Industrial and Applications Paper

Efficient Secure k-Nearest Neighbours over Encrypted Data

Manish Kesarwani
IBM Research India
Bangalore, India
manishkesarwani@in.ibm.com

Akshar Kaul
IBM Research India
Bangalore, India
akshar.kaul@in.ibm.com

Prasad Naldurg
IBM Research India
Bangalore, India
pnaldurg@in.ibm.com

Sikhar Patranabis
IIT Kharagpur
India
sikhar.patranabis@iitkgp.ernet.in

Gagandeep Singh
IBM Research India
Bangalore, India
gagandeep_singh@in.ibm.com

Sameep Mehta
IBM Research India
Delhi, India
sameepmehta@in.ibm.com

Debdeep Mukhopadhyay
IIT Kharagpur
India
debdeep@cse.iitkgp.ernet.in

ABSTRACT

Enterprise customers of cloud services are wary of outsourcing sensitive user and business data due to inherent security and privacy concerns. In this context, storing and computing directly on encrypted data is an attractive solution, especially against insider attacks. Homomorphic encryption, the keystone enabling technology is unfortunately prohibitively expensive. In this paper, we focus on finding k-Nearest Neighbours (k-NN) directly on encrypted data, a basic data-mining and machine learning algorithm. The goal is to compute the nearest neighbours to a given query, and present exact results to the clients, without the cloud learning anything about the data, query, results, or the access and search patterns. We describe a novel protocol in the two-party cloud setting, using an underlying somewhat homomorphic encryption scheme. In comparison to the state-of-the-art protocol in this setting, we provide asymptotically faster performance, without sacrificing any security guarantees. We implemented our protocol to demonstrate that it is efficient and practical on large and relevant real-world datasets and study how it scales well across different parameters on simulated data.

1 INTRODUCTION

Finding k-Nearest Neighbours (k-NN) is viewed as one of the simplest data mining algorithms for discovering patterns. It is non-parametric, making no assumptions about underlying data distributions. It is also a lazy learning technique, where no attempt is made to generalize the data until a query is presented. The goal of a k-NN algorithm for a given query is to find its k "nearest" neighbours according to a suitable measure of nearness or distance. It has applications in finding candidate patterns for image segmentation, location-based search, and in the classification of symptoms and diagnosis over medical records, to name a few. These patterns may be subsequently used as inputs for more sophisticated learning algorithms.

In the context of cloud computing services, client data records stored on clouds are increasingly confidential in nature, from customer transactions, order histories, credit card numbers and other personally identifiable information (PII). Algorithms, and especially algorithmic parameters, e.g., in recommendation systems, are also proprietary and sensitive. Inadvertent or unauthorised disclosure of data or computation can have serious legal or business consequences. In order to protect the confidentiality of sensitive information, clients can store encrypted data in data centres. However, it is not cost effective to import back all the data to the client, decrypt and perform computations, as this negates the advantages of moving to the cloud platform in the first place.

The Secure k-NN problem for encrypted data has been a topic of active research [13, 14, 34, 39–41]. The goal is to compute k-NN over encrypted data stored in the cloud, given a query, with the requirement that the cloud provider does not get any information about the plaintext values in the database, the query, the results, the access patterns during query evaluation, and search patterns. The technology to work effectively on encrypted data is provided by a class of encryption schemes called homomorphic encryption (HE), which allow one to compute arbitrary functions on encrypted data and produce encrypted results. Clients, who have the secret keys can decrypt and use these results safely, without revealing the data or results to the servers. Fully HE [16] schemes are expensive and impractical for now [12]. Existing solutions for Secure k-NN, therefore, work in one of two models: the centralised (private) or the two-party (public with multiple servers) cloud model. These solutions rely on a combination of partially or somewhat homomorphic schemes (PHE or (S)HE) for computation [14, 28, 39] of simpler mathematical functions, such as order preserving encryption, homomorphic addition, computation of limited degree polynomials, etc., which are faster to compute. These choices impose different tradeoffs on computation and communication overheads, and provide different security guarantees, making this a rich field for new research.

We present a new protocol in the two-party honest-but-curious cloud model for computing k-NN on encrypted data, which is *asymptotically faster* than the current state-of-the-art protocol [14], without compromising on strong security guarantees. We use LFHE (leveled fully homomorphic encryption) [9], and compute squared Euclidean distances directly on encrypted data. In order to compute the ranking among the distances, we transform the distances suitably to preserve their order and offload the comparison to a federated public cloud, who has secret keys. Since this cloud has access only to transformed results of computations performed on plaintext data, we show that in spite

of this knowledge, this honest-but-curious cloud does not learn anything useful about the original database, the results, or the query.

We implement our protocols and run experiments on two real-world datasets from the UCI machine learning repository[1]: the cervical cancer (risk factors) dataset, and the default of credit card clients dataset. Both datasets have a large number of dimensions (32 and 23 respectively). Our implementation shows that we are able to find k-NN very efficiently: 166 s on the cancer data and 373 s for the credit card dataset for our secure version for an 8-NN query, making our implementation practical for real-world applications. This trend is also echoed in the simulation results on synthetic datasets, to study the sensitivity of our protocol to various parameter choices empirically.

The rest of the paper is organized as follows: In Section 2 we present background information about our adversary and cloud models as well as a brief tutorial on homomorphic encryption. In Section 3 we present our protocol in detail, followed by a security analysis in Section 4. We describe our implementation and experiments in Section 5, relevant related work in Section 6, and conclude with future work in Section 7.

2 BACKGROUND

In this section, we present our two-party cloud architecture and describe our adversary model and trust assumptions. We also specify the Secure k-NN problem in terms of what functions need to be computed on encrypted data, and briefly introduce homomorphic encryption, with the goal of making this paper self-contained.

2.1 Cloud Architecture

The security implications of outsourcing data or computation to cloud servers need to be studied carefully. There are many deployment choices, including private clouds, public clouds and hybrid clouds. For enterprises that place a high value on data confidentiality and computational integrity, outsourcing data and computation may not be a justifiable risk. Private cloud solutions address this need, with services being accessible only within enterprise intranets, with limited benefits of cloud computing such as on-demand scaling and load balancing, as well as added initial infrastructure and set-up costs to new services. At the other end of the spectrum are the public clouds, where large third-party owned server farms and data centres host client data and computation services for hire. Two concerns stand out here: *insider attacks*, where employees within the public cloud enterprise can observe and infer trade secrets, and *side channels*, e.g., inadvertent or indirect information leaks when data belonging to different companies who may be competitors share the same bare metal. Analysis of side channels is outside the scope of this paper. A third model, the hybrid cloud model adopts a best of both worlds where depending on the sensitivity of the data and computation, enterprises may choose to offload only a part of their services on public clouds.

Our Secure k-NN solution is specifically targeted at public clouds. In particular, we work in what is called the two-party federated cloud setting, with two non-colluding public cloud servers, which is introduced in Twin Clouds [11] and subsequently used by the current state-of-the-art secure k-NN solution [14]. Federated clouds are an example of what are called interclouds [20], a collection of global stand-alone clouds. Interclouds allow better load balancing and allocation of resources and help in addressing specific scenarios, e.g., services that are region-specific and require the data to be stored in a particular geography. Coordination of services across the intercloud can be centralized, or peer-to-peer as in the case of federated clouds. A detailed survey of the taxonomy of intercloud architectures is presented in [20].

[1] http://archive.ics.uci.edu/ml/

2.2 Trust Assumptions

With federated clouds, as with public clouds, the trust model is of an *honest-but-curious* or the semi-honest adversary, who does not tamper directly with the computation or data. However, the adversary is free to observe inputs and outputs, as well as side-effects of computation and other behavioural characteristics on the cloud networks and servers. This type of adversary is different from the passive observer or the malicious adversarial model traditionally considered in the past, as the adversary is trusted to perform the computation correctly, but in addition has access to the internal state of the service, which includes client data. Such an adversary can observe the state of memory, and network traffic, or study operating systems behaviour in response to client queries. The choice of this particular type of adversary is justified, as data owners have to relinquish this control to the cloud service providers. While these exposures can be limited and controlled by legal contracts and liabilities, the threat of a curious insider can never be ruled out. This is where computing directly over encrypted data fills the gap. The goal is that even though we operate in the honest-but-curious adversary model, a malicious insider cannot obtain meaningful information from the data or the computation by observation. Further, we want to prevent the adversary from learning database access patterns, such as the set of (encrypted) result tuples returned corresponding to a particular input query, as well as the search patterns of queries, which may reveal information such as how many times the same query was issued.

Given an honest-but-curious adversary, the task of computing the k-nearest neighbours (k-NN) of a given query using a public cloud server, with the objective of achieving the above security goals in the cloud environment is difficult. One of the most attractive solutions is to use homomorphic encryption(HE). FHE (fully homomorphic encryption) [16] schemes allow the computation of arbitrary functions on encrypted data. Using FHE, data owners can offload their encrypted data to the public cloud, keeping all private or symmetric keys secret. When a client sends a query, the query is also encrypted using the same secret key and sent over to the cloud server. All computation on the cloud server is done on encrypted data and the resulting encrypted result is sent back to the clients, without compromising data confidentiality or the integrity of computation, even in the honest-but-curious adversary setting. Note that in this model, however, the actual algorithm, which is the sequence of computational steps on the data, is known to the insider, and is considered public. Care must be taken to design the algorithm to prevent leakage of search patterns and access patterns. Knowledge of the algorithm should not reveal anything about the data, query or results. Such FHE schemes however come at a significant operational cost and are not currently practical. For example, the state-of-the-art homomorphic sorting algorithm [12], takes 2 minutes to just sort 64 data items (32 bit), whereas as we show in our paper in Section 5, we are able to compute k-NN securely, with $k = 2$ for 30000 real-world data points (each point having 23 dimensions) in the same time.

To address performance issues of FHE, somewhat and partial HE ((S)HE and PHE) schemes have been proposed, which allow a restricted set of operations or sequences of operations on encrypted data, and are therefore more efficient. Examples include Paillier encryption [27] (which allow encrypted additions), LFHE [9], and BGN [8] (which allow computing restricted depth functions on encrypted data). While using simpler and more efficient schemes with restricted functionality, only a part of the computation is performed directly on the encrypted data. When more complex calculations are called for, these have to be done on plaintext values of intermediate results. To do this, another cloud provider is incorporated in the protocol. This cloud service provider is also assumed to be honest-but-curious and has access to secret keys which will allow it to decrypt the result of partial computations. Using intermediate plaintext values, more complicated operations are performed on these partial results, re-encrypted and sent back to the original cloud. The two clouds do not collude, the assumption is justified as these are public servers governed by legal contracts, and may even be business rivals. Colluding with each other will affect their reputation. This setting is called the *federated cloud model* as explained earlier.

The challenge now is to show that knowledge of such intermediate results does not leak information about the original data, query, and results as well as the access and search patterns. In this context, the state-of-the-art algorithm designed by Yousef et al. [14], shows that it is possible to design a secure k-NN scheme, where some of the secrets are given to one of the cloud servers, but both cloud servers do not learn anything about the original data. Our work also takes advantage of this model but explores a novel construction using a (S)HE scheme, and a more efficient protocol, allowing for asymptotically more efficient implementations for real-world scenarios.

2.3 Encrypted Computation

To find k-Nearest Neighbours, the distance between a given query point and all the other points in the database needs to be computed (in some appropriate dimension and measure). For computing Euclidean distance say in a two-dimensional space i.e., between two points (x_1, y_1) and (x_2, y_2) we need to compute $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. In the interest of efficiency of computation, we can avoid the square root operation and work with squared Euclidean distances. This computation requires subtraction, squaring and addition operations in that order. After we compute the squared Euclidean distances in the cloud, we need to find the k minimum values. This requires us to order encrypted values.

Order Preserving Encryption (OPE), first proposed in [7] is also emerging as an important area of new research. Using OPE, it is possible for an observer to compute the order between two ciphertexts without having to decrypt them. Any OPE solution therefore necessarily leaks the original plaintext order among the encrypted ciphertexts [2], even if it does not reveal the value of the plaintexts themselves. OPE solutions by themselves are not sufficient for our problem, since our definition of Secure k-NN does not allow the adversary to learn anything about the original points or the query, and points have to be in plaintext for the server to order them after computing the distances. In [40], it is shown that solving Secure k -NN in the single cloud model securely implies that a secure OPE solution exists in this context, one which does not leak any information, including the very order, which is leaked by definition, and this is not possible. (S)HE schemes on the other hand offer more functionality (rather than simple OPE) on encrypted data, and can be used cleverly to build a secure k-NN scheme.

2.4 Homomorphic Encryption and the LFHE scheme

We now give a brief description of the salient features of HE schemes and highlight the features of the leveled FHE (LFHE) [9] scheme we used in our implementation. HE allows direct computations to be performed on encrypted values, giving an encrypted result, which when decrypted gives the same plaintext results as if the same computations were performed on the plaintext values. An FHE scheme allows the computation of arbitrary functions on encrypted data.

A public key FHE scheme is an ensemble of four polynomial time algorithms:

- $(sk, pk) \leftarrow \mathbf{KeyGen}(\$)$, the generation of a random public key and corresponding secret key pair.
- $c \leftarrow \mathbf{Enc}_{pk}(m)$, the encryption of the a message m with the public key pk to produce a ciphertext c.
- $m \leftarrow \mathbf{Dec}_{sk}(c)$, the decryption of the ciphertext with secret key sk to produce the same plaintext m.
- $c^{'} \leftarrow \mathbf{Eval}_{pk}(\phi, c_1, \cdots, c_n)$ where ϕ is an arbitrary function in the message space, $c_1, \cdots, c_n$ are encryptions of inputs $m_1, \cdots, m_n$ to function ϕ. The result $c^{'}$ is the encryption of $m^{'} = \phi(m_1, \cdots, m_n)$, the encrypted output of application of the function on the plaintext inputs.

A partial homomorphic encryption scheme PHE is an HE scheme with a pre-defined function $\phi^{'}$, a restriction on the arbitrary function allowed in FHE. This restriction can be one function, such as addition or a sequence of functions in order, leading to a somewhat homomorphic encryption scheme ((S)HE) such as BGN, or a restriction on the kind of function (degree of the polynomial it can evaluate), for performance reasons. We assume that PHE and (S)HE satisfy the standard notion of security, whether it is against chosen plaintext attacks, which guarantee that ciphertexts output by the chosen HE scheme are indistinguishable from random, even to an adversary with access to an encryption oracle.

The (S)HE scheme we use in this paper is the Levelled FHE (LFHE) scheme implemented in HELib [17]. This LFHE scheme is based on the Brakerski-Gentry-Vaikuntanathan (BGV) scheme[10], and includes optimizations to make homomorphic evaluation runs faster, focusing mostly on the effective use of the Smart-Vercauteren [33] ciphertext packing techniques and the Gentry-Halevi-Smart [18] optimizations, with support for bootstrapping. The scheme is based on the ring learning with errors (RLWE) and the LWE problem, which have 2^{λ} security against known attacks. In addition to the **KeyGen**, **Enc**, and **Dec** functions, picking a level in the implementation L picks a depth L arithmetic circuit in **Eval**, with the computation quasilinear in the security parameter λ, lower levels corresponding to lower overheads. Further details of the LFHE scheme are presented in Appendix A.

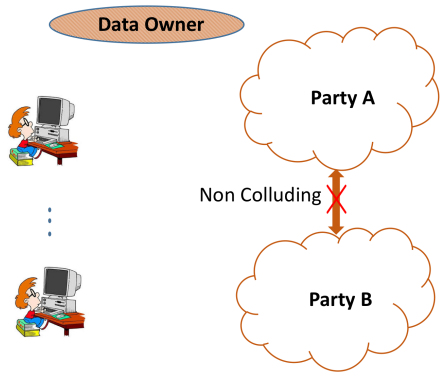
3 SECURE k-NN

In this Section, we now describe our new Secure k-NN protocol in the non-colluding two-party setting described in Section 2. Section 3.1 presents the entities involved and Section 3.2 lays down the notation. Section 3.3 gives protocol details. The security guarantees of our protocol are discussed in detail in Section 4

and the performance characteristics are explored in detail in Section 5.

3.1 Entities

Figure 1: Entities and Relationships



The main entities in our protocol are shown in Figure: 1.

- **Data owner** : The data owner is a trusted entity, the legal owner of the plaintext database, who outsources the storage of the encrypted data to Party A for k-NN computation, i.e., the plaintext database is not revealed to the party A. Once the data is outsourced, the data owner can be offline.
- **Party A** : Party A implements the storage and computation on an encrypted database. It receives encrypted query inputs and responds with k encrypted database points that represent the k-NN of the query point. It does not have access to any secret keys and works only on the encrypted data. Party A is assumed to be an honest-but-curious adversary, who will not tamper with the normal execution of the protocol, but has access to the internal state of its implementation, and could attempt to infer additional information about the characteristics of the plaintext data, query, results, or access and search patterns.
- **Party B** : Party B has access to the secret keys used to encrypt a given database, and does not have access to the encrypted database or query directly. Instead, it only has access to (partial) results of computations done on the encrypted data. Similar to Party A, Party B is honest-but-curious and does not tamper with the computation. However, it can use any information provided to infer characteristics about the plaintext database and query. A Secure k-NN solution will show that this information cannot be learned by Party B even if it has the secret key.
- **Clients** : These are users who are authorised by the data owner to interface with Party A and ask k-NN queries on the outsourced database. Clients have access to keys which allow them to send encrypted queries and decrypt the corresponding responses.

As the name suggests, in the non-colluding two party setting, Party A and Party B do not collude to expose the plaintext database to each other.

3.2 Notation

Our database P consists of n points $p_1, p_2 \ldots p_n$. Each point is d-dimensional. The query Q is also a d-dimensional point.

- $D(p_i, Q)$ represents the squared Euclidean distance between p_i and Q.
- (S)HE: is a somewhat homomorphic encryption scheme which allows computation of this distance measure in the encrypted domain itself. For the distance measure used in this paper i.e. Euclidean Distance, we use the LFHE scheme [9]. Depending on the level chosen, e.g., with level 2, we can compute other measures such as Manhattan distance etc.

3.3 Setup

Figure 2 shows the entities, the communication and the computation phases in our main protocol. The Setup phase is executed once at the time of transferring the encrypted data to Party A. Let $(sk, pk) \leftarrow \mathbf{KeyGen}(\$)$ be the secret-key and public-key respectively for the the chosen (S)HE.

- Party A receives the public key pk from the data owner, as shown in label 1 in Figure 2.
- Party A also receives the encrypted database P' from the data owner, where the magnitude of each dimension d of each point p_i is encrypted under (S)HE using $\mathbf{Enc}_{pk}$. These d encrypted values together constitute the d dimensional encrypted data points $p_i^{'}$.
- Party B receives both sk and pk, as shown in label 2 in Figure 2.
- Clients receive both sk and pk, as shown in label 3 in Figure 2.

The inputs to our Secure k-NN protocol are the n encrypted points in P' and an encrypted d-dimensional query point $Q' \leftarrow \mathbf{Enc}_{pk}(Q)$, of the plaintext query point Q computed by the client as shown in label 4 in Figure 2.

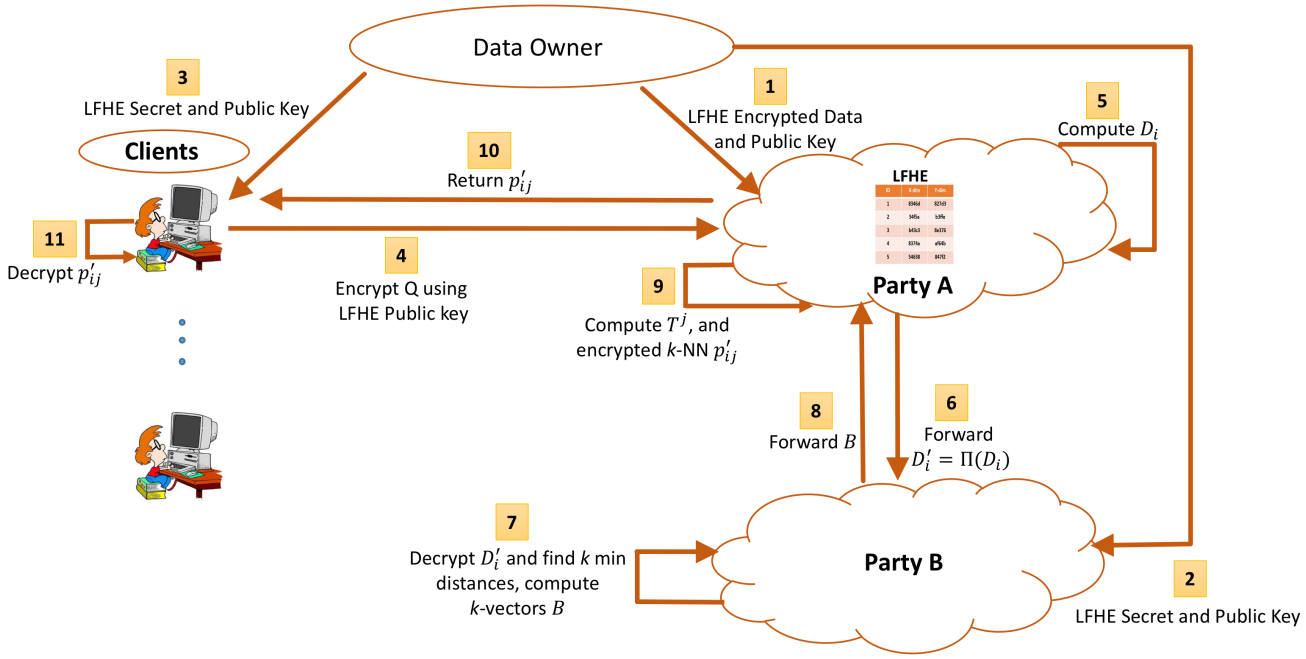
The output of the protocol, returned by Party A to the client, is the set of k encrypted points $\langle p_{i1}^{'}, p_{i2}^{'}, \cdots, p_{ik}^{'} \rangle$, which are the encrypted k nearest neighbours to query Q in database P.

3.4 Secure k-NN Protocol

We describe our Secure k-NN protocol, which involves only one round of communication between the two parties, alternating between computations performed at Party A and Party B. At Party A, the computations are performed on encrypted data, and we show how no information is revealed to a curious insider apart from the protocol parameters, which are inputs to the algorithm itself. Party B has the secret key, but only sees values that are obtained by applying some function derived from the points in the database. We present the security guarantees of our protocol in detail in Section 4.

Compute Distances: In the first phase of our protocol after Setup at Party A, as shown in label 5 in Figure 2, we are given the encrypted database P', and encrypted query Q'. For each d-dimensional point $p_i^{'} \in P'$, i.e., the for all the n points in the encrypted database, we find the squared Euclidean distance between the given point and the query using our (S)HE with $\mathbf{Eval}_{pk}$ as shown in Steps 2–4 in Algorithm 1. Each ED_i is the encrypted value of the (square of the) distance between the given query and the i-th database point. In Step 5, we pick a polynomial $m(x)$ of the form $a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_p \cdot x^p$ for some random $p \in \mathbb{N}$, where the coefficients $a_0, \cdots, a_p$ are picked uniformly at random within the range of the values of the (S)HE domain. For example, in our prototype implementation presented in this paper, we use positive random values picked from the range of points in $[1, 2^{32} - 1]$. We evaluate this monotonically increasing

Figure 2: Secure k-NN Protocol



1 Compute Distances: Party A

Data: P', Q'
Result: D'_i, the D_i in random permuted order

1 **begin**
2 **for** *Each* $p'_i \in P'$ **do**
3 $ED_i \leftarrow$ **FindEncryptedDistance**(p'_i, Q')
4 **FindEncryptedDistance** uses $\mathbf{Eval}_{pk}$ to compute $\sum_{i=1}^{d} (p'_i - Q'_i)^2$
5 Pick m a monotonically increasing polynomial with random coefficients
6 **for** $i \leftarrow 1$ **to** n **do**
7 $D_i \leftarrow$ **EvalPoly**(m, ED_i)
8 **EvalPoly** uses $\mathbf{Eval}_{pk}$ to evaluate m with ED_i
9 Pick permutation Π over i points
10 Send $D'_i \leftarrow \Pi(D_i)$ to Party B

2 Find Neighbours: Party B

Data: D'_i, the D_i in random permuted order
Result: $\mathcal{B} = B^{i1}, \cdots, B^{ik}$

1 **begin**
2 $NNindex[i] \leftarrow 0$
3 **for** $i \leftarrow 1$ **to** k **do**
4 $NN[i] \leftarrow \mathbf{Dec}_{sk}(D'_i)$
5 $NNindex[i] = i$
6 **for** $i \leftarrow k+1$ **to** n **do**
7 $max \leftarrow NN[1]$, $maxindex \leftarrow 1$;
8 **for** $j \leftarrow 2$ **to** k **do**
9 **if** $NN[j] > max$ **then**
10 $max \leftarrow NN[j]$; $maxindex \leftarrow j$
11 **if** $(d \leftarrow \mathbf{Dec}_{sk}(D'_i)) < max$ **then**
12 $NN[maxindex] = d$
13 $NNindex[maxindex] = i$
14 $\mathcal{B} \leftarrow \emptyset$
15 **for** $j \leftarrow 1$ **to** k **do**
16 **for** $i \leftarrow 1$ **to** n **do**
17 **if** $i = NNindex[j]$ **then**
18 $B^j_i \leftarrow \mathbf{Enc}_{pk}(1)$
19 **else**
20 $B^j_i \leftarrow \mathbf{Enc}_{pk}(0)$
21 $\mathcal{B} \leftarrow \mathcal{B} \cup B^j$
22 Send $\mathcal{B}$ to Party A

polynomial transformation on each encrypted database point using **EvalPoly** to obtain the D_is in Steps 6–8. Finally, we pick a permutation Π uniformly at random over the $n(n-1)/2$ points D_is as $D'_i \leftarrow \Pi(D_i)$, in Steps 9–10 and send the distances to Party B out of order, as shown in label 6 in Figure 2.

Find Neighbours: In the second phase of our protocol at Party B, as shown in label 7 in Figure 2, the transformed points D'_i are received in random permuted order. We now form two vectors (arrays) of size k: NN and $NNindex$. In Steps 2–5 in Algorithm 2, we initialize $NN[i]$ to the first k points by decrypting the D'_is (since we have the secret key sk), and store the corresponding index values $(1, \cdots, k)$ in $NNindex$ as shown. For the remaining $k-n$ points, in steps 7–10, we first find the point with the maximum distance in NN and its corresponding $maxindex$ as shown. Next, in Steps 11–13, for the new point in the ith position, we decrypt the value D'_i and check if it is smaller than the max distance we have seen so far. If it is, then we replace

the maximum value in NN with this and update the index value appropriately. At the end of this outer loop, the array NN will contain the k smallest values and the corresponding $NNindex$ will track the index of these points. It is easy to see that the points corresponding to these permuted indices will be the k nearest neighbours for the query point in the original database, however, we need to transfer this information to Party A without revealing either the permuted indices or these values directly.

To do this, we construct the set $\mathcal{B}$ of k row vectors B^j, where each B^j is a n-vector. For each permuted index value in $NNIndex$, we populate the array B^j as follows: for the position i in B^j_i at the $NNIndex[j]$ value, i.e., $i = NNindex[j]$ we store an encryption of the value 1 (Party B also has the public key pk), and in the remaining $n-1$ i-positions we store an encryption of the value 0 as shown in Steps 15–21. This new vector B^j is added to the set $\mathcal{B}$. We repeat this k times until all indices in $NNIndex$ have been processed. At the end of this phase, as shown in label 8 in Figure 2, we send the k row vectors $\mathcal{B}$ to Party A (Step 22 in Algorithm 2).

3 Return kNN: Party A

Data: $\mathcal{B} \leftarrow B^1, \cdots, B^k$
Result: $p^{'}_{i1}, p^{'}_{i2}, \cdots, p^{'}_{ik}$

1 **begin**
2 Receive set of k $n-$dimensional vectors $\mathcal{B}$
3 **for** $j \leftarrow 1$ **to** k **do**
4 $T^j \leftarrow (\Pi(P^{'}))\cdot B^j$
5 $p^{'}_{ij} \leftarrow \mathbf{Enc}(0)$
6 **for** $l \leftarrow 1$ **to** n **do**
7 $p^{'}_{ij} \leftarrow p^{'}_{ij} + T^j_l$
8 Return $p^{'}_{i1}, p^{'}_{i2}, \cdots, p^{'}_{ik}$

Return kNN: In the next phase of our protocol, Party A receives k n-dimensional row vectors (Algorithm 3 Step 2). As shown in label 9 of Figure 2, and Step 4 of the algorithm, we first apply the permutation Π selected by Party A in Algorithm 1 to the encrypted database points $P^{'}$. Applying the permutation Π on the points corrects the order of the 0s and 1s in B^j that was derived from the permuted sums in Party B. Next, we compute the scalar dot product (pointwise multiplication) between this permuted row vector with each of our B^js, giving us an n-dimensional vector T^j. After this multiplication, the original point that is one of the k-NNs, i.e., $p^{'}_{ij}$, will remain (in encrypted form) in T^j, all other entries will become (encrypted) zeros. We sum the n elements of each of the T^j row vectors as shown in Steps 5–7. The k vector sums are the encrypted k-NNs, i.e., the points $p^{'}_{i1}, p^{'}_{i2}, \cdots, p^{'}_{ik}$, which can now be returned as the final result in Step 8 of algorithm and label 10 in Figure 2. Clients can decrypt these $p^{'}_{ij}$s using sk to obtain the plaintext points that correspond to the k nearest neighbours as shown in label 9 in Figure 2.

There are two main *novel* ideas in our protocol:

- The use of m, a monotonically increasing polynomial with uniformly random coefficients, which effectively masks the database values to Party B.
- The construction of vectors B^j_i along with the uniformly random permutation Π, which prevents Party A from learning the query results, as well as the access pattern and query search pattern.

The communication and computation overheads in our protocol are as follows. Party A computes $O(n)$ encrypted values, and sends one set of n values to Party B. Party B computes n decryptions and $O(nk)$ encryptions and returns $O(nk)$ encrypted values to Party A. In the last phase Party A performs $O(nkd)$ operations on the ciphertexts.

3.5 Comparison of Performance and Efficiency with Yousef et al.

We present an algorithmic comparison of the efficiency of our protocol with that of Yousef et al. [14], the current state-of-the-art scheme for Secure k-NN in the two-party model. Both schemes involve a pair of non-colluding parties, one of which is in possession of the secret key of the somewhat homomorphic encryption scheme underlying the respective protocol. The first major advantage of our protocol is the reduced number of round communications - one, as compared to k for Yousef et al., where k is the number of nearest neighbours requested by the client. Thus, the cost of round communication in their protocol depends on the query, while in our protocol it is constant. Additionally, the construction of Yousef et al. performs bit-level decomposition operations on the encrypted data, which are costly and require specific capabilities which only certain (S)HE schemes can afford (e.g. Yousef et al. use the Paillier cryptosystem-based (S)HE). On the other hand, our protocol only exploits the basic capabilities that any (S)HE scheme can afford. In other words, our protocol uses the (S)HE scheme as a black-box, which can be easily instantiated using known (S)HE schemes such as LFHE (leveled fully homomorphic encryption) with optimized implementations. The comparison in terms of computational overheads is presented in Table 1 for computing k-NN over n data points, each of dimension d such that each dimension takes l bit values and m is a degree D polynomial in our solution.

4 SECURITY GUARANTEES

In this section, we discuss the security of our protocol in detail. We begin with the assumptions. In our secure k-NN protocol, Party A and Party B are assumed to be honest-but-curious, as described earlier. Party A will follow the protocol steps correctly, but we cannot rule out insider attacks. As discussed earlier, we do not address side channels in this paper. We also assume Party A and Party B do not collude. Party B additionally is trusted with the secret key of the (S)HE scheme. Party B again is honest-but-curious, and anything that Party B has or can compute is also assumed to be exposed to an insider. Informally, our protocol is secure in the following sense: even if this key is compromised and the data values exposed to an insider in Party B, the original database, query, and results are still secret to both Party A and B. For emphasis, we recall that Party B and Party A do not collude, and Party A does not have the decryption keys.

We present our security arguments focusing on the views of the two untrusted parties – Party A and Party B, under the assumption that they are mutually non-colliding. All other parties (clients and database owners) are trusted.

4.1 Leakage Profile for Party A

We explicitly enumerate the leakage to Party A at different phases of the overall protocol. Observe that Party A is involved in **Compute Distances** and **Return kNN**, we focus on the leakage to Party A during each of these:

Table 1: Computational overheads

	Yousef et al	Our Secure k-NN protocol
Number of Homomorphic operations	$O(n(2kl+d))$	$O(n(k+d+D))$
Number of encryptions	$O(nkl)$	$O(nk)$
Number of decryptions (Party B)	$O(n(kl+d))$	$O(n)$
Number of round communications	$O(k)$	1
Communication overhead per round	$O(nl+d)$ bits	$O(nl)$ bits

- **Compute Distances:** In this phase, Party A computes the set of encrypted distances ED_i between each encrypted data point $p_i^{'}$ and the encrypted query point Q' (Steps 3–4). Each of these computations is performed using the **FindEncryptedDistance** function, which in turn uses the homomorphic evaluation function **Eval** of the (S)HE scheme. Hence, the CPA security guarantees of the underlying (S)HE ensures that none of these computations reveal any information about the underlying data points or the plaintext distances to Party A. Subsequently, Party A homomorphically evaluates a randomly chosen polynomial m on each ED_i using the **EvalPoly** function to obtain the corresponding encrypted output D_i. Again, since all polynomial evaluations are on encrypted data, any non-negligible leakage to Party A from these computations amounts to a violation of the CPA security of the (S)HE scheme.
- **Return kNN:** This phase is an oblivious transfer of the knowledge of the k-nearest data points to Party A. Specifically, we argue that Party A does not learn which k points in the encrypted database correspond to the output set of points $p'_{i1}, \cdots, p'_{ik}$. The first observation that we use is that the set of k n-dimensional vectors $B^1, \cdots, B^k$ received by Party A contain encryptions of 0 and 1. As already mentioned, the CPA security guarantees of the underlying (S)HE ensures that Party A cannot distinguish between these 0 and 1 entries. Hence the vectors themselves do not reveal to Party A their correspondence to the respective points in the database. The inner product computations in Steps 4–7 are again performed homomorphically and leak no information to Party A. Finally, observe that multiplying an encrypted point by an encryption of 1 essentially results in a randomized re-encryption of the same point. Since, Party A has no knowledge of which entry in the vectors $B^1, \cdots, B^k$ corresponds to 1, the output points $p'_{i1}, \cdots, p'_{ik}$ cannot be traced back to the originally encrypted points in the database. Once again, any violation of the above guarantee amounts to a violation of the CPA security of the (S)HE scheme.

The aforementioned leakage profile for Party A leads to the following security guarantee with respect to Party A:

Theorem 4.1. **Secure k-NN Guarantee: Party A:** *Our secure k-nearest neighbour protocol leaks no information to Party A except the number of nearest neighbours k returned by the protocol. In particular, Party A gains no knowledge of the* access pattern, *that is, the set of points in the database corresponding to the k-nearest neighbour points returned by the protocol, and does not learn the query pattern, which reveals if two queries were the same.*

4.2 Leakage Profile for Party B

In this section, we explicitly enumerate the leakage to Party B in the **Find Neighbours** phase of the protocol. Party B receives the set $D_i^{'}$ of encrypted polynomial evaluation outputs $m(ED_i)$ in random permuted order. Note that since we have picked a secure pseudorandom permutation, which is computationally difficult to invert, implying that the exact identity of points associated with any given difference value is hidden from Party B. Since the polynomial m is order preserving, Party B can sort the decrypted polynomial outputs. We now examine the possibility of any leakage to Party B from the resulting system of ordered equations. Let $d_1^{''} < d_2^{''} < \cdots < d_n^{''}$ be the ordered set of plaintext distances, and $m(d_1^{''}) < m(d_2^{''}) < \cdots < m(d_n^{''})$ be the ordered set of polynomial outputs obtained by Party B upon decryption. As mentioned earlier, the polynomial $m(x)$ is of the form $a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_p \cdot x^p$ for some random $p \in \mathbb{N}$. Party B can formulate the following system of equations:

$$\begin{aligned} m(d_1^{''}) &= a_0 + a_1 \cdot d_1^{''} + a_2 \cdot (d_1^{''})^2 + \cdots + a_n \cdot (d_1^{''})^p \\ m(d_2^{''}) &= a_0 + a_1 \cdot d_2^{''} + a_2 \cdot (d_2^{''})^2 + \cdots + a_n \cdot (d_2^{''})^p \\ &\vdots \\ m(d_n^{''}) &= a_0 + a_1 \cdot d_n^{''} + a_2 \cdot (d_n^{''})^2 + \cdots + a_n \cdot (d_n^{''})^p \end{aligned}$$

where only the left-hand side of each equation is known to Party B. Without loss of generality, we may assume that Party B can guess with high probability the degree p of the polynomial chosen by Party A, as well as the range of values (say $[0, 2^N]$) that each plaintext distance $d_i^{''}$ can take. This is a particularly relevant assumption in the context of real-world datasets, where the adversary may possess some apriori knowledge of the range of Euclidean distances between the data points. In addition, since homomorphic polynomial evaluation in the encrypted domain is a costly operation, the degree p can only take a small range of values, which Party B can also accurately guess in a small number of trials. However, we prove that even if Party B has full knowledge of the aforementioned parameters, it cannot recover the original data points within a feasible amount of computation time. Observe that the system of equations has exactly $n+p+1$ unknown variables from Party B's point of view, while the number of equations is only n. Hence, Party B must correctly guess the $p+1$ smallest distances $d_1^{''}, d_2^{''}, \cdots, d_{p+1}^{''}$ to recover the polynomial coefficients. The average number of possible values that these distances can take is $\binom{2^N}{p+1}$, which is approximately the same as $2^{N\cdot(p+1)}$ for $2^N >> (p+1)$. In other words, the probability that Party B successfully recovers the polynomial coefficients, and subsequently the plaintext distances, is approximately $1/2^{N\cdot(p+1)}$, which is close to negligible. For example, for $N = 16$ and $p = 9$, the probability that Party B is able to recover the plaintext distances is approximately 2^{-160}, which is close to negligible for a security level of 160 bits. Thus even when the range of plaintext distances and the degree of the polynomial chosen by Party A are reasonably small and known apriori to Party B, the information leakage is negligible. Also note that Party A refreshes the polynomial for each query point, implying that Party B gains no

additional information across the queries. Finally, even if Party B is able to recover the plaintext distances in some extreme cases (e.g., when the plaintext distance values follow some specific pattern), it still does not directly reveal the plaintext data points to Party B, as the query point is also unknown.

The only possible leakage to Party B in this round is the presence of such points in the database that are equidistant from the query point Q. This is leaked from the presence of identical values in the set of values $m(d_1''), m(d_2''), \cdots, m(d_n'')$. However, since the order of the values is randomly permuted by Party A, Party B cannot map these values back to the original index of the data points in the database.

The aforementioned leakage profile for Party B leads to the following security guarantee with respect to Party B:

THEOREM 4.2. **Secure k-NN Guarantee: Party B:** *Our secure k-nearest neighbour protocol leaks no information to Party B except the number of nearest neighbours k to be returned by the protocol and the number of equidistant points in the database with respect to a given query point Q.*

4.3 Comparison of Security Guarantees with Yousef et al.

We conclude the discussion on the security of our k-NN protocol with a comparison of our security guarantees with those afforded by the current state-of-the-art scheme proposed by Yousef et al. [14]. Both schemes involve a pair of non-colluding parties, one of which is in possession of the secret key of the somewhat homomorphic encryption scheme underlying the respective protocol. In both schemes, one of the parties (Party A in our case), encounters only encrypted data and consequently, learns nothing about the data access pattern or search pattern from the protocol. In Yousef et al.'s protocol, one of the parties (equivalent to Party B) learns the distance between the query point and the global nearest neighbour at each stage of the protocol. Moreover, the presence of an irreversible permutation implies that this leakage cannot be mapped back to an actual point in the database. This distance value is not leaked by our protocol.

In our protocol, Party B learns the presence of pairwise equidistant points in the database with respect to a given query point. That is, whether there are two or more points that are equidistant is revealed to Party B, not the value of the distances unlike in Yousef et al. Moreover, the pairwise distances in our protocol are randomly permuted, which ensures that these leakages cannot be mapped back to identify the corresponding point pairs in the actual database. In both protocols, in spite of the knowledge of some distance values, or the knowledge of the number of equidistant points, the original database, query, results or access and search patterns cannot be deduced.

In the next section, we describe our reference implementation and show how it is asymptotically faster than the state-of-the-art algorithm and present its performance characteristics on simulated and real-world data sets.

5 IMPLEMENTATION AND EXPERIMENTS

In this section, we present our implementation details and record the experiments carried out to test the performance of the proposed protocol. Our experimental setup consists of four machines, representing the data owner, Party A, Party B and client. The configuration of machines representing Party A and Party B are: 4 core 2.8 GHz processors, 16 GB RAM running Ubuntu 16.04 LTS. The configuration of machines representing Data Owner and Client are: 4 core 2.8 GHz processors, 8 GB RAM running Ubuntu 16.04 LTS.

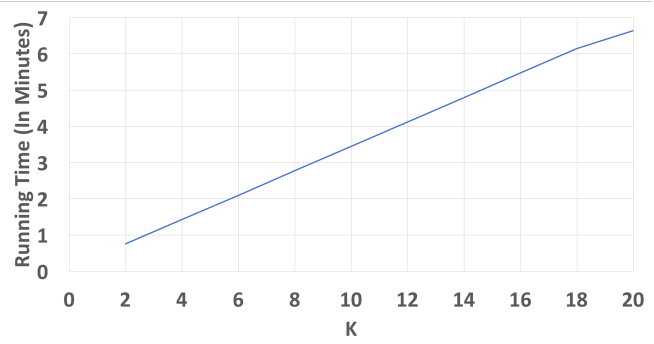


Figure 3: Running time for real world cancer data set, 858 points with 32 dimensions

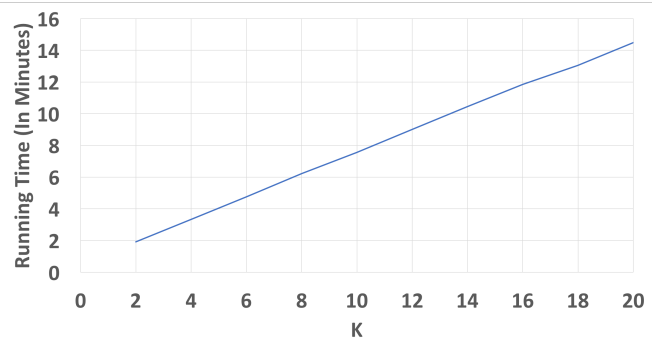


Figure 4: Running time for real world credit card data set, 30000 points with 23 dimensions

We used the HELib [17] library, with LFHE as the underlying encryption mechanism. This library is written in C++ and we implemented our protocols also in C++. In our implementation, we set $p = 1099511627689$, a large prime between 2^{40} and 2^{87}, the maximum depth to 10 and the security parameter to 128, i.e., offering 2^{128} bits of security.

5.1 Real world data

Our first set of results is on real-world data from the UCI Machine learning repository [24]. We focus on two datasets: Cervical cancer (Risk Factors) and Default of credit card clients. Our goal is to test how our Secure k-NN algorithm performs when we attempt to find reports that cluster near a chosen query report. The cervical cancer dataset contains 858 data points each having 32 dimensions, representing demographic information, habits, and historic medical records of 858 patients. The default of credit card clients dataset contains 30000 data points each having 23 dimensions including sensitive information such as the amount of given credit, gender, age, education, marital status etc. We preprocessed these datasets so that they contain only non-negative integer values. Both these datasets have PII information and are good candidates for encrypted analytics. Again we emphasize that our goal in this study is not to comment on the predictive accuracy of k-NN by itself as a suitable data mining algorithm for these datasets.

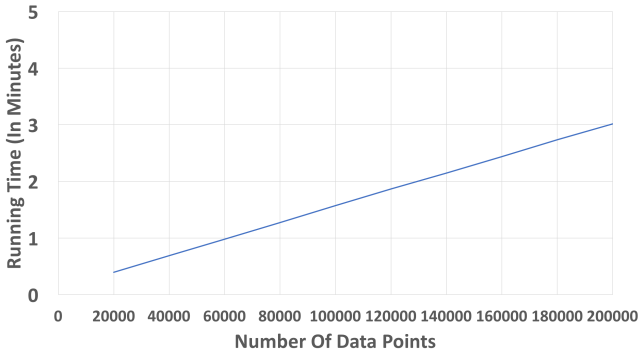


Figure 5: Running time when number of dimensions = 2 and k = 5

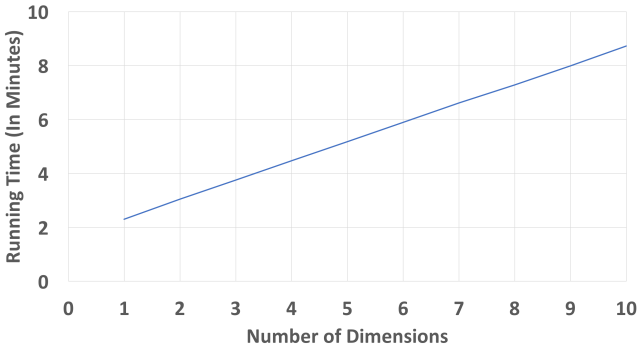


Figure 6: Running time when number of data points = 200000 and k = 2

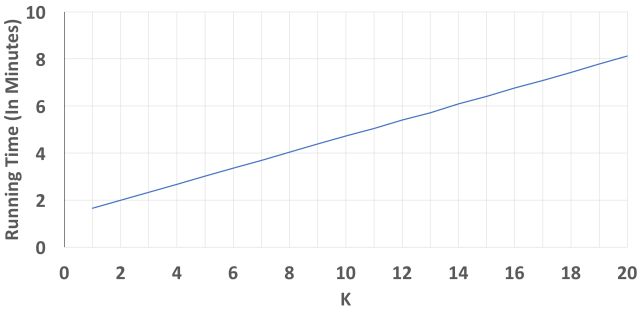


Figure 7: Running time when number of data points = 200000 and dimensions = 2

Figure 3 shows the measured running time of our protocol when we vary k from 2 to 20 on the cancer data set. As shown in the figure, we can compute 2-NN for all the points in less than a minute (45 s), 8-NN in 2 minutes and 45 seconds and 16-NN in 5 minutes and 28 seconds. The function grows linearly with k.

Figure 4 shows the measured running time of our protocol when we vary k from 2 to 20 for the credit card clients data. As shown in the figure, we can compute 2-NN in less than 2 minutes and 20-NN in 14 minutes and 20 seconds, with the function growing linearly in k. In each of these experiments, we generate a random data point to serve as the query point. The experiment is then repeated with multiple such query points and the average time taken to execute a query is recorded. From the figures, it is clear that our protocol scales linearly with k.

Both these experiments show that our implementation overheads are very competitive, making a case for real-world application of our algorithms to perform simple pattern matching on sensitive data. Other applications where this technique can be directly applied include spatial databases and location-based search (for example a taxi-for-hire application), where the query looks for points within a small set of records that are already filtered.

Note that in our protocol, the two parties A and B communicate with each other directly, without going back to the client. The communication cost between the parties, i.e., sending the monotonically increasing function of squared Euclidean distances, is independent of the number of dimensions in the original dataset, i.e., only one value is sent for each pair of points (regardless of the value of d). Response from Party B is also independent of the dimension of the data. The computational overhead of calculating the squared Euclidean distances on Party A depends naturally on the dimensions, which is unavoidable.

5.2 Simulation

There are three varying parameters in our protocol, a) number of data points n b) the number of dimensions d and c) k the number of nearest neighbours. By generating synthetic data we are able to keep two parameters fixed and check the effect of the third parameter on running time of the query. Our simulation-data generator uses a uniform random distribution to generate the data.

Figure 5 shows how the running time varies when the number of data points are increased, keeping the number of dimensions d and k constant (5 in this case). From this figure, we can clearly see that the running time of our protocol ranges from 23 seconds up to 3 minutes, and grows linearly with the size of the dataset n, which ranges from 20000 to 200000 points.

Figures 6 shows how the running time varies when the number of dimensions in the data is increased, from 1 to 10, keeping the number of data points n as 200000 and $k = 2$ as constant. The running time ranges from 2 minutes and 17 seconds to less than 9 minutes for 10 dimensions. The figure clearly shows that our protocol scales linearly with the number of dimensions.

Figures 7 shows how the running time varies when k is increased from 1 to 20 keeping the number of data points as 200000 and number of dimensions $d = 2$. The running time ranges from less than 2 minutes to around 8 minutes as shown. The figure clearly shows that the running time scales linearly with k.

From the above experiments, it is clear that the running time of our protocol is linear in k, n and d. Note that our protocol requires only **one** round of communication between Party A and Party B. In comparison, Yousef et al., requires at least $O(k)$ rounds of communication to compute the secure k-NN, with each round sending $O(nb)$ points where n is the database size and b the bit size of every element. Additionally, more rounds are required to compute the squared Euclidean distances and secure bit decomposition, which are used subsequently to compute k-NN.

For a similar machine configuration[2], for 2000 points and 6 dimensions, with $k = 25$ our protocol runs in 1 minute and 37 seconds, whereas Yousef et al., report a running time of 55 minutes and 39 seconds. We emphasize that while this comparison is

[2] 6 cores, 3.07 GHz processor and 12GB RAM running Ubuntu 10.04 LTS

done on different machines, the trend observed is explained by our efficient one-round communication and the simplified computations on Parties A and B, without compromising security guarantees.

6 RELATED WORK

Finding k- Nearest Neighbours is a fundamental operation in many data mining and machine learning algorithms. In this section, we present an overview of the techniques developed for secure k-nearest neighbour problem (SkNN). SkNN is a specific use case of the much broader problem of secure processing over outsourced encrypted data. This area has been well studied and many techniques have been proposed. Some of the techniques are generic, i.e. they are able to support a wide variety of queries over the encrypted data, while others enable a particular operation, such as k-NN, over the encrypted data. The existing work can be categorized into various buckets depending on the underlying mechanism used, including privacy preserving data mining, garbled circuit implementations of secure k-NN, secure multiparty computation (SMC), private information retrieval (PIR) schemes, secure hardware-based techniques and other solutions that directly implement secure k-NN protocols.

The first three solutions, privacy-preserving data mining, garbled circuits, and SMC, cannot be compared directly with our work, as the goals, assumptions and models are different as discussed next. In privacy-preserving data mining [3] [15] [23] [25] [35] privacy is achieved by transforming the data using anonymization models such as k-anonymity, data perturbation i.e adding noise to the data, suppressing some tuples, etc. The transformed data preserves the ability to answer the required query within an accepted error limit. The drawback of these techniques is that they lead to information loss. Typically a superset of the results are returned and the client has to process the query results further. These techniques outsource the transformed data to the server in plaintext, which leaks information as well.

Songhori et al[34] propose to use Yao's garbled circuit protocol for secure k-NN computation. In their setting, Alice has a query point Q and Bob has the dataset S. They want to jointly compute the k-NN of Q in S such that Bob does not learn anything about Q and Alice does not learn anything about S except k-NN. This solution is not geared to outsourcing as Bob has the data in plaintext.

Secure multiparty computation (SMC) techniques enable multiple parties to securely evaluate a function of their private inputs without revealing the inputs of one party to the others. SMC has been leveraged to compute SkNN by various solutions [32] [31] [37]. They partition the data and give to multiple parties, which then compute k-NN as an SMC. Again, this is fundamentally different from our work because the parties can view the data in plaintext.

Providing secure database-as-a-service has been investigated by CryptDB[30] and Monomi [36]. They use property preserving encryption such as OPE [2][7][6] to handle a portion of the query processing directly on encrypted data stored at the cloud efficiently. The portion of query which cannot be processed by these property preserving encryption schemes is executed at the client side after decryption of data returned by cloud. Monomi [36] shows that such solutions can provide good performance if they are tuned properly for the query workload, though for an arbitrary query the performance can deteriorate significantly. Also, it needs a full-fledged database engine at the client side which is not always possible, e.g., on mobile devices.

Secure Hardware based techniques, including Cipherbase and others[1] [26] [4] [5] assume the availability of a secure co-processor at the cloud server. These co-processors are specially built such that no outside entity can read the data stored inside it (tamper-proof hardware). The computations performed inside the co-processor are also isolated from the outside world. Data owners upload the keys to the co-processor in a secure manner. The logic for processing, for example, k-NN, is also installed on the co-processor. For query processing, the encrypted data and encrypted query point are sent to the co-processor, which decrypts the data using the keys stored on it and runs the installed logic on the decrypted data returning the final answer in encrypted form to the client. Such techniques give strong security guarantees. However, the co-processors are resource constrained and are less powerful than the regular processors with limited memory available to them. Also, the deployment and maintenance of such co-processors is not straight forward, as some operations have to be performed by the clients, e.g., key refresh, requiring the client to maintain the hardware.

Using PIR (Private Information retrieval) for secure k-NN has been studied in [19] [29]. PIR allows users to retrieve an object X_i from a set $X = X_i, X_i, \ldots X_n$ stored at the server without revealing i to server. Again, PIR schemes work on plaintext data on the server and guarantee that a user′s query point will not be revealed, different from our setting.

Wong et al[39] propose a new encryption scheme called ASPE (Asymmetric Scalar Product Preserving Encryption), to compare the distance between a query point and compute the distances required for k-NN. Hu et al[21] propose a secure k-NN method based on provable secure homomorphic encryption. However, the setting used by them is different from us in that the client has the ciphertext while the server has the capability to decrypt. Both solutions are vulnerable to Chosen Plaintext Attacks [40].

Yao et al[40] establish a relationship between the SkNN problem and the order preserving encryption (OPE) problem. They show that SkNN is at least as hard as OPE in a single cloud setting. They propose a solution based on Voronoi diagrams, in which the server returns a superset of results for the k-NN. On the other hand, we return the exact result of k-NN to the client in the two party federated cloud setting.

Sunoh et al[13] provide a solution for secure k-NN which uses mOPE(mutable Order Preserving Encryption). They propose two methods, one based on Voronoi diagrams, which returns the exact k-NNs but is expensive when $k > 1$, and another method based on triangulation, which is more efficient but gives exact results for only $k = 1$. For $k > 1$ it gives false positives which have to be filtered out by the client. Their solution also requires round communication between the server and client to first reduce the potential candidates and to find the k-NN. No security proofs are provided. Also, this solution has an expensive database update procedure which requires changing of encrypted data.

In his seminal work, Gentry [16] proposed a construction for a fully homomorphic encryption scheme (FHE). FHE allows computation of any function directly on encrypted data and a solution for computing k-NN can be developed using this. As discussed earlier, the computational cost of FHE is very high for real world applications.

Yousef et al[14] describe the current state-of-the-art protocol for SkNN in the two party federated cloud model. They use Paillier encryption [27] as the underlying cryptographic tool. Paillier

encryption is additive homomorphic. Using this property they develop protocols which compute k-NN securely in the federated cloud model. Their solution provides very strong security guarantees and is able to hide the data access pattern as well. However, this security comes at the cost of performance, taking minutes to perform queries that we can execute in seconds and milliseconds. We are able to improve upon the performance characteristics as shown in Section 5, while maintaining strong security guarantees, making it more suitable for real-world deployments.

Other recent related work includes Lei et.al. [22], a SkNN scheme for 2-D data points using LSH (location sensitive hashing). They first construct the secure index for the data and then outsource the secure index and encrypted data to the cloud. Since the scheme uses LSH data structures, their results contain false positives. Also related to computing on encrypted data, though not kNN, is Seabed [28], where the authors use an additively symmetric homomorphic encryption scheme to compute large-scale aggregations of data in an enterprise setting and prevent against frequency attacks.

7 CONCLUSIONS

This paper describes a new protocol for efficient secure k- Nearest Neighbours on encrypted data. We use LFHE, a somewhat homomorphic encryption scheme as our basic building block, in the two party federated cloud model. Our adversary model captures insider attacks under the honest-but-curious assumption. Our protocol is fast compared to the state of the art, without compromising on security guarantees. Our implementations are fast and scalable, and our experiments on real-world data show how basic data mining on encrypted data can be practical. In the future, we plan to extend our work to other data mining algorithms, including k-Means and Apriori.

REFERENCES

[1] R. Agrawal, D. Asonov, M. Kantarcioglu, and Yaping Li. 2006. Sovereign Joins. In *22nd International Conference on Data Engineering (ICDE'06)*. 26–26. https://doi.org/10.1109/ICDE.2006.144

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 563–574.

[3] Rakesh Agrawal and Ramakrishnan Srikant. 2000. Privacy-preserving Data Mining. *SIGMOD Rec.* 29, 2 (May 2000), 439–450. https://doi.org/10.1145/335191.335438

[4] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramaratnam Venkatesan. 2013. Orthogonal security with cipherbase. In *Proc. of the 6th CIDR, Asilomar, CA*.

[5] S. Bajaj and R. Sion. 2014. TrustedDB: A Trusted Hardware-Based Database with Privacy and Data Confidentiality. *IEEE Transactions on Knowledge and Data Engineering* 26, 3 (March 2014), 752–765. https://doi.org/10.1109/TKDE.2013.38

[6] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. 2009. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 224–241.

[7] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. 2011. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*. Springer, 578–595.

[8] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. 2005. Evaluating 2-DNF Formulas on Ciphertexts. In *Proceedings of the Second International Conference on Theory of Cryptography (TCC'05)*. Springer-Verlag, Berlin, Heidelberg, 325–341. https://doi.org/10.1007/978-3-540-30576-7_18

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12)*. ACM, New York, NY, USA, 309–325. https://doi.org/10.1145/2090236.2090262

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. *ACM Trans. Comput. Theory* 6, 3, Article 13 (July 2014), 36 pages. https://doi.org/10.1145/2633600

[11] Sven Bugiel, Stefan Nürnberger, Ahmad-Reza Sadeghi, and Thomas Schneider. 2011. Twin Clouds: Secure Cloud Computing with Low Latency. In *Proceedings of the 12th IFIP TC 6/TC 11 International Conference on Communications and Multimedia Security (CMS'11)*. 32–44.

[12] Gizem S. Çetin, Yarkin Doröz, Berk Sunar, and Erkay Savas. 2015. Low Depth Circuits for Efficient Homomorphic Sorting. *IACR Cryptology ePrint Archive* 2015 (2015), 274.

[13] S. Choi, G. Ghinita, H. S. Lim, and E. Bertino. 2014. Secure kNN Query Processing in Untrusted Cloud Environments. *IEEE Transactions on Knowledge and Data Engineering* 26, 11 (Nov 2014), 2818–2831. https://doi.org/10.1109/TKDE.2014.2302434

[14] Y. Elmehdwi, B. K. Samanthula, and W. Jiang. 2014. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *2014 IEEE 30th International Conference on Data Engineering*. 664–675. https://doi.org/10.1109/ICDE.2014.6816690

[15] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and Johannes Gehrke. 2002. Privacy Preserving Mining of Association Rules. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)*. ACM, New York, NY, USA, 217–228. https://doi.org/10.1145/775047.775080

[16] Craig Gentry. 2009. *A Fully Homomorphic Encryption Scheme*. Ph.D. Dissertation. Stanford, CA, USA. Advisor(s) Boneh, Dan. AAI3382729.

[17] Craig Gentry and Shai Halevi. 2011. Implementing Gentry's Fully-homomorphic Encryption Scheme. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT'11)*. Springer-Verlag, Berlin, Heidelberg, 129–148. http://dl.acm.org/citation.cfm?id=2008684.2008697

[18] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Better Bootstrapping in Fully Homomorphic Encryption. In *Proceedings of the 15th International Conference on Practice and Theory in Public Key Cryptography (PKC'12)*. Springer-Verlag, Berlin, Heidelberg, 1–16. https://doi.org/10.1007/978-3-642-30057-8_1

[19] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. 2008. Private Queries in Location Based Services: Anonymizers Are Not Necessary. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*. ACM, New York, NY, USA, 121–132. https://doi.org/10.1145/1376616.1376631

[20] Nikolay Grozev and Rajkumar Buyya. 2014. Inter-Cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience* 44, 3 (2014).

[21] Haibo Hu, Jianliang Xu, Chushi Ren, and Byron Choi. 2011. Processing Private Queries over Untrusted Data Cloud Through Privacy Homomorphism. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering (ICDE '11)*. IEEE Computer Society, Washington, DC, USA, 601–612. https://doi.org/10.1109/ICDE.2011.5767862

[22] X. Lei, A. X. Liu, and R. Li. 2017. Secure KNN Queries over Encrypted Data: Dimensionality Is Not Always a Curse. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 231–234. https://doi.org/10.1109/ICDE.2017.91

[23] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. 2007. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE 2007*. IEEE, 106–115.

[24] M. Lichman. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[25] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. 2007. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 3.

[26] E. Mykletun and G. Tsudik. 2005. Incorporating a secure coprocessor in the database-as-a-service model. In *Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'05)*. 7 pp.–. https://doi.org/10.1109/IWIA.2005.28

[27] Pascal Paillier. 1999. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. Springer Berlin Heidelberg, Berlin, Heidelberg, 223–238. https://doi.org/10.1007/3-540-48910-X_16

[28] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big data analytics over encrypted datasets with seabed. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 587–602.

[29] Stavros Papadopoulos, Spiridon Bakiras, and Dimitris Papadias. 2010. Nearest Neighbor Search with Strong Location Privacy. *Proc. VLDB Endow.* 3, 1-2 (Sept. 2010), 619–629. https://doi.org/10.14778/1920841.1920920

[30] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. ACM, New York, NY, USA, 85–100. https://doi.org/10.1145/2043556.2043566

[31] Yinian Qi and Mikhail J Atallah. 2008. Efficient privacy-preserving k-nearest neighbor search. In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*. IEEE, 311–319.

[32] Mark Shaneck, Yongdae Kim, and Vipin Kumar. 2006. Privacy preserving nearest neighbor search. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*. IEEE, 541–545.

[33] N. P. Smart and F. Vercauteren. 2010. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography (PKC'10)*. Springer-Verlag, Berlin, Heidelberg, 420–443. https://doi.org/10.1007/978-3-642-13013-7_25

[34] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. 2015. Compacting Privacy-preserving K-nearest Neighbor Search Using Logic Synthesis. In *Proceedings of the 52Nd Annual Design Automation*

Conference (DAC '15). ACM, New York, NY, USA, Article 36, 6 pages. https://doi.org/10.1145/2744769.2744808
[35] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.
[36] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing Analytical Queries over Encrypted Data. *Proc. VLDB Endow.* 6, 5 (March 2013), 289–300. https://doi.org/10.14778/2535573.2488336
[37] Jaideep Vaidya and Chris Clifton. 2005. Privacy-preserving top-k queries. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. IEEE, 545–546.
[38] V. Vaikuntanathan. 2011. Computing Blindfolded: New Developments in Fully Homomorphic Encryption. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. 5–16. https://doi.org/10.1109/FOCS.2011.98
[39] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. 2009. Secure kNN Computation on Encrypted Databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD '09)*. ACM, New York, NY, USA, 139–152. https://doi.org/10.1145/1559845.1559862
[40] Xiaokui Xiao, Feifei Li, and Bin Yao. 2013. Secure Nearest Neighbor Revisited. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013) (ICDE '13)*. IEEE Computer Society, Washington, DC, USA, 733–744. https://doi.org/10.1109/ICDE.2013.6544870
[41] Youwen Zhu, Zhiqiu Huang, and Tsuyoshi Takagi. 2016. Secure and controllable -NN query over encrypted cloud data with key confidentiality. *J. Parallel and Distrib. Comput.* 89 (2016), 1 – 12. https://doi.org/10.1016/j.jpdc.2015.11.004

A THE LFHE SYSTEM

The general encryption of BGV scheme that can be instantiated to both LWE and RLWE. We will describe RLWE which used by HELib. The RLWE-based public key encryption scheme as follows. Most of the description and equations are taken from[9] [38].

In general, homomorphic encryption scheme is a tuple (HE.KeyGen, HE.Enc, HE.Dec, HE.Eval) of probabilistic polynomial time algorithms. In BGV, the message space of the scheme will always be some ring ***RM*** and our computational model will be arithmetic circuits over this ring (i.e. addition and multiplication gates).

(1) HE.*KeyGen* takes the security parameter (and possibly other parameters of the scheme) and produces a secret key *sk* and a public key *pk*.
(2) HE.*Enc* takes the public key *pk* a message *m* and produces a ciphertext *c*, which is the encryption of *m*.
(3) HE.*Dec* takes the secret key *sk* and a ciphertext *c* and produces a message *m*.
(4) HE.*Eval* takes the public key *pk*, an arithmetic circuit f over ***RM***, and ciphertexts $c_1, ..., c_l$ where l is the number of inputs to f, and outputs a ciphertext c_f.

Given the security parameter λ and an additional parameter μ, first choose a μ-bit modulus q. Where q an odd positive modulus $q = q(\lambda)$. For RLWE scheme, chose the degree $d = d(\lambda, \mu)$, a "noise" distribution $\chi = \chi(\lambda, \mu)$, let the "dimension" $n = [3 \log q]$. Let $R_q = \mathbb{Z}_q[x]/(f(x))$ with $f(x)$ a polynomial of degree d. $f(x) = x^d + 1$ and $d = d(\lambda)$ is a power of 2. To get the secret key, first draw **s'** uniformly from χ. The secret key is then

$$\mathbf{s} = (1, \mathbf{s'}) \in R_q^2$$

To get the public key, first generate vectors $\mathbf{A'} \longleftarrow R_q^n$, $e \leftarrow \chi^n$, then set $b = -\mathbf{A's'} + 2e$. Set public key $A = (b|\mathbf{A'}) \in R_q^{n \times 2}$. Note that $A.s = 2e$.

Suppose $m \in 0, 1$ Is the bit we wanting to encrypt. To encrypt, we do the following:

(1) Select a random $r \in R_2^n$ and expand the message $m = (m, 0) \in R_q^n$.
(2) Output = $m + A^T r \in R_q^n$.

According to $RLWE_{d,q,\chi}$ where χ is a uniform distribution over R_q, we can use this scheme a polynomial number of times with negligible probability that an adversary can guess **s**.

To decrypt, do the following:

(1) Compute $b' = [\langle \mathbf{c}, \mathbf{s} \rangle]_q$
(2) Output $m = [b']_2$