in plaintext on the  $C_B$  side while DPSSSED 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  $u \leq v$ , 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  $\langle E(m_2), E(m_1) \rangle$  to  $C_B$ . If  $F_1: u < v$  is chosen,  $C_A$  sends  $\langle E(m_1), E(m_2) \rangle$  to  $C_B$ . Third,  $C_B$  obtains  $\langle m_2, m_1 \rangle$  by decrypting  $\langle E(m_2), E(m_1) \rangle$  if  $F_0: u > v$  is chosen. If  $F_1: u < v$  is chosen,  $C_B$  obtains  $\langle m_1, m_2 \rangle$  by decrypting  $\langle E(m_1), E(m_2) \rangle$ . 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

1<sup>st</sup> and 2<sup>nd</sup> ADD gates, respectively. If  $F_I: u < v$  is selected,  $C_A$  puts  $-r_u$  and  $-r_v$  into the 1<sup>st</sup> and 2<sup>nd</sup> ADD gates. Fifth, if  $F_O: u > v$  is selected,  $C_B$  puts  $m_2$  and  $m_1$  into the 1<sup>st</sup> and 2<sup>nd</sup> ADD gates, respectively. If  $F_I: u < v$  is selected,  $C_B$  puts  $m_1$  and  $m_2$  into the 1<sup>st</sup> and 2<sup>nd</sup> ADD gates. Sixth, the 1<sup>st</sup> ADD gate adds two input values and puts the output  $result_1$  into CMP gate. Similarly, the 2<sup>nd</sup> 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  $\alpha$  and sends  $E(\alpha)$  to  $C_A$ . Finally, only when the selected functionality is  $F_O: u > v$ ,  $C_A$  computes  $E(\alpha) = \text{SBN}(E(\alpha))$  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  $E(\text{range})$  as inputs. Here, the *range* consists of the  $E(lb_j)$  and the  $E(ub_j)$  for  $1 \leq j \leq m$ . If  $E(p_j) \leq E(\text{range}.ub_j)$  and  $E(p_j) \geq E(\text{range}.lb_j)$ , 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  $1 \leq j \leq 2m$ .  $C_A$  obtains packed values  $RA$  and  $RB$  by packing  $ra_j$  and  $rb_j$ , respectively, using the following equation (2) for  $1 \leq j \leq 2m$ .

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

Here,  $\sigma$  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  $E(\omega_j) = E(\text{range}.lb_j)^2$  for  $1 \leq j \leq m$ .  $C_A$  also computes  $E(\delta_j) = E(p_j)^2 \times E(1)$  and  $E(\rho_j) = E(\text{range}.ub_j)^2 \times E(1)$  for  $1 \leq j \leq m$ . Third,  $C_A$  randomly selects one functionality between  $F_O: u > v$  and  $F_I: v > u$ . Then,  $C_A$  performs data packing by using the  $E(\mu_j)$  and  $E(\rho_j)$ , depending on the selected functionality.

- If  $F_O: u > v$  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  $F_I: v > u$  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  $F_O: u > v$  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  $F_I: v > u$  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  $1 \leq j \leq 2m$ . Here,  $u_j$  (or  $v_j$ ) is one of the  $\mu_j$ ,  $\rho_j$ ,  $\omega_j$ , and  $\delta_j$ . 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  $1 \leq j \leq 2m$ . 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  $\alpha_j'$  is returned to  $C_B$ . Then,  $C_B$  encrypts  $\alpha'$  and sends  $E(\alpha')$  to  $C_A$ . Sixth,  $C_A$  performs  $E(\alpha_j') = \text{SBN}(E(\alpha_j'))$  for  $1 \leq j \leq 2m$  only when the selected functionality is  $F_O: u > v$ . Then,  $C_A$  computes  $E(\alpha) = \text{SM}(E(\alpha), E(\alpha_j'))$  where the initial value of  $E(\alpha)$  is  $E(1)$ . Only when all of the  $E(\alpha_j')$  for  $1 \leq j \leq 2m$  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  $E(1)$ .

**SXS<sub>n</sub> protocol.** SXS<sub>n</sub> (Secure Maximum Selection) returns the maximum value among the inputs when  $E(d_i)$  for  $1 \leq i \leq n$  are given as inputs. SXS<sub>n</sub> can be realized by converting the logic of SMS<sub>n</sub> in opposite way. Therefore, we omit the detailed procedure of SXS<sub>n</sub> due to the space limitation.

#### IV. KNN CLASSIFICATION ALGORITHM

In this section, we present our  $k$ NN classification algorithm (SkNNC<sub>G</sub>) 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.

##### A. 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.

The procedure of the encrypted kd-tree search step is shown in Algorithm 1. First,  $C_A$  securely finds nodes which include a query by executing  $E(\alpha_z) \leftarrow \text{GSPE}(E(q), E(\text{node}_z))$  for  $1 \leq z \leq \text{numnode}$  where  $\text{numnode}$  means the total number of kd-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  $\pi$  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 Figure 2 because the  $q$  is given inside the  $\text{node}_1$ . Assuming that  $\pi$  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  $\alpha'$  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  $\text{numnode}/c - 1$  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.,  $NG_1' = \{2, 1, 3, 4\}$ .

---

**Algorithm 1: Encrypted kd-tree search**

---

**Input :**  $E(q)$ ,  $E(\text{node})$   
**Output :**  $E(\text{cand})$  // all the data inside nodes related to a query  
**C<sub>A</sub> : 01:** for  $1 \leq z \leq \text{num}_{\text{node}}$  //  $\text{num}_{\text{node}} = 2^{h-1}$  ( $h$  : level of the kd-tree)  
    **02:**  $E(\alpha_z) \leftarrow \text{GSPE}(E(q), E(\text{node}_z))$   
    **03:**  $E(\alpha') \leftarrow \pi(E(\alpha_z))$ ; send  $E(\alpha')$  to  $C_B$   
**C<sub>B</sub> : 04:**  $\alpha' \leftarrow D(E(\alpha'))$ ;  $c \leftarrow$  the number of '1' in  $\alpha'$   
    **05:** create  $c$  number of Group  
    **06:** for each  $NG$   
    **07:** assign a node with  $\alpha'=1$  and  $\text{num}_{\text{node}}/c-1$  nodes with  $\alpha'=0$   
    **08:**  $NG' \leftarrow$  shuffle the  $ids$  of nodes  
    **09:** send  $NG'$  to  $C_A$   
**C<sub>A</sub> : 10:**  $\text{cnt} \leftarrow 0$   
    **11:**  $NG^* \leftarrow$  permute node  $ids$  using  $\pi^{-1}$  for each  $NG'$   
    **12:** for each  $NG^*$   
    **13:** for  $1 \leq s \leq F$   
    **14:** for  $1 \leq i \leq \text{num}$  (# nodes in the selected  $NG^*$ )  
    **15:**  $z = id$  of  $i^{\text{th}}$  node of  $NG^*$   
    **16:**  $E(t'_{i,j}) \leftarrow \text{SM}(\text{node}_{z,t_{s,j}}, E(\alpha_z))$  for  $1 \leq j \leq m+1$   
    **17:** for  $1 \leq j \leq m+1$   
    **18:**  $E(\text{cand}_{\text{cnt},j}) \leftarrow \prod_{i=1}^{\text{num}} E(t'_{i,j})$   
    **19:**  $\text{cnt} \leftarrow \text{cnt}+1$   
    **20:** return  $E(\text{cand})$

---

Fifth,  $C_A$  obtains  $NG^*$  by permuting the  $ids$  of nodes using  $\pi^{-1}$  in each  $NG'$  (line 11). Six,  $C_A$  gets access to one datum in a node for each  $NG^*$  and executes  $E(t'_{i,j}) = \text{SM}(E(\text{node}_{z,t_{s,j}}), E(\alpha_z))$  for  $1 \leq s \leq F$  and  $1 \leq j \leq m+1$  where  $E(\alpha_z)$  is the result of GSPE corresponding to  $\text{node}_z$  (line 12~16). As a result, SM results in  $E(\text{node}_{z,t_{s,j}})$  only for the data inside the nodes related to the query because their  $E(\alpha_z)$  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  $E(\text{max})$ , instead of using  $E(\text{node}_{z,t_{s,j}})$ . Here,  $E(\text{max})$  is the largest value in the domain. When  $C_A$  accesses one datum from every node in a  $NG^*$ ,  $C_A$  performs  $E(\text{cand}_{\text{cnt},j}) \leftarrow \prod_{i=1}^{\text{num}} E(t'_{i,j})$  where  $\text{num}$  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  $E(\text{cand}_{\text{cnt},j})$  for  $1 \leq i \leq \text{cnt}$  and  $1 \leq j \leq m+1$  where  $\text{cnt}$  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  $\pi^{-1}$ .  $C_A$  gains access to  $E(t_5)$  in  $\text{node}_3$ ,  $E(t_7)$  in  $\text{node}_4$ ,  $E(t_3)$  in  $\text{node}_2$ , and  $E(t_1)$  in  $\text{node}_1$ . 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  $E(t'_1)$ ,  $E(t'_2)$  and  $E(t'_3)$ , respectively. However, the results of SM using  $E(t_1)$  become  $\{E(2), E(1), E(1)\}$  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  $E(t'_4)$ . Thus, the final attribute-wise homomorphic addition of  $E(t'_i)$  for  $1 \leq i \leq 4$  are  $\{E(2), E(1), E(1)\}$ . Accordingly, one datum  $E(t_1)$  in  $\text{node}_1$  is securely extracted. By repeating this, the encrypted kd-tree search step can extract all of the data in  $\text{node}_1$  (e.g.,  $E(t_1)$  and  $E(t_2)$ ) and finally stores them into  $E(\text{cand})$ .

**B. Step 2 :  $k$ NN retrieval step**

In the  $k$ NN retrieval phase, we retrieve the  $k$  closest data from the query by partially utilizing the  $SkNN_m$  scheme [4].

However, we only consider  $E(\text{cand}_i)$  for  $1 \leq i \leq \text{cnt}$ , which are extracted in the index search phase, whereas the  $SkNN_m$  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 :**  $E(q)$ ,  $E(\text{cand})$ ,  $k$   
**Output :**  $E(t')$  // candidate  $k$ NN result  
**C<sub>A</sub> : 01:** for  $1 \leq i \leq \text{cnt}$   
    **02:**  $E(d_i) \leftarrow \text{ESSED}(E(q), E(\text{cand}_i))$   
    **03:** for  $1 \leq s \leq k$   
    **04:**  $E(d_{\min}) \leftarrow \text{SMS}_n(E(d_1), \dots, E(d_{\text{cnt}}))$   
    **05:** for  $1 \leq i \leq \text{cnt}$   
        **06:**  $E(\tau_i) \leftarrow E(d_{\min}) \times E(d_i)^{N-1}$ ;  $E(\tau'_i) \leftarrow E(\tau_i)^{r_i}$   
    **07:**  $E(\beta) \leftarrow \pi(\tau)$ ; send  $E(\beta)$  to  $C_B$   
**C<sub>B</sub> : 08:** for  $1 \leq i \leq \text{cnt}$   
    **09:** if  $D(\beta_i)=0$  then  $E(U_i) \leftarrow E(1)$ ; else  $E(U_i) \leftarrow E(0)$   
    **10:** send  $E(U)$  to  $C_A$   
**C<sub>A</sub> : 11:**  $E(V) \leftarrow \pi^{-1}(U)$   
    **12:** for  $1 \leq i \leq \text{cnt}$   
    **13:**  $E(V'_{i,j}) \leftarrow \text{SM}(E(V_i), E(\text{cand}_{i,j}))$  for  $1 \leq j \leq m+1$   
    **14:**  $E(t'_{s,j}) = \prod_{i=1}^{\text{cnt}} E(V'_{i,j})$  for  $1 \leq j \leq m+1$   
    **15:** if  $s < k$   
    **16:**  $E(d_i) \leftarrow \text{SM}(E(V_i), E(\text{max})) \times \text{SM}(SBN(E(V_i)), E(d_i))$   
    **17:** return  $E(t')$

---

First, using our proposed ESSED,  $C_A$  securely calculates the squared Euclidean distances  $E(d_i)$  between a query and  $E(\text{cand}_i)$  for  $1 \leq i \leq \text{cnt}$  (lines 1-2). Then, instead of using the inefficient  $\text{SMIN}_n$ ,  $C_A$  performs  $\text{SMS}_n$  to find the minimum value  $E(d_{\min})$  among  $E(d_i)$  for  $1 \leq i \leq \text{cnt}$ . Second,  $C_A$  calculates  $E(\tau_i) = E(d_{\min}) \times E(d_i)^{N-1}$ , i.e., the difference between the  $E(d_{\min})$  and  $E(d_i)$ , for  $1 \leq i \leq \text{cnt}$ . Then,  $C_A$  computes  $E(\tau'_i) = E(\tau_i)^{r_i}$  (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  $\pi$  and then sends  $E(\beta)$  to the  $C_B$  (line 7). For example, because  $E(\text{cand}) = \{E(t_1), E(t_2)\}$  is given from the index search phase,  $E(d_1) = E(4)$  and  $E(d_2) = E(5)$ . By performing  $\text{SMS}_n$ ,  $E(d_{\min})$  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  $\pi$  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  $\pi^{-1}$  (line 8-11). Then,  $C_A$  performs SM protocol by using  $E(V_i)$  and  $E(\text{cand}_{i,j})$  to obtain  $E(V'_{i,j})$ . By computing  $E(t'_{s,j}) = \prod_{i=1}^{\text{cnt}} E(V'_{i,j})$  for  $1 \leq j \leq m+1$ ,  $C_A$  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  $\pi^{-1}$ . For the  $x$ -attribute,  $C_A$  performs  $\text{SM}(E(\text{cand}_{1,1}), E(V_1)) = E(2)$  and  $\text{SM}(E(\text{cand}_{2,1}), E(V_2)) = E(0)$ . By adding the two values, the  $x$ -attribute value of  $E(t_1)$ , i.e.,  $E(2)$ , is securely calculated. Similarly, we can compute  $E(1)$ , the  $y$ -attribute value of  $E(t_1)$ . Therefore, we can store  $E(t_1) = \{E(2), E(1)\}$  into  $E(t'_1)$  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  $E(\text{max})$  by performing  $E(d_i) = \text{SM}(E(V_i),$

$E(max)) \times SM(SBN(E(V_i)), E(d_i))$  (line 15-16). This procedure is repeated for  $k$  rounds to find the  $k$ NN 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.

### C. 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.

The procedure of the result verification step is shown in Algorithm 3. First,  $C_A$  computes  $E(dist_k) = ESSED(E(q), E(t'_k))$  to calculate the squared Euclidean distance between the query and the  $k^{th}$  closest result among  $E(t')$ , i.e., the output of the  $k$ NN retrieval step (line 1). Second,  $C_A$  performs GSCMP by using  $E(q_j)$  and  $E(node_z.lb_j)$  for  $1 \leq z \leq num_{node}$  and  $1 \leq j \leq m$  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  $1 \leq z \leq num_{node}$  and  $1 \leq j \leq m$  and then stores the result into  $E(\psi_2)$ . In addition,  $C_A$  calculates  $E(\psi_3)$  by executing  $E(\psi_1) \times E(\psi_2) \times SM(E(\psi_1), E(\psi_2))^{N-2}$  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  $SM(E(\psi_3), E(q_j)) \times SM(SBN(E(\psi_3)), f(E(lb_{z,j}), E(ub_{z,j})))$  for  $1 \leq z \leq num_{node}$  and  $1 \leq j \leq m$ , where  $f(E(lb_j), E(ub_j))$  means  $SM(E(\psi_1), E(lb_{z,j})) \times SM(SBN(E(\psi_1)), E(ub_{z,j}))$  (lines 7-10). For example, assuming that the required  $k$  is 2,  $E(dist_2) = E(5)$  because  $E(t_2)$  is the current 2NN. Meanwhile, in Figure 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  $E(sp_{3,2}) = E(1)$ .

---

### Algorithm 3: Result verification

---

**Input** :  $E(q), E(node), E(t'), k$

**Output** :  $E(c)$

**$C_A$**  : **01**:  $E(dist_k) = ESSED(E(q), E(t'_k))$

**02**: for  $1 \leq z \leq num_{node}$

**03**: for  $1 \leq j \leq m$

**04**:  $E(\psi_1) \leftarrow GSCMP(E(q_j), E(node_z.lb_j))$

**05**:  $E(\psi_2) \leftarrow GSCMP(E(q_j), E(node_z.ub_j))$

**06**:  $E(\psi_3) \leftarrow E(\psi_1) \times E(\psi_2) \times SM(E(\psi_1), E(\psi_2))^{N-2}$  // bit-xor

**07**:  $E(temp) \leftarrow SM(E(\psi_1), E(node_z.lb_j))$

**08**:  $E(temp) \leftarrow E(temp) \times SM(SBN(E(\psi_1)), E(node_z.ub_j))$

**09**:  $E(temp) \leftarrow SM(E(temp), SBN(E(\psi_3)))$

**10**:  $E(sp_{z,j}) \leftarrow E(temp) \times SM(E(\psi_3), E(q_j))$

**11**:  $E(spdist_z) \leftarrow ESSED(E(q), E(sp_z))$

**12**:  $E(spdist_z) \leftarrow SM(E(\alpha_z), E(max)) \times SM(SBN(E(\alpha_z)), E(spdist_z))$

**13**:  $E(\alpha_z) \leftarrow GSCMP(E(spdist_z), E(dist_k))$

**14**:  $E(t'') \leftarrow$  perform 4~20 lines of Algorithm 1

**15**:  $E(t') \leftarrow$  append the  $E(t'')$  to  $E(t')$

**16**:  $E(result) \leftarrow$  perform Algorithm 2

**17**: for  $1 \leq i \leq k$

**18**:  $E(c_i) \leftarrow E(result_{i,m+1})$

**19**: return  $E(c)$

---

Fourth,  $C_A$  calculates  $E(spdist_z)$ , the squared Euclidean distances between the query and  $E(sp_z)$  for  $1 \leq z \leq num_{node}$  by using ESSED. In addition,  $C_A$  securely updates the  $E(spdist_z)$  of the retrieved nodes into  $E(max)$  by computing  $E(spdist_z) = SM(E(\alpha_z), E(max)) \times SM(SBN(E(\alpha_z)), E(spdist_z))$  (lines 11-12). Here,  $E(\alpha_z)$  is the output of GSPE computed in index search step. Then,  $C_A$  performs  $E(\alpha_z) = GSCMP(E(spdist_z), E(dist_k))$  (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 Figure 2, and  $E(spdist)$  is updated as  $(E(max), E(16), E(1), E(26))$ . Therefore, the result of GSCMP becomes  $E(\alpha) = (E(0), E(0), E(1), E(0))$  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  $1 \leq i \leq k$  (lines 14-16). Finally,  $C_A$  stores  $E(result_{i,m+1})$  into  $E(c_i)$  for  $1 \leq i \leq k$  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 Figure 2, the final  $E(c)$  becomes  $(E(1), E(1))$ .

### D. 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  $1 \leq j \leq w$  and  $E(c_i)$  for  $1 \leq i \leq k$  to obtain  $E(f(label_j))$ . Then,  $C_A$  finds the maximum value, i.e.,  $E(f_{max})$ , among  $E(f(label_j))$  for  $1 \leq j \leq w$  by using SXS<sub>n</sub> (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  $E(\tau_i) = E(f_{max}) \times E(f(label_j))^{N-1}$  for  $1 \leq i \leq w$ . Then,  $C_A$  computes  $E(\tau_i) = E(\tau_i)^{r_i}$  and obtains  $E(\beta)$  by shuffling  $E(\tau)$  by using  $\pi$  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  $\pi^{-1}$  (line 6-9). Then,  $C_A$  performs  $E(output) = \prod_{j=1}^w SM(E(V_j), E(label_j))$  for  $1 \leq j \leq w$  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  $E(f(label)) = (E(2), E(0), E(0))$ . 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:**  $E(f(label)) \leftarrow SF(E(label), E(c))$   
**02:**  $E(f_{max}) \leftarrow SXS_n(E(f(label_1)), \dots, E(f(label_w)))$   
**03:** for  $1 \leq j \leq w$   
**04:**  $E(\tau_j) \leftarrow E(f_{max}) \times E(f(label_j))^{w-1}$ ;  $E(\tau'_j) \leftarrow E(\tau_j)^{r_j}$ .  
**05:**  $E(\beta) \leftarrow \pi(\tau'_j)$ ; send  $E(\beta)$  to  $C_B$   
 $C_B$  : **06:** for  $1 \leq j \leq w$   
**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  $1 \leq j \leq k$   
**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  $output$

V. PERFORMANCE ANALYSIS

In this section, we compare our  $SkNNC_G$  (secure  $kNN$  classification algorithm using the Yao’s garbled circuit) with  $PPkNN$  [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  $SkNNC_I$  (secure  $kNN$  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  $SkNNC_I$  and  $PPkNN$  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 1. 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 3.10GHz and 32GB 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.

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

Table 1. Experimental parameters

In Figure 3, we measure the performance of  $SkNNC_I$  and our  $SkNNC_G$  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  $SkNNC_G$  and  $SkNNC_I$ , respectively. Whereas, as  $h$  increases, the number of data in the node decreases, thus requiring less computation cost for distance calculation. However, our  $SkNNC_G$  outperforms  $SkNNC_I$  because our scheme uses both efficient secure protocols based on the Yao’s garbled circuit and the data packing technique.

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  $SkNNC_I$  and  $SkNNC_G$  are increased as the  $n$  increases, they are less affected by  $n$  than  $PPkNN$ . Overall, our  $SkNNC_G$  shows 17.1 and 4.7 times better performance than  $PPkNN$  and  $SkNNC_I$ , respectively. Due to the index-based data filtering, both  $SkNNC_G$  and  $SkNNC_I$  shows better performance than  $PPkNN$ . However, our  $SkNNC_G$  outperforms  $SkNNC_I$  because our algorithm can reduce the computation cost by using the Yao’s garbled circuit and the data packing technique.

Fig. 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  $SkNNC_G$  shows 17.7 and 4.2 times better performance than  $PPkNN$  and  $SkNNC_I$ , respectively, due to the same reasons described for Figure 3.

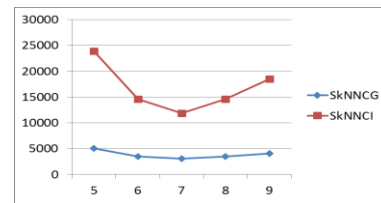
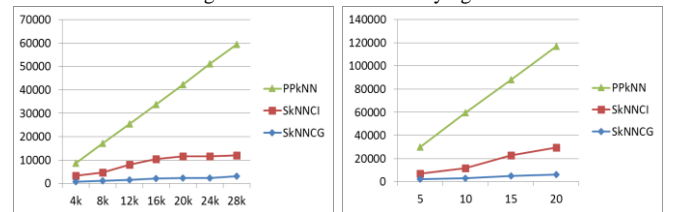


Figure 3. Performance of varying  $h$



(a) Performance of varying  $n$  (b) Performance of varying  $k$   
Figure 4. Classification time for varying  $n$



## VI. CONCLUSION

Databases need to be encrypted before being outsourced to the cloud, due to its privacy issues. However, the existing  $k$ NN 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  $k$ NN 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  $k$ NN 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 PP $k$ NN 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.

## REFERENCES

- [1] S. Vimercati, S. Foresti, P. Samarati, Managing and accessing data in the cloud: Privacy risks and approaches, in CRiSIS, 2012, pp. 1-9.
- [2] J. W. Riley, C. Alfons, E. Fredäng, P. Lind, Nearest Neighbor Classifiers, 2009.
- [3] B. Samanthula, Y. Elmehdwi, W. Jiang, K-nearest neighbor classification over semantically secure encrypted relational data, TKDE, vol. 27, no. 5, 2015, pp. 1261-1273.
- [4] Y. Elmehdwi, B. K. Samanthula, W. Jiang, Secure k-nearest neighbor query over encrypted data in outsourced environments, in ICDE, 2014, pp. 664-675.
- [5] H. Hu, J. Xu, C. Ren, B. Choi, Processing private queries over untrusted data cloud through privacy homomorphism, in ICDE, 2011, pp. 601-612.
- [6] 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.
- [7] H. Kim, H. Kim, J. Chang, A  $k$ NN query processing algorithm using a tree index structure on the encrypted database, in Big Data and Smart Computing (BigComp), 2016, pp. 93-100.
- [8] A. C. C. Yao, How to Generate and Exchange Secrets, in Foundations of Computer Science, 1986, pp. 162-167.
- [9] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in EUROCRYPT, 1999, pp. 223-238.
- [10] A. Liu, K. Ahengy, L. Liz, G. Liu, L. Zhao, X. Zhou, Efficient Secure Similarity Computation on Encrypted Trajectory Data, in ICDE, 2015, pp. 66-77.
- [11] Y. Huang, D. Evans, J. Katz, L. Malka, Faster Secure Two-Party Computation Using Garbled Circuits, USENIX Security, vol. 201, no. 1, 2011.
- [12] J. Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism, in ISC, 2002, pp. 471-483.
- [13] B. K. Samanthula, H. Chun, W. Jiang, An efficient and probabilistic secure bit-decomposition, in ASIACCS, 2013, pp. 541-546.
- [14] Hanan Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann, 2006.
- [15] <http://archive.ics.uci.edu/ml/>