k-Nearest Neighbor Classification over Semantically Secure Encrypted Relational Data

Bharath K. Samanthula, *Member, IEEE*, Yousef Elmehdwi, and Wei Jiang, *Member, IEEE*

Abstract—Data Mining has wide applications in many areas such as banking, medicine, scientific research and among government agencies. Classification is one of the commonly used tasks in data mining applications. For the past decade, due to the rise of various privacy issues, many theoretical and practical solutions to the classification problem have been proposed under different security models. However, with the recent popularity of cloud computing, users now have the opportunity to outsource their data, in encrypted form, as well as the data mining tasks to the cloud. Since the data on the cloud is in encrypted form, existing privacy-preserving classification techniques are not applicable. In this paper, we focus on solving the classification problem over encrypted data. In particular, we propose a secure k-NN classifier over encrypted data in the cloud. The proposed protocol protects the confidentiality of data, privacy of user's input query, and hides the data access patterns. To the best of our knowledge, our work is the first to develop a secure k-NN classifier over encrypted data under the semi-honest model. Also, we empirically analyze the efficiency of our proposed protocol using a real-world dataset under different parameter settings.

Index Terms—Security, *k*-NN classifier, outsourced databases, encryption

1 Introduction

Recently, the cloud computing paradigm [1] is revolutionizing the organizations' way of operating their data particularly in the way they store, access and process data. As an emerging computing paradigm, cloud computing attracts many organizations to consider seriously regarding cloud potential in terms of its cost-efficiency, flexibility, and offload of administrative overhead. Most often, organizations delegate their computational operations in addition to their data to the cloud. Despite tremendous advantages that the cloud offers, privacy and security issues in the cloud are preventing companies to utilize those advantages. When data are highly sensitive, the data need to be encrypted before outsourcing to the cloud. However, when data are encrypted, irrespective of the underlying encryption scheme, performing any data mining tasks becomes very challenging without ever decrypting the data. There are other privacy concerns, demonstrated by the following example.

Example 1. Suppose an insurance company outsourced its encrypted customers database and relevant data mining tasks to a cloud. When an agent from the company wants to determine the risk level of a potential new customer, the agent can use a classification method to determine the risk level of the customer. First, the agent needs to generate a data record q for the customer containing certain personal information of the customer, e.g., credit score, age, marital status, etc. Then this record can be sent to the cloud, and the cloud will compute the class label for q. Nevertheless, since q contains sensitive information, to protect the customer's privacy, q should be encrypted before sending it to the cloud.

The above example shows that data mining over encrypted data (denoted by DMED) on a cloud also needs to protect a user's record when the record is a part of a data mining process. Moreover, cloud can also derive useful and sensitive information about the actual data items by observing the data access patterns even if the data are encrypted [2], [3]. Therefore, the privacy/security requirements of the DMED problem on a cloud are threefold: (1) confidentiality of the encrypted data, (2) confidentiality of a user's query record, and (3) hiding data access patterns.

Existing work on privacy-preserving data mining (PPDM) (either perturbation or secure multi-party computation (SMC) based approach) cannot solve the DMED problem. Perturbed data do not possess semantic security, so data perturbation techniques cannot be used to encrypt highly sensitive data. Also the perturbed data do not produce very accurate data mining results. Secure multi-party computation based approach assumes data are distributed and not encrypted at each participating party. In addition, many intermediate computations are performed based on non-encrypted data. As a result, in this paper, we proposed novel methods to effectively solve the DMED problem assuming that the encrypted data are outsourced to a cloud. Specifically, we focus on the classification problem since it is one of the most common data mining tasks. Because each classification technique has their own advantage, to be concrete, this paper concentrates on executing the k-nearest neighbor classification method over encrypted data in the cloud computing environment.

- *B.K. Samanthula is with the Department of Computer Science, Purdue University, 305 N. University Street, West Lafayette, IN 47907. E-mail: bsamanth@purdue.edu.*
- *Y. Elmehdwi and W. Jiang are with the Department of Computer Science, Missouri University of Science and Technology, 310 CS Building, 500 West 15th St., Rolla, MO 65409. E-mail: {ymez76, wjiang}@mst.edu.*

Manuscript received 23 Oct. 2013; revised 10 Sept. 2014; accepted 29 Sept. 2014. Date of publication 19 Oct. 2014; date of current version 27 Mar. 2015. Recommended for acceptance by G. Miklau.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2364027

1.1 Problem Definition

Suppose Alice owns a database D of n records $t_1, \ldots, t_n$ and $m+1$ attributes. Let $t_{i,j}$ denote the jth attribute value of record t_i. Initially, Alice encrypts her database attribute-wise, that is, she computes $E_{pk}(t_{i,j})$, for $1 \leq i \leq n$ and $1 \leq j \leq m+1$, where column $(m+1)$ contains the class labels. We assume that the underlying encryption scheme is semantically secure [4]. Let the encrypted database be denoted by D'. We assume that Alice outsources D' as well as the future classification process to the cloud.

Let Bob be an authorized user who wants to classify his input record $q = \langle q_1, \ldots, q_m \rangle$ by applying the k-NN classification method based on D'. We refer to such a process as privacy-preserving k-NN (PPkNN) classification over encrypted data in the cloud. Formally, we define the PPkNN protocol as:

$$\text{PP}k\text{NN}(D', q) \rightarrow c_q,$$

where c_q denotes the class label for q after applying k-NN classification method on D' and q.

1.2 Our Contributions

In this paper, we propose a novel PPkNN protocol, a secure k-NN classifier over semantically secure encrypted data. In our protocol, once the encrypted data are outsourced to the cloud, Alice does not participate in any computations. Therefore, no information is revealed to Alice. In addition, our protocol meets the following privacy requirements:

- Contents of D or any intermediate results should not be revealed to the cloud.
- Bob's query q should not be revealed to the cloud.
- c_q should be revealed only to Bob. Also, no other information should be revealed to Bob.
- Data access patterns, such as the records corresponding to the k-nearest neighbors of q, should not be revealed to Bob and the cloud (to prevent any inference attacks).

We emphasize that the intermediate results seen by the cloud in our protocol are either newly generated randomized encryptions or random numbers. Thus, which data records correspond to the k-nearest neighbors and the output class label are not known to the cloud. In addition, after sending his encrypted query record to the cloud, Bob does not involve in any computations. Hence, data access patterns are further protected from Bob (see Section 5 for more details).

The rest of the paper is organized as follows. We discuss the existing related work and some concepts as a background in Section 2. A set of privacy-preserving protocols and their possible implementations are provided in Section 3. The formal security proofs for the mentioned privacy-preserving primitives are provided in Section 4. The proposed PPkNN protocol is explained in detail in Section 5. Section 6 discusses the performance of the proposed protocol under different parameter settings. We conclude the paper along with future work in Section 7.

2 Related Work and Background

Due to space limitations, here we briefly review the existing related work and provide some definitions as a background. Please refer to our technical report [5] for a more elaborated related work and background.

At first, it seems fully homomorphic cryptosystems (e.g., [6]) can solve the DMED problem since it allows a third-party (that hosts the encrypted data) to execute arbitrary functions over encrypted data without ever decrypting them. However, we stress that such techniques are very expensive and their usage in practical applications have yet to be explored. For example, it was shown in [7] that even for weak security parameters one "bootstrapping" operation of the homomorphic operation would take at least 30 seconds on a high performance machine.

It is possible to use the existing secret sharing techniques in SMC, such as Shamir's scheme [8], to develop a PPkNN protocol. However, our work is different from the secret sharing based solution in the following aspect. Solutions based on the secret sharing schemes require at least three parties whereas our work require only two parties. For example, the constructions based on Sharemind [9], a well-known SMC framework which is based on the secret sharing scheme, assumes that the number of participating parties is three. Thus, our work is orthogonal to Sharemind and other secret sharing based schemes.

2.1 Privacy-Preserving Data Mining

Agrawal and Srikant [10], Lindell and Pinkas [11] were the first to introduce the notion of privacy-preserving under data mining applications. The existing PPDM techniques can broadly be classified into two categories: (i) data perturbation and (ii) data distribution. Agrawal and Srikant [10] proposed the first data perturbation technique to build a decision-tree classifier, and many other methods were proposed later (e.g., [12], [13], [14]). However, as mentioned earlier in Section 1, data perturbation techniques cannot be applicable for semantically secure encrypted data. Also, they do not produce accurate data mining results due to the addition of statistical noises to the data. On the other hand, Lindell and Pinkas [11] proposed the first decision tree classifier under the two-party setting assuming the data were distributed between them. Since then much work has been published using SMC techniques (e.g., [15], [16], [17]). We claim that the PPkNN problem cannot be solved using the data distribution techniques since the data in our case is encrypted and not distributed in plaintext among multiple parties. For the same reasons, we also do not consider secure k-NN methods in which the data are distributed between two parties (e.g., [18]).

2.2 Query Processing over Encrypted Data

Various techniques related to query processing over encrypted data have been proposed, e.g., [19], [20], [21]. However, we observe that PPkNN is a more complex problem than the execution of simple kNN queries over encrypted data [22], [23]. For one, the intermediate k-nearest neighbors in the classification process, should not be disclosed to the cloud or any users. We emphasize that the recent method in [23] reveals the k-nearest neighbors to the user. Second, even if we know the k-nearest neighbors, it is still very difficult to find the majority class label among these neighbors since they are encrypted at the first place to

prevent the cloud from learning sensitive information. Third, the existing work did not addressed the access pattern issue which is a crucial privacy requirement from the user's perspective.

In our most recent work [24], we proposed a novel secure k-nearest neighbor query protocol over encrypted data that protects data confidentiality, user's query privacy, and hides data access patterns. However, as mentioned above, PPkNN is a more complex problem and it cannot be solved directly using the existing secure k-nearest neighbor techniques over encrypted data. Therefore, in this paper, we extend our previous work in [24] and provide a new solution to the PPkNN classifier problem over encrypted data.

More specifically, this paper is different from our preliminary work [24] in the following four aspects. First, in this paper, we introduced new security primitives, namely secure minimum (SMIN), secure minimum out of n numbers (SMIN$_n$), secure frequency (SF), and proposed new solutions for them. Second, the work in [24] did not provide any formal security analysis of the underlying sub-protocols. On the other hand, this paper provides formal security proofs of the underlying sub-protocols as well as the PPkNN protocol under the semi-honest model. Additionally, we discuss various techniques through which the proposed PPkNN protocol can possibly be extended to a protocol that is secure under the malicious setting. Third, our preliminary work in [24] addresses only secure kNN query which is similar to Stage 1 of PPkNN. However, Stage 2 in PPkNN is entirely new. Finally, our empirical analyses in Section 6 are based on a real dataset whereas the results in [24] are based on a simulated dataset. Furthermore, new experimental results are included in this paper.

2.3 Threat Model

We adopt the security definitions in the literature of secure multi-party computation [25], [26], and there are three common adversarial models under SMC: semi-honest, covert and malicious. In this paper, to develop secure and efficient protocols, we assume that parties are semi-honest. Briefly, the following definition captures the properties of a secure protocol under the semi-honest model [27], [28].

Definition 1. *Let a_i be the input of party P_i, $\Pi_i(\pi)$ be P_i's execution image of the protocol π and b_i be the output for party P_i computed from π. Then, π is secure if $\Pi_i(\pi)$ can be simulated from a_i and b_i such that distribution of the simulated image is computationally indistinguishable from $\Pi_i(\pi)$.*

In the above definition, an execution image generally includes the input, the output and the messages communicated during an execution of a protocol. To prove a protocol is secure under semi-honest model, we generally need to show that the execution image of a protocol does not leak any information regarding the private inputs of participating parties [28].

2.4 Paillier Cryptosystem

The Paillier cryptosystem is an additive homomorphic and probabilistic public-key encryption scheme whose security is based on the Decisional Composite Residuosity Assumption [4]. Let E_{pk} be the encryption function with public key pk given by (N, g), where N is a product of two large primes of similar bit length and g is a generator in $\mathbb{Z}^*_{N^2}$. Also, let D_{sk} be the decryption function with secret key sk. For any given two plaintexts $a, b \in \mathbb{Z}_N$, the Paillier encryption scheme exhibits the following properties:

(1) *Homomorphic addition.*

$$D_{sk}(E_{pk}(a+b)) = D_{sk}(E_{pk}(a) * E_{pk}(b) \bmod N^2).$$

(2) *Homomorphic multiplication.*

$$D_{sk}(E_{pk}(a*b)) = D_{sk}(E_{pk}(a)^b \bmod N^2).$$

(3) *Semantic security.* The encryption scheme is semantically secure[28], [29]. Briefly, given a set of ciphertexts, an adversary cannot deduce any additional information about the plaintext(s).

For succinctness, we drop the $\bmod N^2$ term during homomorphic operations in the rest of this paper.

3 Privacy-Preserving Primitives

Here we present a set of generic sub-protocols that will be used in constructing our proposed k-NN protocol in Section 5. All of the below protocols are considered under two-party semi-honest setting. In particular, we assume the existence of two semi-honest parties P_1 and P_2 such that the Paillier's secret key sk is known only to P_2 whereas pk is public.

- *Secure multiplication (SM).* This protocol considers P_1 with input $(E_{pk}(a), E_{pk}(b))$ and outputs $E_{pk}(a*b)$ to P_1, where a and b are not known to P_1 and P_2. During this process, no information regarding a and b is revealed to P_1 and P_2.
- *Secure squared euclidean distance (SSED).* In this protocol, P_1 with input $(E_{pk}(X), E_{pk}(Y))$ and P_2 with sk securely compute the encryption of squared euclidean distance between vectors X and Y. Here X and Y are m dimensional vectors where $E_{pk}(X) = \langle E_{pk}(x_1), \ldots, E_{pk}(x_m)\rangle$ and $E_{pk}(Y) = \langle E_{pk}(y_1), \ldots, E_{pk}(y_m)\rangle$. The output $E_{pk}(|X-Y|^2)$ will be known only to P_1.
- *Secure bit-decomposition (SBD).* Here P_1 with input $E_{pk}(z)$ and P_2 securely compute the encryptions of the individual bits of z, where $0 \leq z < 2^l$. The output $[z] = \langle E_{pk}(z_1), \ldots, E_{pk}(z_l)\rangle$ is known only to P_1. Here z_1 and z_l are the most and least significant bits of integer z, respectively.
- *Secure minimum.* In this protocol, P_1 holds private input (u', v') and P_2 holds sk, where $u' = ([u], E_{pk}(s_u))$ and $v' = ([v], E_{pk}(s_v))$. Here s_u (resp., s_v) denotes the secret associated with u (resp., v). The goal of SMIN is for P_1 and P_2 to jointly compute the encryptions of the individual bits of minimum number between u and v. In addition, they compute $E_{pk}(s_{\min(u,v)})$. That is, the output is $([\min(u,v)], E_{pk}(s_{\min(u,v)}))$ which will be known only to P_1.

During this protocol, no information regarding the contents of u, v, s_u, and s_v is revealed to P_1 and P_2.

- *Secure minimum out of n numbers*. In this protocol, we consider P_1 with n encrypted vectors $([d_1], \ldots, [d_n])$ along with their respective encrypted secrets and P_2 with sk. Here $[d_i] = \langle E_{pk}(d_{i,1}), \ldots, E_{pk}(d_{i,l})\rangle$ where $d_{i,1}$ and $d_{i,l}$ are the most and least significant bits of integer d_i respectively, for $1 \leq i \leq n$. The secret of d_i is given by s_{d_i}. P_1 and P_2 jointly compute $[\min(d_1, \ldots, d_n)]$. In addition, they compute $E_{pk}(s_{\min(d_1,\ldots,d_n)})$. At the end of this protocol, the output $([\min(d_1, \ldots, d_n)], E_{pk}(s_{\min(d_1,\ldots,d_n)}))$ is known only to P_1. During SMIN$_n$, no information regarding any of d_i's and their secrets is revealed to P_1 and P_2.
- *Secure Bit-OR (SBOR)*. P_1 with input $(E_{pk}(o_1), E_{pk}(o_2))$ and P_2 securely compute $E_{pk}(o_1 \vee o_2)$, where o_1 and o_2 are 2 bits. The output $E_{pk}(o_1 \vee o_2)$ is known only to P_1.
- *Secure frequency*. Here P_1 with private input $(\langle E_{pk}(c_1), \ldots E_{pk}(c_w)\rangle, \langle E_{pk}(c'_1), \ldots, E_{pk}(c'_k)\rangle)$ and P_2 securely compute the encryption of the frequency of c_j, denoted by $f(c_j)$, in the list $\langle c'_1, \ldots, c'_k\rangle$, for $1 \leq j \leq w$. Here we explicitly assume that c_j's are unique and $c'_i \in \{c_1, \ldots, c_w\}$, for $1 \leq i \leq k$. The output $\langle E_{pk}(f(c_1)), \ldots, E_{pk}(f(c_w))\rangle$ will be known only to P_1. During the SF protocol, no information regarding c'_i, c_j, and $f(c_j)$ is revealed to P_1 and P_2, for $1 \leq i \leq k$ and $1 \leq j \leq w$.

Now we either propose a new solution or refer to the most efficient known implementation to each of the above protocols. First of all, efficient solutions to SM, SSED, SBD and SBOR were discussed in [24]. Therefore, in this paper, we discuss SMIN, SMIN$_n$, and SF problems in detail and propose new solutions to each one of them.

Secure minimum. In this protocol, we assume that P_1 holds private input (u', v') and P_2 holds sk, where $u' = ([u], E_{pk}(s_u))$ and $v' = ([v], E_{pk}(s_v))$. Here s_u and s_v denote the secrets corresponding to u and v, respectively. The main goal of SMIN is to securely compute the encryptions of the individual bits of $\min(u, v)$, denoted by $[\min(u, v)]$. Here $[u] = \langle E_{pk}(u_1), \ldots, E_{pk}(u_l)\rangle$ and $[v] = \langle E_{pk}(v_1), \ldots, E_{pk}(v_l)\rangle$, where u_1 (resp., v_1) and u_l (resp., v_l) are the most and least significant bits of u (resp., v), respectively. In addition, they compute $E_{pk}(s_{\min(u,v)})$, the encryption of the secret corresponding to the minimum value between u and v. At the end of SMIN, the output $([\min(u, v)], E_{pk}(s_{\min(u,v)}))$ is known only to P_1.

We assume that $0 \leq u, v < 2^l$ and propose a novel SMIN protocol. Our solution to SMIN is mainly motivated from the work of [24]. Precisely, the basic idea of the proposed SMIN protocol is for P_1 to randomly choose the functionality F (by flipping a coin), where F is either $u > v$ or $v > u$, and to obliviously execute F with P_2. Since F is randomly chosen and known only to P_1, the result of the functionality F is oblivious to P_2. Based on the comparison result and chosen F, P_1 computes $[\min(u, v)]$ and $E_{pk}(s_{\min(u,v)})$ locally using homomorphic properties.

Algorithm 1. SMIN$(u', v') \rightarrow [\min(u, v)], E_{pk}(s_{\min(u,v)})$

Require: P_1 has $u' = ([u], E_{pk}(s_u))$ and $v' = ([v], E_{pk}(s_v))$, where $0 \leq u, v < 2^l$; P_2 has sk

1: P_1:
- (a). Randomly choose the functionality F
- (b). **for** $i = 1$ to l **do**:
  - $E_{pk}(u_i * v_i) \leftarrow \text{SM}(E_{pk}(u_i), E_{pk}(v_i))$
  - $T_i \leftarrow E_{pk}(u_i \oplus v_i)$
  - $H_i \leftarrow H_{i-1}^{r_i} * T_i$; $r_i \in_R \mathbb{Z}_N$ and $H_0 = E_{pk}(0)$
  - $\Phi_i \leftarrow E_{pk}(-1) * H_i$
  - **if** $F : u > v$ **then**:
    - $W_i \leftarrow E_{pk}(u_i) * E_{pk}(u_i * v_i)^{N-1}$
    - $\Gamma_i \leftarrow E_{pk}(v_i - u_i) * E_{pk}(\hat{r}_i)$; $\hat{r}_i \in_R \mathbb{Z}_N$

    else
    - $W_i \leftarrow E_{pk}(v_i) * E_{pk}(u_i * v_i)^{N-1}$
    - $\Gamma_i \leftarrow E_{pk}(y_i - v_i) * E_{pk}(\hat{r}_i)$; $\hat{r}_i \in_R \mathbb{Z}_N$
  - $L_i \leftarrow W_i * \Phi_i^{r'_i}$; $r'_i \in_R \mathbb{Z}_N$
- (c). **if** $F : u > v$ **then**: $\delta \leftarrow E_{pk}(s_v - s_u) * E_{pk}(\overline{r})$
  else $\delta \leftarrow E_{pk}(s_u - s_v) * E_{pk}(\overline{r})$, where $\overline{r} \in_R \mathbb{Z}_N$
- (d). $\Gamma' \leftarrow \pi_1(\Gamma)$ and $L' \leftarrow \pi_2(L)$
- (e). Send δ, Γ' and L' to P_2

2: P_2:
- (a). Receive δ, Γ' and L' from P_1
- (b). Decryption: $M_i \leftarrow D_{sk}(L'_i)$, for $1 \leq i \leq l$
- (c). **if** $\exists\ j$ such that $M_j = 1$ **then** $\alpha \leftarrow 1$
  else $\alpha \leftarrow 0$
- (d). **if** $\alpha = 0$ **then**:
  - $M'_i \leftarrow E_{pk}(0)$, for $1 \leq i \leq l$
  - $\delta' \leftarrow E_{pk}(0)$

  else
  - $M'_i \leftarrow \Gamma'_i * r^N$, where $r \in_R \mathbb{Z}_N$ and is different for $1 \leq i \leq l$
  - $\delta' \leftarrow \delta * r_\delta^N$, where $r_\delta \in_R \mathbb{Z}_N$
- (e). Send $M', E_{pk}(\alpha)$ and δ' to P_1

3: P_1:
- (a). Receive $M', E_{pk}(\alpha)$ and δ' from P_2
- (b). $\widetilde{M} \leftarrow \pi_1^{-1}(M')$ and $\theta \leftarrow \delta' * E_{pk}(\alpha)^{N-\overline{r}}$
- (c). $\lambda_i \leftarrow \widetilde{M}_i * E_{pk}(\alpha)^{N-\hat{r}_i}$, for $1 \leq i \leq l$
- (d). **if** $F : u > v$ **then**:
  - $E_{pk}(s_{\min(u,v)}) \leftarrow E_{pk}(s_u) * \theta$
  - $E_{pk}(\min(u, v)_i) \leftarrow E_{pk}(u_i) * \lambda_i$, for $1 \leq i \leq l$

  else
  - $E_{pk}(s_{\min(u,v)}) \leftarrow E_{pk}(s_v) * \theta$
  - $E_{pk}(\min(u, v)_i) \leftarrow E_{pk}(v_i) * \lambda_i$, for $1 \leq i \leq l$

The overall steps involved in the SMIN protocol are shown in Algorithm 1. To start with, P_1 initially chooses the functionality F as either $u > v$ or $v > u$ randomly. Then, using the SM protocol, P_1 computes $E_{pk}(u_i * v_i)$ with the help of P_2, for $1 \leq i \leq l$. After this, the protocol has the following key steps, performed by P_1 locally, for $1 \leq i \leq l$:

- Compute the encrypted bit-wise XOR between the bits u_i and v_i using the following formulation[1] $T_i = E_{pk}(u_i) * E_{pk}(v_i) * E_{pk}(u_i * v_i)^{N-2}$
- Compute an encrypted vector H by preserving the first occurrence of $E_{pk}(1)$ (if there exists one) in T by initializing $H_0 = E_{pk}(0)$. The rest of the entries of H are computed as $H_i = H_{i-1}^{r_i} * T_i$. We emphasize that

1. In general, for any two given bits o_1 and o_2, the property $o_1 \oplus o_2 = o_1 + o_2 - 2(o_1 * o_2)$ always holds.

at most one of the entry in H is $E_{pk}(1)$ and the remaining entries are encryptions of either 0 or a random number.

- Then, P_1 computes $\Phi_i = E_{pk}(-1) * H_i$. Note that "$-1$" is equivalent to "$N-1$" under $\mathbb{Z}_N$. From the above discussions, it is clear that $\Phi_i = E_{pk}(0)$ at most once since H_i is equal to $E_{pk}(1)$ at most once. Also, if $\Phi_j = E_{pk}(0)$, then index j is the position at which the bits of u and v differ first (starting from the most significant bit position).

Now, depending on F, P_1 creates two encrypted vectors W and Γ as follows, for $1 \leq i \leq l$:

- If $F : u > v$, compute

$$W_i = E_{pk}(u_i * (1 - v_i)),$$
$$\Gamma_i = E_{pk}(v_i - u_i) * E_{pk}(\hat{r}_i) = E_{pk}(v_i - u_i + \hat{r}_i).$$

- If $F : v > u$, compute

$$W_i = E_{pk}(v_i * (1 - u_i)),$$
$$\Gamma_i = E_{pk}(u_i - v_i) * E_{pk}(\hat{r}_i) = E_{pk}(u_i - v_i + \hat{r}_i),$$

where $\hat{r}_i$ is a random number (hereafter denoted by $\in_R$) in $\mathbb{Z}_N$. The observation is that if $F : u > v$, then $W_i = E_{pk}(1)$ iff $u_i > v_i$, and $W_i = E_{pk}(0)$ otherwise. Similarly, when $F : v > u$, we have $W_i = E_{pk}(1)$ iff $v_i > u_i$, and $W_i = E_{pk}(0)$ otherwise. Also, depending of F, Γ_i stores the encryption of the randomized difference between u_i and v_i which will be used in later computations.

After this, P_1 computes L by combining Φ and W. More precisely, P_1 computes $L_i = W_i * \Phi_i^{r'_i}$, where r'_i is a random number in $\mathbb{Z}_N$. The observation here is if $\exists$ an index j such that $\Phi_j = E_{pk}(0)$, denoting the first flip in the bits of u and v, then W_j stores the corresponding desired information, i.e., whether $u_j > v_j$ or $v_j > u_j$ in encrypted form. In addition, depending on F, P_1 computes the encryption of randomized difference between s_u and s_v and stores it in δ. Specifically, if $F : u > v$, then $\delta = E_{pk}(s_v - s_u + \overline{r})$. Otherwise, $\delta = E_{pk}(s_u - s_v + \overline{r})$, where $\overline{r} \in_R \mathbb{Z}_N$.

After this, P_1 permutes the encrypted vectors Γ and L using two random permutation functions π_1 and π_2. Specifically, P_1 computes $\Gamma' = \pi_1(\Gamma)$ and $L' = \pi_2(L)$, and sends them along with δ to P_2. Upon receiving, P_2 decrypts L' component-wise to get $M_i = D_{sk}(L'_i)$, for $1 \leq i \leq l$, and checks for index j. That is, if $M_j = 1$, then P_2 sets α to 1, otherwise sets it to 0. In addition, P_2 computes a new encrypted vector M' depending on the value of α. Precisely, if $\alpha = 0$, then $M'_i = E_{pk}(0)$, for $1 \leq i \leq l$. Here $E_{pk}(0)$ is different for each i. On the other hand, when $\alpha = 1$, P_2 sets M'_i to the re-randomized value of Γ'_i. That is, $M'_i = \Gamma'_i * r^N$, where the term r^N comes from re-randomization and $r \in_R \mathbb{Z}_N$ should be different for each i. Furthermore, P_2 computes $\delta' = E_{pk}(0)$ if $\alpha = 0$. However, when $\alpha = 1$, P_2 sets δ' to $\delta * r_\delta^N$, where r_δ is a random number in $\mathbb{Z}_N$. Then, P_2 sends $M', E_{pk}(\alpha)$ and δ' to P_1. After receiving $M', E_{pk}(\alpha)$ and δ', P_1 computes the inverse permutation of M' as $\widetilde{M} = \pi_1^{-1}(M')$. Then, P_1 performs the following homomorphic operations to compute the encryption of ith bit of $\min(u, v)$, i.e., $E_{pk}(\min(u, v)_i)$, for $1 \leq i \leq l$:

TABLE 1
P_1 Chooses F As $v > u$ Where $u = 55$ and $v = 58$

$[u]$	$[v]$	W_i	Γ_i	G_i	H_i	Φ_i	L_i	Γ_i'	L_i'	M_i	λ_i	$\min_i$
1	1	0	r	0	0	-1	r	$1+r$	r	r	0	1
1	1	0	r	0	0	-1	r	r	r	r	0	1
0	1	1	$-1+r$	1	1	0	1	$1+r$	r	r	-1	0
1	0	0	$1+r$	1	r	r	r	$-1+r$	r	r	1	1
1	1	0	r	0	r	r	r	r	1	1	0	1
1	0	0	$1+r$	1	r	r	r	r	r	r	1	1

All column values are in encrypted form except M_i column. Also, $r \in_R \mathbb{Z}_N$ is different for each row and column.

- Remove the randomness from $\widetilde{M}_i$ by computing $\lambda_i = \widetilde{M}_i * E_{pk}(\alpha)^{N-\hat{r}_i}$
- If $F : u > v$, compute $E_{pk}(\min(u, v)_i) = E_{pk}(u_i) * \lambda_i = E_{pk}(u_i + \alpha * (v_i - u_i))$. Otherwise, compute $E_{pk}(\min(u, v)_i) = E_{pk}(v_i) * \lambda_i = E_{pk}(v_i + \alpha * (u_i - v_i))$.

Also, depending on F, P_1 computes $E_{pk}(s_{\min(u,v)})$ as follows. If $F : u > v$, P_1 computes $E_{pk}(s_{\min(u,v)}) = E_{pk}(s_u) * \theta$, where $\theta = \delta' * E_{pk}(\alpha)^{N-\overline{r}}$. Otherwise, he/she computes $E_{pk}(s_{\min(u,v)}) = E_{pk}(s_v) * \theta$.

In the SMIN protocol, one main observation (upon which we can also justify the correctness of the final output) is that if $F : u > v$, then $\min(u, v)_i = (1 - \alpha) * u_i + \alpha * v_i$ always holds, for $1 \leq i \leq l$. On the other hand, if $F : v > u$, then $\min(u, v)_i = \alpha * u_i + (1 - \alpha) * v_i$ always holds. Similar conclusions can be drawn for $s_{\min(u,v)}$. We emphasize that using similar formulations one can also design a SMAX protocol to compute $[\max(u, v)]$ and $E_{pk}(s_{\max(u,v)})$. Also, we stress that there can be multiple secrets of u and v that can be fed as input (in encrypted form) to SMIN and SMAX. For example, let s_u^1 and s_u^2 (resp., s_v^1 and s_v^2) be two secrets associated with u (resp., v). Then the SMIN protocol takes $([u], E_{pk}(s_u^1), E_{pk}(s_u^2))$ and $([v], E_{pk}(s_v^1), E_{pk}(s_v^2))$ as P_1's input and outputs $[\min(u, v)], E_{pk}(s_{\min(u,v)}^1)$ and $E_{pk}(s_{\min(u,v)}^2)$ to P_1.

Example 2. For simplicity, consider that $u = 55$, $v = 58$, and $l = 6$. Suppose s_u and s_v be the secrets associated with u and v, respectively. Assume that P_1 holds $([55], E_{pk}(s_u))$ $([58], E_{pk}(s_v))$. In addition, we assume that P_1's random permutation functions are as given below. Without loss of generality, suppose P_1 chooses the functionality $F : v > u$. Then, various intermediate results based on the SMIN protocol are as shown in Table 1. Following from Table 1, we observe that:

i	=	1	2	3	4	5	6
		$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$\pi_1(i)$	=	6	5	4	3	2	1
$\pi_2(i)$	=	2	1	5	6	3	4

- At most one of the entry in H is $E_{pk}(1)$, namely H_3, and the remaining entries are encryptions of either 0 or a random number in $\mathbb{Z}_N$.
- Index $j = 3$ is the first position at which the corresponding bits of u and v differ.

- $\Phi_3 = E_{pk}(0)$ since H_3 is equal to $E_{pk}(1)$. Also, since $M_5 = 1$, P_2 sets α to 1.
- $E_{pk}(s_{\min(u,v)}) = E_{pk}(\alpha * s_u + (1-\alpha) * s_v) = E_{pk}(s_u)$.

At the end, only P_1 knows $[\min(u,v)] = [u] = [55]$ and $E_{pk}(s_{\min(u,v)}) = E_{pk}(s_u)$.

Secure minimum out of n numbers. Consider P_1 with private input $([d_1], \ldots, [d_n])$ along with their encrypted secrets and P_2 with sk, where $0 \leq d_i < 2^l$ and $[d_i] = \langle E_{pk}(d_{i,1}), \ldots, E_{pk}(d_{i,l})\rangle$, for $1 \leq i \leq n$. Here the secret of d_i is denoted by $E_{pk}(s_{d_i})$, for $1 \leq i \leq n$. The main goal of the SMIN$_n$ protocol is to compute $[\min(d_1, \ldots, d_n)] = [d_{\min}]$ without revealing any information about d_i's to P_1 and P_2. In addition, they compute the encryption of the secret corresponding to the global minimum, denoted by $E_{pk}(s_{d_{\min}})$. Here we construct a new SMIN$_n$ protocol by utilizing SMIN as the building block. The proposed SMIN$_n$ protocol is an iterative approach and it computes the desired output in an hierarchical fashion. In each iteration, minimum between a pair of values and the secret corresponding to the minimum value are computed (in encrypted form) and fed as input to the next iteration, thus, generating a binary execution tree in a bottom-up fashion. At the end, only P_1 knows the final result $[d_{\min}]$ and $E_{pk}(s_{d_{\min}})$.

Algorithm 2. SMIN$_n(([d_1], E_{pk}(s_{d_1})), \ldots, ([d_n], E_{pk}(s_{d_n}))) \rightarrow ([d_{\min}], E_{pk}(s_{d_{\min}}))$

Require: P_1 has $(([d_1], E_{pk}(s_{d_1})), \ldots, ([d_n], E_{pk}(s_{d_n})))$; P_2 has sk

1: P_1:
   (a). $[d'_i] \leftarrow [d_i]$ and $s'_i \leftarrow E_{pk}(s_{d_i})$, for $1 \leq i \leq n$
   (b). $num \leftarrow n$
2: **for** $i = 1$ to $\lceil \log_2 n \rceil$:
   (a). **for** $1 \leq j \leq \lfloor \frac{num}{2} \rfloor$:
   - **if** $i = 1$ **then**:
     - $([d'_{2j-1}], s'_{2j-1}) \leftarrow$ SMIN(x, y), where $x = ([d'_{2j-1}], s'_{2j-1})$ and $y = ([d'_{2j}], s'_{2j})$
     - $[d'_{2j}] \leftarrow 0$ and $s'_{2j} \leftarrow 0$

     else
     - $([d'_{2i(j-1)+1}], s'_{2i(j-1)+1}) \leftarrow$ SMIN(x, y), where $x = ([d'_{2i(j-1)+1}], s'_{2i(j-1)+1})$ and $y = ([d'_{2ij-1}], s'_{2ij-1})$
     - $[d'_{2ij-1}] \leftarrow 0$ and $s'_{2ij-1} \leftarrow 0$

   (b). $num \leftarrow \lceil \frac{num}{2} \rceil$
3: P_1: $[d_{\min}] \leftarrow [d'_1]$ and $E_{pk}(s_{d_{\min}}) \leftarrow s'_1$

The overall steps involved in the proposed SMIN$_n$ protocol are highlighted in Algorithm 2. Initially, P_1 assigns $[d_i]$ and $E_{pk}(s_{d_i})$ to a temporary vector $[d'_i]$ and variable s'_i, for $1 \leq i \leq n$, respectively. Also, he/she creates a global variable num and initializes it to n, where num represents the number of (non-zero) vectors involved in each iteration. Since the SMIN$_n$ protocol executes in a binary tree hierarchy (bottom-up fashion), we have $\lceil \log_2 n \rceil$ iterations, and in each iteration, the number of vectors involved varies. In the first iteration (i.e., $i = 1$), P_1 with private input $(([d'_{2j-1}], s'_{2j-1}), ([d'_{2j}], s'_{2j}))$ and P_2 with sk involve in the SMIN protocol, for $1 \leq j \leq \lfloor \frac{num}{2} \rfloor$. At the end of the first iteration, only P_1 knows $[\min(d'_{2j-1}, d'_{2j})]$ and $s'_{\min(d'_{2j-1}, d'_{2j})}$, and nothing is revealed to P_2, for $1 \leq j \leq \lfloor \frac{num}{2} \rfloor$. Also, P_1 stores the result $[\min(d'_{2j-1}, d'_{2j})]$ and $s'_{\min(d'_{2j-1}, d'_{2j})}$ in $[d'_{2j-1}]$ and s'_{2j-1}, respectively. In addition, P_1 updates the values of $[d'_{2j}]$, s'_{2j} to 0 and num to $\lceil \frac{num}{2} \rceil$, respectively.

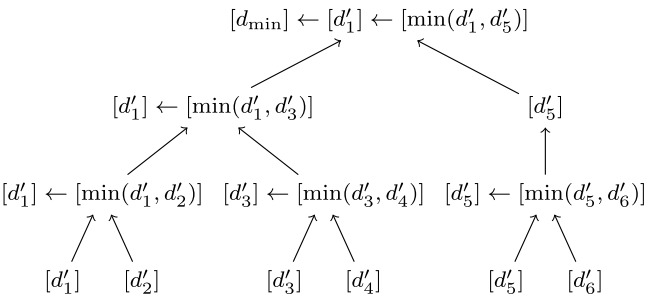


Fig. 1. Binary execution tree for $n = 6$ based on SMIN$_n$.

During the ith iteration, only the non-zero vectors (along with the corresponding encrypted secrets) are involved in SMIN, for $2 \leq i \leq \lceil \log_2 n \rceil$. For example, during the second iteration (i.e., $i = 2$), only $([d'_1], s'_1), ([d'_3], s'_3)$, and so on are involved. Note that in each iteration, the output is revealed only to P_1 and num is updated to $\lceil \frac{num}{2} \rceil$. At the end of SMIN$_n$, P_1 assigns the final encrypted binary vector of global minimum value, i.e., $[\min(d_1, \ldots, d_n)]$ which is stored in $[d'_1]$, to $[d_{\min}]$. Also, P_1 assigns s'_1 to $E_{pk}(s_{d_{\min}})$.

Example 3. Suppose P_1 holds $\langle [d_1], \ldots, [d_6]\rangle$ (i.e., $n = 6$). For simplicity, here we are assuming that there are no secrets associated with d_i's. Then, based on the SMIN$_n$ protocol, the binary execution tree (in a bottom-up fashion) to compute $[\min(d_1, \ldots, d_6)]$ is shown in Fig. 1. Note that, initially $[d'_i] = [d_i]$.

Secure frequency. Let us consider a situation where P_1 holds private input $(\langle E_{pk}(c_1), \ldots, E_{pk}(c_w)\rangle, \langle E_{pk}(c'_1), \ldots, E_{pk}(c'_k)\rangle)$ and P_2 holds the secret key sk. The goal of the SF protocol is to securely compute $E_{pk}(f(c_j))$, for $1 \leq j \leq w$. Here $f(c_j)$ denotes the number of times element c_j occurs (i.e., frequency) in the list $\langle c'_1, \ldots, c'_k\rangle$. We explicitly assume that $c'_i \in \{c_1, \ldots, c_w\}$, for $1 \leq i \leq k$.

The output $\langle E_{pk}(f(c_1)), \ldots, E_{pk}(f(c_w))\rangle$ is revealed only to P_1. During the SF protocol, neither c'_i nor c_j is revealed to P_1 and P_2. Also, $f(c_j)$ is kept private from both P_1 and P_2, for $1 \leq i \leq k$ and $1 \leq j \leq w$.

The overall steps involved in the proposed SF protocol are shown in Algorithm 3. To start with, P_1 initially computes an encrypted vector S_i such that $S_{i,j} = E_{pk}(c_j - c'_i)$, for $1 \leq j \leq w$. Then, P_1 randomizes S_i component-wise to get $S'_{i,j} = E_{pk}(r_{i,j} * (c_j - c'_i))$, where $r_{i,j}$ is a random number in $\mathbb{Z}_N$. After this, for $1 \leq i \leq k$, P_1 randomly permutes S'_i component-wise using a random permutation function π_i (known only to P_1). The output $Z_i \leftarrow \pi_i(S'_i)$ is sent to P_2. Upon receiving, P_2 decrypts Z_i component-wise, computes a vector u_i and proceeds as follows:

- If $D_{sk}(Z_{i,j}) = 0$, then $u_{i,j}$ is set to 1. Otherwise, $u_{i,j}$ is set to 0.
- The observation is, since $c'_i \in \{c_1, \ldots, c_w\}$, that exactly one of the entries in vector Z_i is an encryption of 0 and the rest are encryptions of random numbers. This further implies that exactly one of the decrypted values of Z_i is 0 and the rest are random numbers. Precisely, if $u_{i,j} = 1$, then $c'_i = c_{\pi^{-1}(j)}$.

- Compute $U_{i,j} = E_{pk}(u_{i,j})$ and send it to P_1, for $1 \leq i \leq k$ and $1 \leq j \leq w$.

Then, P_1 performs row-wise inverse permutation on it to get $V_i = \pi_i^{-1}(U_i)$, for $1 \leq i \leq k$. Finally, P_1 computes $E_{pk}(c_j) = \prod_{i=1}^{k} V_{i,j}$ locally, for $1 \leq j \leq w$.

Algorithm 3. $\text{SF}(\Lambda, \Lambda') \rightarrow \langle E_{pk}(f(c_1)), \ldots, E_{pk}(f(c_w))\rangle$

Require: P_1 has $\Lambda = \langle E_{pk}(c_1), \ldots, E_{pk}(c_w)\rangle$, $\Lambda' = \langle E_{pk}(c'_1), \ldots, E_{pk}(c'_k)\rangle$ and $\langle \pi_1, \ldots, \pi_k\rangle$; P_2 has sk

1: P_1:
  (a). **for** $i = 1$ to k **do:**
  - $T_i \leftarrow E_{pk}(c'_i)^{N-1}$
  - **for** $j = 1$ to w **do:**
    - $S_{i,j} \leftarrow E_{pk}(c_j) * T_i$
    - $S'_{i,j} \leftarrow S_{i,j}{}^{r_{i,j}}$, where $r_{i,j} \in_R \mathbb{Z}_N$
  - $Z_i \leftarrow \pi_i(S'_i)$

  (b). Send Z to P_2

2: P_2:
  (a). Receive Z from P_1
  (b). **for** $i = 1$ to k **do**
  - **for** $j = 1$ to w **do:**
    - **if** $D_{sk}(Z_{i,j}) = 0$ **then** $u_{i,j} \leftarrow 1$ **else** $u_{i,j} \leftarrow 0$
    - $U_{i,j} \leftarrow E_{pk}(u_{i,j})$

  (c). Send U to P_1

3: P_1:
  (a). Receive U from P_2
  (b). $V_i \leftarrow \pi_i^{-1}(U_i)$, for $1 \leq i \leq k$
  (c). $E_{pk}(f(c_j)) \leftarrow \prod_{i=1}^{k} V_{i,j}$, for $1 \leq j \leq w$

4 Security Analysis of Privacy-Preserving Primitives under the Semi-Honest Model

First of all, we emphasize that the outputs in the above mentioned protocols are always in encrypted format, and are known only to P_1. Also, all the intermediate results revealed to P_2 are either random or pseudo-random.

Since the proposed SMIN protocol (which is used as a sub-routine in SMIN_n) is more complex than other protocols mentioned above and due to space limitations, we are motivated to provide its security proof rather than providing proofs for each protocol. Therefore, here we only include a formal security proof for the SMIN protocol based on the standard simulation argument [28]. Nevertheless, we stress that similar proof strategies can be used to show that other protocols are secure under the semi-honest model. For completeness, we provided the security proofs for the other protocols in our technical report [5].

4.1 Proof of Security for SMIN

As mentioned in Section 2.3, to formally prove that SMIN is secure [28] under the semi-honest model, we need to show that the simulated image of SMIN is computationally indistinguishable from the actual execution image of SMIN.

An execution image generally includes the messages exchanged and the information computed from these messages. Therefore, according to Algorithm 1, let the execution image of P_2 be denoted by $\Pi_{P_2}(\text{SMIN})$, given by

$$\{\langle \delta, s + \overline{r} \bmod N\rangle, \langle \Gamma'_i, \mu_i + \hat{r}_i \bmod N\rangle, \langle L'_i, \alpha\rangle\}.$$

Observe that $s + \overline{r} \bmod N$ and $\mu_i + \hat{r}_i \bmod N$ are derived upon decrypting δ and Γ'_i, for $1 \leq i \leq l$, respectively. Note that the modulo operator is implicit in the decryption function. Also, P_2 receives L' from P_1 and let α denote the (oblivious) comparison result computed from L'. Without loss of generality, suppose the simulated image of P_2 be $\Pi^S_{P_2}(\text{SMIN})$, given by

$$\{\langle \delta^*, r^*\rangle, \langle s'_{1,i}, s'_{2,i}\rangle, \langle s'_{3,i}, \alpha'\rangle \mid \text{for } 1 \leq i \leq l\}.$$

Here $\delta^*, s'_{1,i}$ and $s'_{3,i}$ are randomly generated from $\mathbb{Z}_{N^2}$ whereas r^* and $s'_{2,i}$ are randomly generated from $\mathbb{Z}_N$. In addition, α' denotes a random bit. Since E_{pk} is a semantically secure encryption scheme with resulting ciphertext size less than N^2, δ is computationally indistinguishable from δ^*. Similarly, Γ'_i and L'_i are computationally indistinguishable from $s'_{1,i}$ and $s'_{3,i}$, respectively. Also, as $\overline{r}$ and $\hat{r}_i$ are randomly generated from $\mathbb{Z}_N$, $s + \overline{r} \bmod N$ and $\mu_i + \hat{r}_i \bmod N$ are computationally indistinguishable from r^* and $s'_{2,i}$, respectively. Furthermore, because the functionality is randomly chosen by P_1 (at step 1(a) of Algorithm 1), α is either 0 or 1 with equal probability. Thus, α is computationally indistinguishable from α'. Combining all these results together, we can conclude that $\Pi_{P_2}(\text{SMIN})$ is computationally indistinguishable from $\Pi^S_{P_2}(\text{SMIN})$ based on Definition 1. This implies that during the execution of SMIN, P_2 does not learn any information regarding u, v, s_u, s_v and the actual comparison result. Intuitively speaking, the information P_2 has during an execution of SMIN is either random or pseudo-random, so this information does not disclose anything regarding u, v, s_u and s_v. Additionally, as F is known only to P_1, the actual comparison result is oblivious to P_2.

On the other hand, the execution image of P_1, denoted by $\Pi_{P_1}(\text{SMIN})$, is given by

$$\Pi_{P_1}(\text{SMIN}) = \{M'_i, E_{pk}(\alpha), \delta' \mid \text{for } 1 \leq i \leq l\}.$$

M'_i and δ' are encrypted values, which are random in $\mathbb{Z}_{N^2}$, received from P_2 (at step 3(a) of Algorithm 1). Let the simulated image of P_1 be $\Pi^S_{P_1}(\text{SMIN})$, where

$$\Pi^S_{P_1}(\text{SMIN}) = \{s'_{4,i}, b', b'' \mid \text{for } 1 \leq i \leq l\}.$$

The values $s'_{4,i}, b'$ and b'' are randomly generated from $\mathbb{Z}_{N^2}$. Since E_{pk} is a semantically secure encryption scheme with resulting ciphertext size less than N^2, it implies that $M'_i, E_{pk}(\alpha)$ and δ' are computationally indistinguishable from $s'_{4,i}, b'$ and b'', respectively. Therefore, $\Pi_{P_1}(\text{SMIN})$ is computationally indistinguishable from $\Pi^S_{P_1}(\text{SMIN})$ based on Definition 1. As a result, P_1 cannot learn any information regarding u, v, s_u, s_v and the comparison result during the execution of SMIN. Putting everything together, we claim that the proposed SMIN protocol is secure under the semi-honest model (according to Definition 1).

5 The Proposed PPkNN Protocol

In this section, we propose a novel privacy-preserving k-NN classification protocol, denoted by PPkNN, which is

constructed using the protocols discussed in Section 3 as building blocks. As mentioned earlier, we assume that Alice's database consists of n records, denoted by $D = \langle t_1, \ldots, t_n \rangle$, and $m+1$ attributes, where $t_{i,j}$ denotes the jth attribute value of record t_i. Initially, Alice encrypts her database attribute-wise, that is, she computes $E_{pk}(t_{i,j})$, for $1 \leq i \leq n$ and $1 \leq j \leq m+1$, where column $(m+1)$ contains the class labels. Let the encrypted database be denoted by D'. We assume that Alice outsources D' as well as the future classification process to the cloud. Without loss of generality, we assume that all attribute values and their euclidean distances lie in $[0, 2^l)$. In addition, let w denote the number of unique class labels in D.

In our problem setting, we assume the existence of two non-colluding semi-honest cloud service providers, denoted by C_1 and C_2, which together form a federated cloud. Under this setting, Alice outsources her encrypted database D' to C_1 and the secret key sk to C_2. Here it is possible for the data owner Alice to replace C_2 with her private server. However, if Alice has a private server, we can argue that there is no need for data outsourcing from Alice's point of view. The main purpose of using C_2 can be motivated by the following two reasons. (i) With limited computing resource and technical expertise, it is in the best interest of Alice to completely outsource its data management and operational tasks to a cloud. For example, Alice may want to access her data and analytical results using a smart phone or any device with very limited computing capability. (ii) Suppose Bob wants to keep his input query and access patterns private from Alice. In this case, if Alice uses a private server, then she has to perform computations assumed by C_2 under which the very purpose of outsourcing the encrypted data to C_1 is negated.

In general, whether Alice uses a private server or cloud service provider C_2 actually depends on her resources. In particular to our problem setting, we prefer to use C_2 as this avoids the above mentioned disadvantages (i.e., in case of Alice using a private server) altogether. In our solution, after outsourcing encrypted data to the cloud, Alice does not participate in any future computations.

The goal of the PPkNN protocol is to classify users' query records using D' in a privacy-preserving manner. Consider an authorized user Bob who wants to classify his query record $q = \langle q_1, \ldots, q_m \rangle$ based on D' in C_1. The proposed PPkNN protocol mainly consists of the following two stages:

- Stage 1—*Secure Retrieval of k-Nearest Neighbors (SRkNN)*. In this stage, Bob initially sends his query q (in encrypted form) to C_1. After this, C_1 and C_2 involve in a set of sub-protocols to securely retrieve (in encrypted form) the class labels corresponding to the k-nearest neighbors of the input query q. At the end of this step, encrypted class labels of k-nearest neighbors are known only to C_1.
- Stage 2—*Secure Computation of Majority Class (SCMC$_k$)*. Following from Stage 1, C_1 and C_2 jointly compute the class label with a majority voting among the k-nearest neighbors of q. At the end of this step, only Bob knows the class label corresponding to his input query record q.

The main steps involved in the proposed PPkNN protocol are as shown in Algorithm 4. We now explain each of the two stages in PPkNN in detail.

Algorithm 4. PPkNN$(D', q) \rightarrow c_q$

Require: C_1 has D' and π; C_2 has sk; Bob has q

1: Bob:
   (a). Compute $E_{pk}(q_j)$, for $1 \leq j \leq m$
   (b). Send $E_{pk}(q) = \langle E_{pk}(q_1), \ldots, E_{pk}(q_m) \rangle$ to C_1
2: C_1 and C_2:
   (a). C_1 receives $E_{pk}(q)$ from Bob
   (b). **for** $i = 1$ to n do:
   - $E_{pk}(d_i) \leftarrow \text{SSED}(E_{pk}(q), E_{pk}(t_i))$
   - $[d_i] \leftarrow \text{SBD}(E_{pk}(d_i))$

3: **for** $s = 1$ to k do:
   (a). C_1 and C_2:
   - $([d_{\min}], E_{pk}(I), E_{pk}(c')) \leftarrow \text{SMIN}_n(\theta_1, \ldots, \theta_n)$, where $\theta_i = ([d_i], E_{pk}(I_{t_i}), E_{pk}(t_{i,m+1}))$
   - $E_{pk}(c'_s) \leftarrow E_{pk}(c')$

   (b). C_1:
   - $\Delta \leftarrow E_{pk}(I)^{N-1}$
   - **for** $i = 1$ to n do:
     - $\tau_i \leftarrow E_{pk}(i) * \Delta$
     - $\tau'_i \leftarrow \tau_i^{r_i}$, where $r_i \in_R \mathbb{Z}_N$
   - $\beta \leftarrow \pi(\tau')$; send β to C_2

   (c). C_2:
   - $\beta'_i \leftarrow D_{sk}(\beta_i)$, for $1 \leq i \leq n$
   - Compute U', for $1 \leq i \leq n$:
     - if $\beta'_i = 0$, then $U'_i = E_{pk}(1)$
     - otherwise, $U'_i = E_{pk}(0)$

   Send U' to C_1

   (d). C_1: $V \leftarrow \pi^{-1}(U')$
   (e). C_1 and C_2, for $1 \leq i \leq n$ and $1 \leq \gamma \leq l$:
   - $E_{pk}(d_{i,\gamma}) \leftarrow \text{SBOR}(V_i, E_{pk}(d_{i,\gamma}))$

4: $\text{SCMC}_k(E_{pk}(c'_1), \ldots, E_{pk}(c'_k))$

5.1 Stage 1: Secure Retrieval of *k*-Nearest Neighbors

During Stage 1, Bob initially encrypts his query q attribute-wise, that is, he computes $E_{pk}(q) = \langle E_{pk}(q_1), \ldots, E_{pk}(q_m) \rangle$ and sends it to C_1. The main steps involved in Stage 1 are shown as steps 1 to 3 in Algorithm 4. Upon receiving $E_{pk}(q)$, C_1 with private input $(E_{pk}(q), E_{pk}(t_i))$ and C_2 with the secret key sk jointly involve in the SSED protocol. Here $E_{pk}(t_i) = \langle E_{pk}(t_{i,1}), \ldots, E_{pk}(t_{i,m}) \rangle$, for $1 \leq i \leq n$. The output of this step, denoted by $E_{pk}(d_i)$, is the encryption of squared euclidean distance between q and t_i, i.e., $d_i = |q - t_i|^2$. As mentioned earlier, $E_{pk}(d_i)$ is known only to C_1, for $1 \leq i \leq n$. We emphasize that the computation of exact euclidean distance between encrypted vectors is hard to achieve as it involves square root. However, in our problem, it is sufficient to compare the squared euclidean distances as it preserves relative ordering. Then, C_1 with input $E_{pk}(d_i)$ and C_2 securely compute the encryptions of the individual bits of d_i using the SBD protocol. Note that the output $[d_i] = \langle E_{pk}(d_{i,1}), \ldots, E_{pk}(d_{i,l}) \rangle$ is known only to C_1, where $d_{i,1}$ and $d_{i,l}$ are the most and least significant bits of d_i, for $1 \leq i \leq n$, respectively.

After this, C_1 and C_2 compute the encryptions of class labels corresponding to the k-nearest neighbors of q in an

iterative manner. More specifically, they compute $E_{pk}(c'_1)$ in the first iteration, $E_{pk}(c'_2)$ in the second iteration, and so on. Here c'_s denotes the class label of sth nearest neighbor to q, for $1 \leq s \leq k$. At the end of k iterations, only C_1 knows $\langle E_{pk}(c'_1), \ldots, E_{pk}(c'_k)\rangle$. To start with, consider the first iteration. C_1 and C_2 jointly compute the encryptions of the individual bits of the minimum value among $d_1, \ldots, d_n$ and encryptions of the location and class label corresponding to $d_{\min}$ using the SMIN$_n$ protocol. That is, C_1 with input $(\theta_1, \ldots, \theta_n)$ and C_2 with sk compute $([d_{\min}], E_{pk}(I), E_{pk}(c'))$, where $\theta_i = ([d_i], E_{pk}(I_{t_i}), E_{pk}(t_{i,m+1}))$, for $1 \leq i \leq n$. Here $d_{\min}$ denotes the minimum value among $d_1, \ldots, d_n$; I_{t_i} and $t_{i,m+1}$ denote the unique identifier and class label corresponding to the data record t_i, respectively. Specifically, $(I_{t_i}, t_{i,m+1})$ is the secret information associated with t_i. For simplicity, this paper assumes $I_{t_i} = i$. In the output, I and c' denote the index and class label corresponding to $d_{\min}$. The output $([d_{\min}], E_{pk}(I), E_{pk}(c'))$ is known only to C_1. Now, C_1 performs the following operations locally:

- Assign $E_{pk}(c')$ to $E_{pk}(c'_1)$. Remember that, according to the SMIN$_n$ protocol, c' is equivalent to the class label of the data record that corresponds to $d_{\min}$. Thus, it is same as the class label of the most nearest neighbor to q.
- Compute the encryption of difference between I and i, where $1 \leq i \leq n$. That is, C_1 computes $\tau_i = E_{pk}(i) * E_{pk}(I)^{N-1} = E_{pk}(i - I)$, for $1 \leq i \leq n$.
- Randomize τ_i to get $\tau'_i = \tau_i^{r_i} = E_{pk}(r_i * (i - I))$, where r_i is a random number in $\mathbb{Z}_N$. Note that τ'_i is an encryption of either 0 or a random number, for $1 \leq i \leq n$. Also, it is worth noting that exactly one of the entries in τ' is an encryption of 0 (which happens iff $i = I$) and the rest are encryptions of random numbers. Permute τ' using a random permutation function π (known only to C_1) to get $\beta = \pi(\tau')$ and send it to C_2.

Upon receiving β, C_2 decrypts it component-wise to get $\beta'_i = D_{sk}(\beta_i)$, for $1 \leq i \leq n$. After this, he/she computes an encrypted vector U' of length n such that $U'_i = E_{pk}(1)$ if $\beta'_i = 0$, and $E_{pk}(0)$ otherwise. Since exactly one of entries in τ' is an encryption of 0, this further implies that exactly one of the entries in U' is an encryption of 1 and the rest of them are encryptions of 0's. It is important to note that if $\beta'_k = 0$, then $\pi^{-1}(k)$ is the index of the data record that corresponds to $d_{\min}$. Then, C_2 sends U' to C_1. After receiving U', C_1 performs inverse permutation on it to get $V = \pi^{-1}(U')$. Note that exactly one of the entries in V is $E_{pk}(1)$ and the remaining are encryptions of 0's. In addition, if $V_i = E_{pk}(1)$, then t_i is the most nearest tuple to q. However, C_1 and C_2 do not know which entry in V corresponds to $E_{pk}(1)$.

Finally, C_1 updates the distance vectors $[d_i]$ due to the following reason:

- It is important to note that the first nearest tuple to q should be obliviously excluded from further computations. However, since C_1 does not know the record corresponding to $E_{pk}(c'_1)$, we need to obliviously eliminate the possibility of choosing this record again in next iterations. For this, C_1 obliviously updates the distance corresponding to $E_{pk}(c'_1)$ to the maximum value, i.e., $2^l - 1$. More specifically, C_1 updates the distance vectors with the help of C_2 using the SBOR protocol as below, for $1 \leq i \leq n$ and $1 \leq \gamma \leq l$

$$E_{pk}(d_{i,\gamma}) = \text{SBOR}\big(V_i, E_{pk}(d_{i,\gamma})\big).$$

  Note that when $V_i = E_{pk}(1)$, the corresponding distance vector d_i is set to the maximum value. That is, under this case, $[d_i] = \langle E_{pk}(1), \ldots, E_{pk}(1)\rangle$. On the other hand, when $V_i = E_{pk}(0)$, the OR operation has no effect on the corresponding encrypted distance vector.

The above process is repeated until k iterations, and in each iteration $[d_i]$ corresponding to the current chosen label is set to the maximum value. However, C_1 and C_2 does not know which $[d_i]$ is updated. In iteration s, $E_{pk}(c'_s)$ is known only to C_1. At the end of Stage 1, C_1 has $\langle E_{pk}(c'_1), \ldots, E_{pk}(c'_k)\rangle$, the list of encrypted class labels of k-nearest neighbors to the query q.

5.2 Stage 2: Secure Computation of Majority Class

Without loss of generality, let us assume that Alice's dataset D consists of w unique class labels denoted by $c = \langle c_1, \ldots, c_w\rangle$. We assume that Alice outsources her list of encrypted classes to C_1. That is, Alice outsources $\langle E_{pk}(c_1), \ldots, E_{pk}(c_w)\rangle$ to C_1 along with her encrypted database D' during the data outsourcing step. Note that, for security reasons, Alice may add dummy categories into the list to protect the number of class labels, i.e., w from C_1 and C_2. However, for simplicity, we assume that Alice does not add any dummy categories to c.

During Stage 2, C_1 with private inputs $\Lambda = \langle E_{pk}(c_1), \ldots, E_{pk}(c_w)\rangle$ and $\Lambda' = \langle E_{pk}(c'_1), \ldots, E_{pk}(c'_k)\rangle$, and C_2 with sk securely compute $E_{pk}(c_q)$. Here c_q denotes the majority class label among $c'_1, \ldots, c'_k$. At the end of stage 2, only Bob knows the class label c_q.

Algorithm 5. SCMC$_k(E_{pk}(c'_1), \ldots, E_{pk}(c'_k)) \rightarrow c_q$

Require: $\langle E_{pk}(c_1), \ldots, E_{pk}(c_w)\rangle$, $\langle E_{pk}(c'_1), \ldots, E_{pk}(c'_k)\rangle$ are known only to C_1; sk is known only to C_2

1: C_1 and C_2:
   (a). $\langle E_{pk}(f(c_1)), \ldots, E_{pk}(f(c_w))\rangle \leftarrow \text{SF}(\Lambda, \Lambda')$, where $\Lambda = \langle E_{pk}(c_1), \ldots, E_{pk}(c_w)\rangle$, $\Lambda' = \langle E_{pk}(c'_1), \ldots, E_{pk}(c'_k)\rangle$
   (b). **for** $i = 1$ to w do:
   - $[f(c_i)] \leftarrow \text{SBD}(E_{pk}(f(c_i)))$

   (c). $([f_{\max}], E_{pk}(c_q)) \leftarrow \text{SMAX}_w(\psi_1, \ldots, \psi_w)$, where $\psi_i = ([f(c_i)], E_{pk}(c_i))$, for $1 \leq i \leq w$

2: C_1:
   (a). $\gamma_q \leftarrow E_{pk}(c_q) * E_{pk}(r_q)$, where $r_q \in_R \mathbb{Z}_N$
   (b). Send γ_q to C_2 and r_q to Bob

3: C_2:
   (a). Receive γ_q from C_1
   (b). $\gamma'_q \leftarrow D_{sk}(\gamma_q)$; send γ'_q to Bob

4: Bob:
   (a). Receive r_q from C_1 and γ'_q from C_2
   (b). $c_q \leftarrow \gamma'_q - r_q \bmod N$

The overall steps involved in Stage 2 are shown in Algorithm 5. To start with, C_1 and C_2 jointly compute the

encrypted frequencies of each class label using the k-nearest set as input. That is, they compute $E_{pk}(f(c_i))$ using (Λ, Λ') as C_1's input to the secure frequency (SF) protocol, for $1 \leq i \leq w$. The output $\langle E_{pk}(f(c_1)), \ldots, E_{pk}(f(c_w))\rangle$ is known only to C_1. Then, C_1 with $E_{pk}(f(c_i))$ and C_2 with sk involve in the secure bit-decomposition protocol to compute $[f(c_i)]$, that is, vector of encryptions of the individual bits of $f(c_i)$, for $1 \leq i \leq w$. After this, C_1 and C_2 jointly involve in the SMAX$_w$ protocol. Briefly, SMAX$_w$ utilizes the sub-routine SMAX to eventually compute $([f_{\max}], E_{pk}(c_q))$ in an iterative fashion. Here $[f_{\max}] = [\max(f(c_1), \ldots, f(c_w))]$ and c_q denotes the majority class out of Λ'. At the end, the output $([f_{\max}], E_{pk}(c_q))$ is known only to C_1. After this, C_1 computes $\gamma_q = E_{pk}(c_q + r_q)$, where r_q is a random number in $\mathbb{Z}_N$ known only to C_1. Then, C_1 sends γ_q to C_2 and r_q to Bob. Upon receiving γ_q, C_2 decrypts it to get the randomized majority class label $\gamma'_q = D_{sk}(\gamma_q)$ and sends it to Bob. Finally, upon receiving r_q from C_1 and γ'_q from C_2, Bob computes the output class label corresponding to q as $c_q = \gamma'_q - r_q \bmod N$.

5.3 Security Analysis of PPkNN under the Semi-Honest Model

First of all, we stress that due to the encryption of q and by semantic security of the Paillier cryptosystem, Bob's input query q is protected from Alice, C_1 and C_2 in our PPkNN protocol. Apart from guaranteeing query privacy, the goal of PPkNN is to protect data confidentiality and hide data access patterns.

In this paper, to prove a protocol's security under the semi-honest model, we adopted the well-known security definitions from the literature of SMC. More specifically, as mentioned in Section 2.3, we adopt the security proofs based on the standard simulation paradigm [28]. For presentation purpose, we provide formal security proofs (under the semi-honest model) for Stages 1 and 2 of PPkNN separately. Note that the outputs returned by each sub-protocol are in encrypted form and known only to C_1.

5.3.1 Proof of Security for Stage 1

As mentioned earlier, the computations involved in Stage 1 of PPkNN are given as steps 1 to 3 in Algorithm 4. For simplicity, we consider the messages exchanged between C_1 and C_2 in a single iteration (similar analysis can be deduced for other iterations).

According to Algorithm 4, the execution image of C_2 is given by $\Pi_{C_2}(\text{PP}k\text{NN}) = \{\langle \beta_i, \beta'_i\rangle \,|\, \text{for } 1 \leq i \leq n\}$ where β_i is an encrypted value which is random in $\mathbb{Z}_{N^2}$. Also, β'_i is derived upon decrypting β_i by C_2. Remember that, exactly one of the entries in β' is 0 and the rest are random numbers in $\mathbb{Z}_N$. Without loss of generality, let the simulated image of C_2 be given $\Pi^S_{C_2}(\text{PP}k\text{NN}) = \{\langle a'_{1,i}, a'_{2,i}\rangle \,|\, \text{for } 1 \leq i \leq n\}$. Here $a'_{1,i}$ is randomly generated from $\mathbb{Z}_{N^2}$ and the vector a'_2 is randomly generated in such a way that exactly one of the entries is 0 and the rest are random numbers in $\mathbb{Z}_N$. Since E_{pk} is a semantically secure encryption scheme with resulting ciphertext size less than $\mathbb{Z}_{N^2}$, we claim that β_i is computationally indistinguishable from $a'_{1,i}$. In addition, since the random permutation function π is known only to C_1, β' is a random vector of exactly one 0 and random numbers in $\mathbb{Z}_N$. Thus, β' is computationally indistinguishable from a'_2. By combining the above results, we can conclude that $\Pi_{C_2}(\text{PP}k\text{NN})$ is computationally indistinguishable from $\Pi^S_{C_2}(\text{PP}k\text{NN})$. This implies that C_2 does not learn anything during the execution of Stage 1.

On the other hand, the execution image of C_1 is given by $\Pi_{C_1}(\text{PP}k\text{NN}) = \{U'\}$ where U' is an encrypted value sent by C_2 (at step 3(c) of Algorithm 4). Let the simulated image of C_1 in Stage 1 be $\Pi^S_{C_1}(\text{PP}k\text{NN}) = \{a'\}$. Here the value of a' is randomly generated from $\mathbb{Z}_{N^2}$. Since E_{pk} is a semantically secure encryption scheme with resulting ciphertexts in $\mathbb{Z}_{N^2}$, we claim that U' is computationally indistinguishable from a'. This implies that $\Pi_{C_1}(\text{PP}k\text{NN})$ is computationally indistinguishable from $\Pi^S_{C_1}(\text{PP}k\text{NN})$. Hence, C_1 cannot learn anything during the execution of Stage 1 in PPkNN. Combining all these results together, it is clear that Stage 1 is secure under the semi-honest model.

In each iteration, it is worth pointing out that C_1 and C_2 do not know which data record belongs to current global minimum. Thus, data access patterns are protected from both C_1 and C_2. Informally speaking, at step 3(c) of Algorithm 4, a component-wise decryption of β reveals the tuple that satisfy the current global minimum distance to C_2. However, due to the random permutation by C_1, C_2 cannot trace back to the corresponding data record. Also, note that decryption operations on vector β by C_2 will result in exactly one 0 and the rest of the results are random numbers in $\mathbb{Z}_N$. Similarly, since U' is an encrypted vector, C_1 cannot know which tuple corresponds to current global minimum distance.

5.3.2 Security Proof for Stage 2

In a similar fashion, we can formally prove that Stage 2 of PPkNN is secure under the semi-honest model. Briefly, since the sub-protocols SF, SBD, and SMAX$_w$ are secure, no information is revealed to C_2. Also, the operations performed by C_1 are entirely on encrypted data and thus no information is revealed to C_1.

Furthermore, the output data of Stage 1 which are passed as input to Stage 2 are in encrypted format. Therefore, the sequential composition of the two stages lead to our PPkNN protocol and we claim it to be secure under the semi-honest model according to the Composition Theorem [28]. In particular, based on the above discussions, it is clear that the proposed PPkNN protocol protects the confidentiality of the data, the user's input query, and also hides data access patterns from Alice, C_1, and C_2. Note that Alice does not participate in any computations of PPkNN.

5.4 Security under the Malicious Model

The next step is to extend our PPkNN protocol into a secure protocol under the malicious model. Under the malicious model, an adversary (i.e., either C_1 or C_2) can arbitrarily deviate from the protocol to gain some advantage (e.g., learning additional information about inputs) over the other party. The deviations include, as an example, for C_1 (acting as a malicious adversary) to instantiate the PPkNN protocol with modified inputs (say $E_{pk}(q')$ and $E_{pk}(t'_i)$) and to abort the protocol after gaining partial information. However, in PPkNN, it is worth pointing out that neither C_1 nor C_2

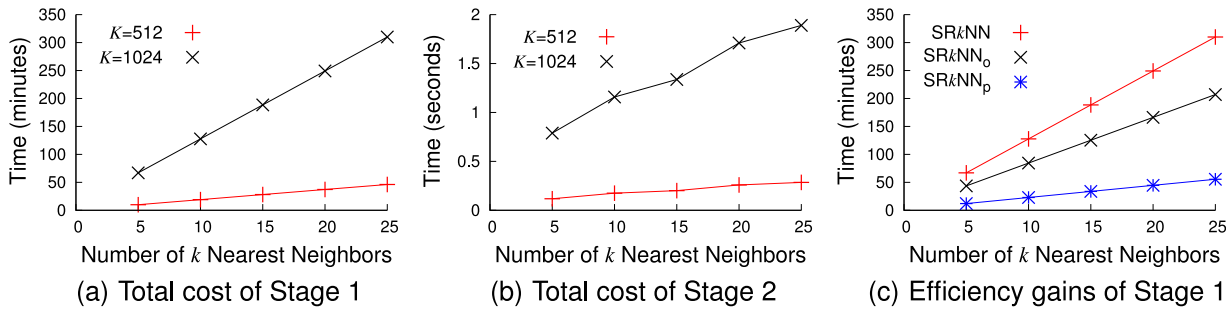


Fig. 2. Computation costs of PPkNN for varying number of k nearest neighbors and encryption key size K.

knows the results of Stages 1 and 2. In addition, all the intermediate results are either random or pseudo-random values. Thus, even when an adversary modifies the intermediate computations he/she cannot gain any additional information. Nevertheless, as mentioned above, the adversary can change the intermediate data or perform computations incorrectly before sending them to the honest party which may eventually result in the wrong output. Therefore, we need to ensure that all the computations performed and messages sent by each party are correct.

Remember that the main goal of SMC is to ensure the honest parties to get the correct result and to protect their private input data from the malicious parties. Therefore, under the two-party SMC scenario, if both parties are malicious, there is no point to develop or adopt an SMC protocol at the first place. In the literature of SMC [30], it is the norm that at most one party can be malicious under the two-party scenario. When only one of the party is malicious, the standard way of preventing the malicious party from misbehaving is to let the honest party validate the other party's work using zero-knowledge proofs [31]. However, checking the validity of operations at each step of PPkNN can significantly increase the cost.

An alternative approach, as proposed in [32], is to instantiate two independent executions of the PPkNN protocol by swapping the roles of the two parties in each execution. At the end of the individual executions, each party receives the output in encrypted form. This is followed by an equality test on their outputs. More specifically, suppose $E_{pk_1}(c_{q,1})$ and $E_{pk_2}(c_{q,2})$ be the outputs received by C_1 and C_2 respectively, where pk_1 and pk_2 are their respective public keys. Note that the outputs in our case are in encrypted format and the corresponding ciphertexts (resulted from the two executions) are under two different public key domains. Therefore, we stress that the equality test based on the additive homomorphic encryption properties which was used in [32] is not applicable to our problem. Nevertheless, C_1 and C_2 can perform the equality test based on the traditional garbled-circuit technique [33].

5.5 Complexity Analysis

The total computation complexity of Stage 1 is bounded by $O(n * (l + m + k * l * \log_2 n))$ encryptions and exponentiations. On the other hand, the total computation complexity of Stage 2 is bounded by $O(w * (l + k + l * \log_2 w))$ encryptions and exponentiations. Due to space limitations, we refer the reader to [5] for detailed complexity analysis of PPkNN. In general, as $w \ll n$, the computation cost of Stage 1 should be significantly higher than that of Stage 2. This observation is further justified by our empirical results given in the next section.

6 Empirical Results

In this section, we discuss some experiments demonstrating the performance of our PPkNN protocol under different parameter settings. We used the Paillier cryptosystem [4] as the underlying additive homomorphic encryption scheme and implemented the proposed PPkNN protocol in C. Various experiments were conducted on a Linux machine with an Intel Xeon Six-Core CPU 3.07 GHz processor and 12 GB RAM running Ubuntu 12.04 LTS. To the best of our knowledge, our work is the first effort to develop a secure k-NN classifier under the semi-honest model. There is no existing work to compare with our approach. Hence, we evaluate the performance of our PPkNN protocol under different parameter settings.

6.1 Dataset and Experimental Setup

For our experiments, we used the Car Evaluation dataset from the UCI KDD archive [34]. It consists of 1,728 records (i.e., $n = 1,728$) and six attributes (i.e., $m = 6$). Also, there is a separate class attribute and the dataset is categorized into four different classes (i.e., $w = 4$). We encrypted this dataset attribute-wise, using the Paillier encryption whose key size is varied in our experiments, and the encrypted data were stored on our machine. Based on our PPkNN protocol, we then executed a random query over this encrypted data. For the rest of this section, we do not discuss about the performance of Alice since it is a one-time cost. Instead, we evaluate and analyze the performances of the two stages in PPkNN separately.

6.2 Performance of PP*k*NN

We first evaluated the computation costs of Stage 1 in PPkNN for varying number of k-nearest neighbors. Also, the Paillier encryption key size K is either 512 or 1,024 bits. The results are shown in Fig. 2a. For $K = 512$ bits, the computation cost of Stage 1 varies from 9.98 to 46.16 minutes when k is changed from 5 to 25, respectively. On the other hand, when $K = 1{,}024$ bits, the computation cost of Stage 1 varies from 66.97 to 309.98 minutes when k is changed from 5 to 25, respectively. In either case, we observed that the cost of Stage 1 grows almost linearly with k. For any given k, we identified that the cost of Stage 1 increases by almost a factor of 7 whenever K is doubled. E.g., when $k = 10$, Stage

1 took 19.06 and 127.72 minutes to generate the encrypted class labels of the 10 nearest neighbors under $K = 512$ and 1024 bits, respectively. Moreover, when $k = 5$, we observe that around 66.29 percent of cost in Stage 1 is accounted due to SMIN_n which is initiated k times in PPkNN (once in each iteration). Also, the cost incurred due to SMIN_n increases from 66.29 to 71.66 percent when k is increased from 5 to 25.

We now evaluate the computation costs of Stage 2 for varying k and K. As shown in Fig. 2b, for $K = 512$ bits, the computation time for Stage 2 to generate the final class label corresponding to the input query varies from 0.118 to 0.285 seconds when k is changed from 5 to 25. On the other hand, for $K = 1,024$ bits, Stage 2 took 0.789 and 1.89 seconds when $k = 5$ and 25, respectively. The low computation costs of Stage 2 were due to SMAX_w which incurs significantly less computations than SMIN_n in Stage 1. This further justifies our theoretical analysis in Section 5.5. Note that, in our dataset, $w = 4$ and $n = 1{,}728$. Like in Stage 1, for any given k, the computation time of Stage 2 increases by almost a factor of 7 whenever K is doubled. E.g., when $k = 10$, the computation time of Stage 2 varies from 0.175 to 1.158 seconds when the encryption key size K is changed from 512 to 1,024 bits. As shown in Fig. 2b, a similar analysis can be observed for other values of k and K.

It is clear that the computation cost of Stage 1 is significantly higher than that of Stage 2 in PPkNN. Specifically, we observed that the computation time of Stage 1 accounts for at least 99 percent of the total time in PPkNN. For example, when $k = 10$ and $K = 512$ bits, the computation costs of Stage 1 and 2 are 19.06 minutes and 0.175 seconds, respectively. Under this scenario, cost of Stage 1 is 99.98 percent of the total cost of PPkNN. We also observed that the total computation time of PPkNN grows almost linearly with n and k.

6.3 Performance Improvement of PPkNN

We now discuss two different ways to boost the efficiency of Stage 1 (as the performance of PPkNN depends primarily on Stage 1) and empirically analyze their efficiency gains. First, we observe that some of the computations in Stage 1 can be pre-computed. For example, encryptions of random numbers, 0 and 1's can be pre-computed (by the corresponding parties) in the offline phase. As a result, the online computation cost of Stage 1 (denoted by $\text{SR}k\text{NN}_\text{o}$) is expected to be improved. To see the actual efficiency gains of such a strategy, we computed the costs of $\text{SR}k\text{NN}_\text{o}$ and compared them with the costs of Stage 1 without an offline phase (simply denoted by SRkNN) and the results for $K = 1{,}024$ bits are shown in Fig. 2c. Irrespective of the values of k, we observed that $\text{SR}k\text{NN}_\text{o}$ is around 33 percent faster than SRkNN. E.g., when $k = 10$, the computation costs of $\text{SR}k\text{NN}_\text{o}$ and SRkNN are 84.47 and 127.72 minutes, respectively (boosting the online running time of Stage 1 by 33.86 percent).

Our second approach to improve the performance of Stage 1 is by using parallelism. Since operations on data records are independent of one another, we claim that most computations in Stage 1 can be parallelized. To empirically evaluate this claim, we implemented a parallel version of Stage 1 (denoted by $\text{SR}k\text{NN}_\text{p}$) using OpenMP programming and compared its cost with the costs of SRkNN (i.e., the serial version of Stage 1). The results for $K = 1{,}024$ bits are shown in Fig. 2c. The computation cost of $\text{SR}k\text{NN}_\text{p}$ varies from 12.02 to 55.5 minutes when k is changed from 5 to 25. We observe that $\text{SR}k\text{NN}_\text{p}$ is almost six times more efficient than SRkNN. This is because our machine has six cores and thus computations can be run in parallel on six separate threads. Based on the above discussions, it is clear that efficiency of Stage 1 can indeed be improved significantly using parallelism.

On the other hand, Bob's computation cost in PPkNN is mainly due to the encryption of his input query. In our dataset, Bob's computation cost is 4 and 17 milliseconds when K is 512 and 1,024 bits, respectively. It is apparent that PPkNN is very efficient from Bob's computational perspective which is especially beneficial when he issues queries from a resource-constrained device (such as mobile phone and PDA).

6.3.1 A Note on Practicality

Our PPkNN protocol is not very efficient without utilizing parallelization. However, ours is the first work to propose a PPkNN solution that is secure under the semi-honest model. Due to rising demands for data mining as a service in cloud, we believe that our work will be very helpful to the cloud community to stimulate further research along that direction. Hopefully, more practical solutions to PPkNN will be developed (either by optimizing our protocol or investigating alternative approaches) in the near future.

7 Conclusions and Future Work

To protect user privacy, various privacy-preserving classification techniques have been proposed over the past decade. The existing techniques are not applicable to outsourced database environments where the data resides in encrypted form on a third-party server. This paper proposed a novel privacy-preserving k-NN classification protocol over encrypted data in the cloud. Our protocol protects the confidentiality of the data, user's input query, and hides the data access patterns. We also evaluated the performance of our protocol under different parameter settings.

Since improving the efficiency of SMIN_n is an important first step for improving the performance of our PPkNN protocol, we plan to investigate alternative and more efficient solutions to the SMIN_n problem in our future work. Also, we will investigate and extend our research to other classification algorithms.

Acknowledgments

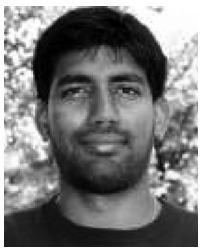
The authors wish to thank the anonymous reviewers for their invaluable feedback and suggestions. This work has been partially supported by the US National Science Foundation under grant CNS-1011984.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," *NIST Special Publication*, vol. 800, p. 145, 2011.

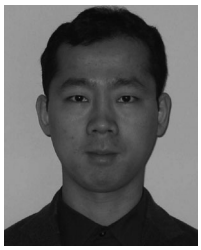
[2] S. De Capitani di Vimercati, S. Foresti, and P. Samarati, "Managing and accessing data in the cloud: Privacy risks and approaches," in *Proc. 7th Int. Conf. Risk Security Internet Syst.*, 2012, pp. 1–9.

[3] P. Williams, R. Sion, and B. Carbunar, "Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage," in *Proc. 15th ACM Conf. Comput. Commun. Security*, 2008, pp. 139–148.
[4] P. Paillier, "Public key cryptosystems based on composite degree residuosity classes," in *Proc. 17th Int. Conf. Theory Appl. Cryptographic Techn.*, 1999, pp. 223–238.
[5] B. K. Samanthula, Y. Elmehdwi, and W. Jiang, "k-nearest neighbor classification over semantically secure encrypted relational data," eprint arXiv:1403.5001, 2014.
[6] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Sympos. Theory Comput.*, 2009, pp. 169–178.
[7] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Proc. 30th Annu. Int. Conf. Theory Appl. Cryptographic Techn.: Adv. Cryptol.*, 2011, pp. 129–148.
[8] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, pp. 612–613, 1979.
[9] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proc. 13th Eur. Symp. Res. Comput. Security: Comput. Security*, 2008, pp. 192–206.
[10] R. Agrawal and R. Srikant, "Privacy-preserving data mining," *ACM Sigmod Rec.*, vol. 29, pp. 439–450, 2000.
[11] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Proc. 20th Annu. Int. Cryptol. Conf. Adv. Cryptol.*, 2000, pp. 36–54.
[12] P. Zhang, Y. Tong, S. Tang, and D. Yang, "Privacy preserving Naive Bayes classification," in *Proc. 1st Int. Conf. Adv. Data Mining Appl.*, 2005, pp. 744–752.
[13] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, "Privacy preserving mining of association rules," *Inf. Syst.*, vol. 29, no. 4, pp. 343–364, 2004.
[14] R. J. Bayardo and R. Agrawal, "Data privacy through optimal k-anonymization," in *Proc. IEEE 21st Int. Conf. Data Eng.*, 2005, pp. 217–228.
[15] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *Proc. IEEE 27th Int. Conf. Data Eng.*, 2011, pp. 601–612.
[16] M. Kantarcioglu and C. Clifton, "Privately computing a distributed k-nn classifier," in *Proc. 8th Eur. Conf. Principles Practice Knowl. Discovery Databases*, 2004, pp. 279–290.
[17] L. Xiong, S. Chitti, and L. Liu, "K nearest neighbor classification across multiple private databases," in *Proc. 15th ACM Int. Conf. Inform. Knowl. Manage.*, 2006, pp. 840–841.
[18] Y. Qi and M. J. Atallah, "Efficient privacy-preserving k-nearest neighbor search," in *Proc. IEEE 28th Int. Conf. Distrib. Comput. Syst.*, 2008, pp. 311–319.
[19] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 563–574.
[20] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service-provider model," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2002, pp. 216–227.
[21] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *VLDB J.*, vol. 21, no. 3, pp. 333–358, 2012.
[22] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 139–152.
[23] X. Xiao, F. Li, and B. Yao, "Secure nearest neighbor revisited," in *Proc. IEEE Int. Conf. Data Eng.*, 2013, pp. 733–744.
[24] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure *k*-nearest neighbor query over encrypted data in outsourced environments," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 664–675.
[25] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, 1982, pp. 160–164.
[26] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game—A completeness theorem for protocols with honest majority," in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 218–229.
[27] O. Goldreich, "General cryptographic protocols," in *The Foundations of Cryptography*, vol. 2, Cambridge, U.K.: Cambridge Univ. Press, 2004.
[28] O. Goldreich, "Encryption schemes," in *The Foundations of Cryptography*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2004, pp. 373–470 [Online]. Available: http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/enc.ps
[29] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, pp. 186–208, Feb. 1989.
[30] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proc. 20th Annu. ACM Symp. Theory Comput.*, 1988, pp. 11–19.
[31] J. Camenisch and M. Michels, "Proving in zero-knowledge that a number is the product of two safe primes," in *Proc. 17th Int. Conf. Theory Appl. Cryptographic Techn.*, 1999, pp. 107–122.
[32] Y. Huang, J. Katz, and D. Evans, "Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution," in *Proc. IEEE Symp. Security Privacy*, 2012, pp. 272–284.
[33] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proc. 20th USENIX Conf. Security*, 2011, pp. 35–35.
[34] M. Bohanec and B. Zupan. (1997). The UCI KDD Archive [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Car+Evaluation



Bharath K. Samanthula received the bachelor's degree in computer science and engineering from the International Institute of Information Technology, Hyderabad, India, in 2008, and the PhD degree in computer science from the Missouri University of Science and Technology, Rolla, MO, in 2013. He is a postdoctoral research associate at the Purdue Cyber Center and CERIAS. His research interests include applied cryptography, personal privacy and data security in the fields of social networks, cloud computing, and smart grids. He is a member of the IEEE.

Yousef Elmehdwi received the bachelor's degree in computer science from Benghazi University, Libya, in 1993 and the MSc degree in Information Technology from Mannheim University of Applied Science, Germany, in 2005. He joined the Computer Science Department at the Missouri University of Science and Technology, Rolla, Missouri, as a graduate student in January 2010. He is currently working toward the PhD degree under the supervision of Dr. Wei Jiang. His research interests lie at the crossroads of privacy, security, and data mining with current focus on privacy-preserving query processing over encrypted data outsourced to cloud.



Wei Jiang received the bachelor's degrees in both computer science and mathematics from the University of Iowa, Iowa City, Iowa, in 2002. He received the master's degree in computer science and the PhD degree from Purdue University, West Lafayette, IN, in 2004 and 2008, respectively. He is an assistant professor in the Department of Computer Science at Missouri University of Science and Technology. His research interests include privacy-preserving data mining, data integration, privacy issues in federated search environments, and text sanitization. His research has been funded by the US National Science Foundation, the Office of Naval Research, and the University of Missouri Research Board. He is a member of the IEEE.