1-2021

A Secure and Verifiable Computation for k-Nearest Neighbor Queries in Cloud

Salma Yahya Bokhary

Abstract

The popularity of cloud computing has increased significantly in the last few years due to scalability, cost efficiency, resiliency, and quality of service. Organizations are more interested in outsourcing the database and DBMS functionalities to the cloud owing to the tremendous growth of big data and on-demand access requirements. As the data is outsourced to untrusted parties, security has become a key consideration to achieve the confidentiality and integrity of data. Therefore, data owners must transform and encrypt the data before outsourcing. In this paper, we focus on a Secure and Verifiable Computation for k-Nearest Neighbor (SVC-kNN) problem. The existing verifiable computation approaches for the kNN problem delegate the verification task solely to a single semi-trusted party. We show that these approaches are unreliable in terms of security, as the verification server could be either dishonest or compromised. To address these issues, we propose a novel solution to the SVC-kNN problem that utilizes the random-splitting approach in conjunction with the homomorphic properties under a two-cloud model. Specifically, the clouds generate and send verification proofs to end-users, allowing them to verify the computation results efficiently. Our solution is highly efficient from the data owner and query issuers' perspective as it significantly reduces the encryption cost and pre-processing time. Furthermore, we demonstrated the correctness of our solution using Proof by Induction methodology to prove the Euclidean Distance Verification.

MONTCLAIR STATE UNIVERSITY

A Secure and Verifiable Computation for k-Nearest Neighbor Queries in Cloud

by

Salma Yahya Bokhary

A Master's Thesis Submitted to the Faculty of

Montclair State University

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

January 2021

College: College of Science and Mathematics

Department: Computer Science

Thesis Committee:

[redacted]

Dr. Bharath K. Samanthula
Thesis Sponsor

[redacted]

Dr. Boxiang Dong
Committee Member

[redacted]

Dr. Jiayin Wang
Committee Member

A SECURE AND VERIFIABLE COMPUTATION FOR K-NEAREST NEIGHBOR QUERIES IN CLOUD

A THESIS

Submitted in partial fulfillment of the requirements

for the degree of Master of Science

by

SALMA YAHYA BOKHARY

Montclair State University

Montclair, NJ

2021

Acknowledgements

This journey would not have been possible without the help of Allah; then, the support of my family, professors, and friends. I am thankful to everyone who has helped me to pursue my degree. Thanks to ALLAH for giving me the strength and determination to complete this thesis.

I would like to express my sincere gratitude to Dr. Bharath K. Samanthula, who has given much time and effort, and knowledge to assist in the completion of this dissertation. Thank you for your encouragement, patience, and enthusiasm. The knowledge I have gained is not only in this research topic, but it is also in the field of Cybersecurity.

I cannot begin to express my thanks to my parents, who were always the heroes behind the scenes, helping me in each station of my life, and providing emotional and financial support whenever I need it.

I owe thanks to my lovely husband, Ahmad, for his endless love and support, and encouragement to pursue my master's degree. I appreciate your help, sacrifice, and trust. Your beliefs in me sparked my inspiration, motivated me towards this journey, and made my dreams come true! I am also grateful to my little daughter, Nour, whose journey in life has started at the beginning of this program. She has been the source of strength, love, and passion that motivated me towards all my academic achievements.

I also would like to thank my sisters for encouraging me in all of my pursuits. I am deeply indebted to my brother, Mohammad, for his invaluable help and support.

I am also thankful to my dissertation committee, Dr. Boxiang Dong and Dr. Jiayin Wang. Thank you for the time you spent reading my thesis and for the interest you have shown.

Last but not least, I would like to thank my friend, Gowri P. Sundarapandi, for her support during my journey. You were a great study partner and an ideal colleague in working with me on this research!

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

Outsourcing data and computational tasks to the cloud is gaining importance as it provides efficiency and flexibility to data owners. It also reduces the infrastructure needs of organizations by providing hardware, software, storage, and maintenance as a service [1] [2]. Cloud computing is known to offer database as a service (DBaS) that helps organizations to achieve improved productivity, better standards, and data security. However, it raises some concerns about preserving the privacy and accuracy of data. In other words, the clouds can be compromised or behave dishonestly due to financial incentives or malicious activities [3], [4].

In this project, we study the kNN problem where the data owner outsources the data and the computational tasks to the cloud. Besides, the end-user submits queries and obtains the top k records from the database. Since the clouds are untrusted parties, there are high chances that they provide inaccurate results to the end-user. Errors can occur unintentionally during the computation process or intentionally due to monetary gain or malicious activities. Specifically, the cloud might choose to return a random result instead of executing heavy computations assuming that it is difficult to detect this behavior due to the lack of a verification mechanism at the end-user's end [5]. Therefore, it is essential to develop a scheme that protects the confidentiality of data and verifies the accuracy of returned results. We focus on addressing the problem of (1) secure processing of queries over encrypted data in the cloud and (2) verifying the correctness of the kNN results.

In our approach, we consider the Euclidean Distance as the distance metric to compute the top k records for a given query Q. More specifically, given a query Q, the scheme should compute the distance between $E_{pk}(Q)$ and encryption of each tuple $E_{pk}(t_i)$ in the database

(DB) and return (1) the k-nearest tuples to Q and (2) proofs to verify the correctness of the results. To preserve confidentiality, all computations should be implemented on encrypted data, and the scheme must hide the data access patterns during the query processing and the verification phase [6], [7].

Several papers have addressed the computation of the SkNN problem [8], [9], [10]; however, they did not take the verification of results into account. The proposed methods in [9] [10] are not CPA-secure. Moreover, the schemes in [9] [11] [12] incur heavy computations at the end-user during the query processing step, and [11] returns approximate kNN results to the end-user. While the proposed SkNN protocol in [8] is considered to be secure, it requires the data owner to encrypt the entire database before outsourcing, which consumes the data owners computational resources. Further, [13] produces a method to verify the integrity of outsourced frequent itemset mining by transforming some real tuples to fake and vice-versa. The correctness and completeness are verified by checking against the fake itemsets. However, using similar verification methods for the kNN problem is inappropriate because the results for kNN computation depend on the user's query, which is different and unpredictable. Lastly, researchers in [14] proposed a verification protocol for the kNN problem. However, there are security considerations in their approach. Firstly, it involves a single party in the verification process. Secondly, the scheme uses Asymmetric Scalar Product Encryption, which is insecure against Known Plaintext Attacks. Lastly, the scheme reveals the data access patterns to the cloud as the information about indices of fake tuples is disclosed to the verifying server.

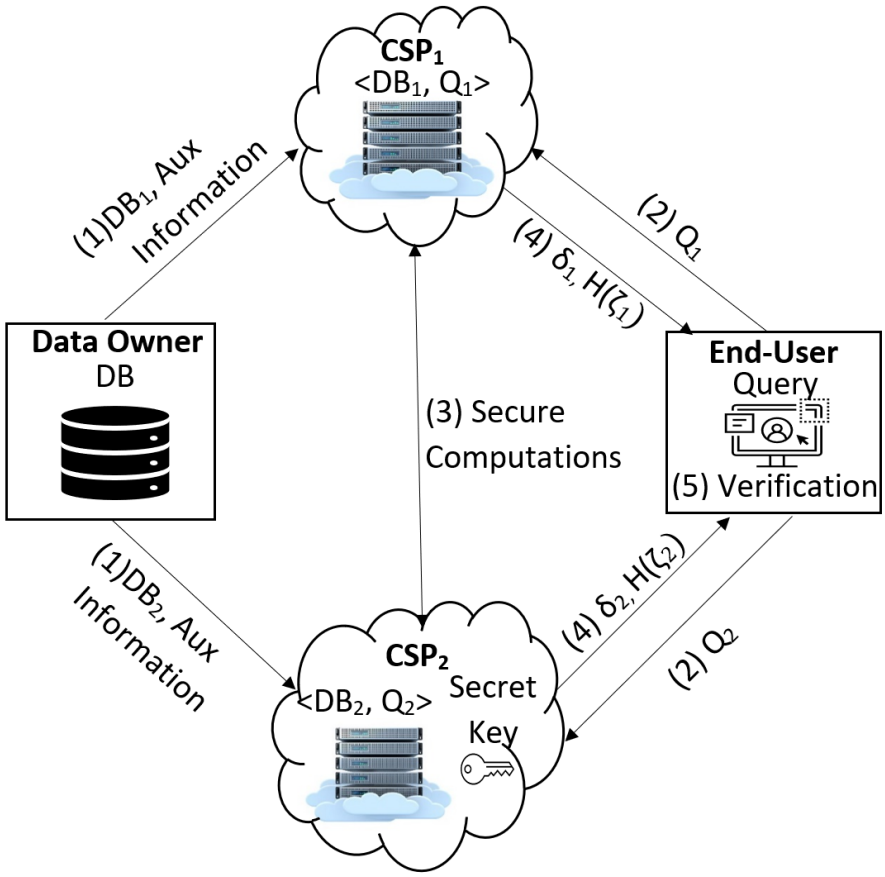

Figure 1: System Model

1.2 Problem Definition

Suppose a data owner owns a database DB of n records, denoted by t_1,..., t_n, and m attributes. Let $t_{i,j}$ represents the j^{th} attribute value of record t_i. Before outsourcing, the data owner adds n_f fake tuples to the DB, randomly splits the database attribute-wise into two shares DB_1 and DB_2, and sends them to a Cloud Service Provider$_1$, CSP_1 and Cloud Service Provider$_2$, CSP_2, respectively. Moreover, future query processing tasks are delegated to the clouds. Similarly, an authorized end-user randomly splits the query Q = $\langle Q_1,...,Q_m \rangle$ attribute-wise into two shares Q_1 and Q_2, and sends them to CSP_1 and CSP_2,

respectively. The two clouds collaborate to obtain the $E_{pk}(DB)$ and $E_{pk}(Q)$, compute the top k records, and send the results to the end-user. It is important to mention that the two clouds do not collude with each other.

In the SVC-kNN protocol, the data owner generates auxiliary information once during the Database Outsourcing Stage. The clouds will use this information to verify the computed top k records. Specifically, CSP_1 and CSP_2 employ the auxiliary information to construct the proofs δ_1, δ_2, $H(\zeta_1)$, and $H(\zeta_2)$. In addition to the computation results, the clouds send the proofs to the end-user for verification purposes. We divide the verification process into the Euclidean Distance Verification and Sort Verification. We emphasize that our verification scheme is probabilistic and the proofs are constructed using only fake tuples. We assume that if verification were successful, the returned results are correct with a high probability. Moreover, it is important to mention that the contents of Q_1, Q_2, DB_1 and DB_2 remain confidential throughout the query processing and verification phases. Furthermore, the data access patterns are also hidden from the clouds. Figure 1 shows the information flow across different parties in the proposed framework.

1.3 Our Contributions

We propose a novel SVC-kNN protocol to verify the correctness of the top k records. Firstly, we verify the results of the computed Euclidean distances (d_i), and we prove that our theorem always holds for validating euclidean distances. Secondly, we verify the sorted distances. Moreover, the SVC-kNN protocol minimizes the workload and the encryption cost on the data owner and the end-user because it does not require them to directly encrypt DB and Q. Instead, it allows them to randomly split their data and send each share to the corresponding CSP. Subsequently, CSP_1 and CSP_2 coordinate to obtain $E_{pk}(DB)$

and $E_{pk}(Q)$, which improves the efficiency. Besides, the data owner is not required to participate in kNN computation and result verification. Furthermore, the protocol is secure under the semi-honest model and satisfies the following requirements:

- Data confidentiality – The contents of DB and Q should remain confidential at all times.
- Hide data access patterns from the cloud – information about the k-nearest tuples to the Q should not be revealed to any intermediate party.
- Accurately compute the k-nearest neighbors of query Q.
- Minimize computation overhead on the data owner.
- Final results should not be revealed to the cloud.
- Accurately generate proofs for verifying the computed results.

We emphasize that our probabilistic verification scheme utilizes two cloud service providers, CSP_1 and CSP_2, to construct the proofs. Each CSP sends its proof to the end-user for each verification stage: Euclidean Distance Verification and Sort Verification. The end-user obtains proofs for each verification step to check the equality. It is important to mention that the clouds start the sort verification only if the Euclidean distance verification is successful.

1.4 Organization

The remaining sections are organized as follows. Section 2 reviews related works. Section 3 reviews preliminaries. Section 4 describes the problem and the proposed framework for secure computation of kNN and secure verification of kNN. Section 5 focuses on the verification scheme. We divide the verification into three subsections: Proof Preparation, Euclidean Distance Verification and Sort Verification. Finally, we conclude the paper with future work.

2 Related Works

2.1 Secure kNN Outsourcing

Researchers have addressed the problem of outsourcing SkNN problem in several papers. Wong et al. [10] proposed the Asymmetric Scalar-product Preserving Encryption (ASPE) scheme, which compares distances by preserving scalar product between the query Q and tuples t in the database DB. The database and queries are encrypted and outsourced to the cloud. However, the decryption key sk is revealed to the end-user and the scheme is vulnerable to chosen-plaintext attacks. Zhu et al. [12] proposed an improved SkNN scheme that does not reveal the decryption key. However, it requires the end-user to involve in heavy computations during the query processing step, which contradicts the concept of outsourcing the computations to the cloud. Furthermore, it releases partial information about the key to end-users while performing kNN computation. Yao et al. [11] proposed a SkNN scheme that depends on partition-based secure Voronoi diagram (SVD). In their proposed solution, they required the cloud to return a relevant encrypted partition $E_{pk}(G)$ for $E_{pk}(T)$ in which G is guaranteed to contain the k-nearest neighbors to the Q. However, this method returns inaccurate results to the end-user. Besides, it requires the end-user to involve in the query processing phase, which is inefficient. Moreover, the proposed solutions in [10], [9], [12] do not protect the data access pattern from the cloud. Samanthula et al. [8] proposed a novel SkNN protocol based on the Paillier Cryptosystem's homomorphic properties. Their scheme solves the SkNN accurately and guarantees the confidentiality of data and queries. It also hides the data access patterns from the cloud; however, it requires the data owner to encrypt the database, which consumes the data owner's resources.

2.2 Verifiable Computation on kNN

Wong et al. [15] addressed the problem of the verification of outsourced frequent itemset mining. They constructed a set of fake (in)frequent itemsets by inserting fake items into the outsourced database, and check against the fake (in)frequent itemsets to verify the mining results. They assumed that the server do not have knowledge about these itemsets in the outsourced datasets. However, the server may be able to collect information about the outsourced database and escape from the verification step [16], [13]. Dong et al. [13] adopted a a slightly different method to verify outsourced frequent itemset mining. They constructed the set of (in)frequent itemsets from the real items. They used this set as evidence to check the integrity of the mining results returned by the cloud. Nevertheless, in SkNN, mining results depend on the query Q given by the end-user. Therefore, using the aforementioned methods for verifying kNN results is not possible. Wang et al. [14] proposed a verifiable approach of secure kNN computation. Their scheme verifies the outsourced kNN computation by utilizing the algebraic properties of Asymmetric Scalar Product Encryption Scheme; however, this scheme is proved to be insecure against chosen and known plaintext attacks [10]. Moreover, their verification entirely depends on one server, assuming that it is a semitrusted party. We argue that such assumption may not stand in practice, as the server may not be able to successfully verify results due to (1) mistakes during verification phase, (2) financial incentives, (3) or malicious activities. Furthermore, their scheme reveals information about indices of fake tuples and does not hide data access patterns from the verifying server.

3 Preliminaries

3.1 Paillier Cryptosystem

The Paillier cryptosystem is a probabilistic asymmetric encryption scheme [17]. Data is encrypted using the public key, pk, and decrypted using the secret key, sk. Let N be the product of two large prime numbers, p and q, that is, $N = p * q$ and $\phi(N)$ denotes the Euler function, which is equal to $\phi = (p-1) * (q-1)$ such that $gcd(N, \phi(N)) = 1$. Consider g is the generator in the group $\mathbb{Z}^*{}_{N^2}$. The public key $pk = (N, g)$, and the private key $sk = (\lambda, \mu)$, in which, $\lambda = lcm(p-1, q-1)$ and $\mu = (L(g^\lambda \bmod N^2))^{-1} \bmod N$, where $L(x) = \frac{x-1}{N}$.

The paillier encryption scheme has the following properties:

1. **Homomorphic Addition**

   $E_{pk}(a+b) \leftarrow E_{pk}(a) * E_{pk}(b) \mod N^2; \quad a, b \in \mathbb{Z}_N$

2. **Homomorphic Multiplication**

   $E_{pk}(a*b) \leftarrow E_{pk}(a)^b \bmod N^2; \quad a, b \in \mathbb{Z}_N$

3. **Semantic Security**

   The encryption scheme is semantically secure [7]. Given a ciphertext, an adversary cannot learn any information about the plaintext.

3.2 Random Splitting of Data

Consider Bob has x and wants to randomly split it into two shares, $x_1 = x + r \bmod N$ and $x_2 = N - r$, and sends them to CSP_1 and CSP_2, respectively. Let r be a random number selected by Bob and N be the group size. The summation of the two shares yields the original value x, that is, $(x + r) + (N - r) \bmod N = x$.

3.3 Euclidean Distance (d)

Euclidean distance (d) is a metric that measures the distance between two data points. Considering two vectors $X = \langle x_1, x_2, ..., x_m \rangle$ and $Y = \langle y_1, y_2, ..., y_m \rangle$:

$$d(X, Y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

In this paper, we assume that a data owner outsources the encryption task to two semi-honest clouds: CSP_1 and CSP_2. We assume that CSP_2 holds the secret key sk, and all the encryption is done under the Paillier cryptosystem. Table I summarizes some notations that will be used extensively in this paper.

Table 1: Notations

pk	CSP_2's public key
sk	CSP_2's secret key
DB_1	CSP_1's share of DB
DB_2	CSP_2's share of DB
Q_1	CSP_1's share of Q
Q_2	CSP_2's share of Q
$E_{pk}(d_i)$	The encryption of the Euclidean Distance of tuple t_i
n_f	Number of fake tuples inserted into DB
ϵ_{a_1} and ϵ_{b_1}	Vectors with n values to randomize the t_i and d_i
ϵ_{a_2} and ϵ_{b_2}	Vectors with n_f values to randomize the fake tuples and the fake distances
ϵ_c	Vector with n numbers to randomize $(t_{i,j} - Q_j)$
ϵ_{d_1}	Vector contains all possible combination of fake tuples
ϵ_{d_2}	Vector contains the summation of each possible combination of fake tuples
μ	Represents the aggregate value of the randomized fake tuples
β	The summation of all possible combinations of fake tuples in two pairs
δ_1 and δ_2	Proofs of the Euclidean Distance verification
$H(\zeta_1)$ and $H(\zeta_2)$	Proofs of sort verification

4 Problem Description and Framework

In this section, we introduce our system model, and the framework of our solution, which consists of four stages:

• Stage 1 - Database Outsourcing: CSP_1 and CSP_2 encrypt their share of the database, DB_1 and DB_2, and the latter sends its encrypted share to CSP_1. Then, CSP_1 adds the two encrypted randomized data, $E_{pk}(DB_1)$ and $E_{pk}(DB_2)$, attribute-wise to obtain the original database $E_{pk}(DB)$ in the encrypted format.

• Stage 2 - Query Outsourcing: Similar to the previous stage, each CSP encrypts its share of the query, Q_1 and Q_2. At the end, CSP_1 obtains $E_{pk}(Q)$.

• Stage 3 - Secure Computation of kNN: This stage consists of Secure Computation of the Euclidean Distance (d) and Secure Sorting of the Euclidean Distances.

• Stage 4 - Secure Verification of kNN: This stage consists of Proof Preparation, Euclidean Distance Verification, and Sort Verification.

The stages are explained in detail in the subsections.

4.1 Stage 1 - Database Outsourcing

Consider a data owner owns a database DB that has n records and m attributes, and randomly splits it attribute-wise, in which, each attribute is divided into two shares, $share_1$ and $share_2$. Let $share_1$ denotes DB_1 and $share_2$ denotes DB_2. Let N be the group size and $r_{i,j}$ be a set of random numbers selected by the data owner, in which, $1 \leq i \leq n$ and $1 \leq j \leq m$. Algorithm 1 summarizes the steps of random splitting.

Algorithm 1 Random Splitting (DB, $r_{i,j}$, N) $\rightarrow$ DB_1, DB_2

1: **Initialize:** Array DB_1, DB_2
2: **for** i = 1 to n **do**
3: **for** j = 1 to m **do**
4: $DB_{1_{i,j}} = t_{i,j} + r_{i,j} \bmod N$
5: $DB_{2_{i,j}} = N - r_{i,j}$
6: **return** DB_1, DB_2

Then, the data owner sends DB_1 and DB_2 to CSP_1 and CSP_2, respectively. Specifically, CSP_1 receives $t_{i,j} + r_{i,j} \bmod N$ and CSP_2 receives $N - r_{i,j}$. We emphasize that the summation of these two shares represents the original value $t_{i,j}$ as shown below:

$$(t_{i,j} + r_{i,j}) + (N - r_{i,j}) \mod N \equiv t_{i,j}$$

CSP_2 will encrypt DB_2 using pk and send $E_{pk}(DB_2)$ to CSP_1. The latter will also encrypt its share, DB_1, using pk. The next step is to use the additive homomorphic property to obtain $E_{pk}(DB)$; that is, $E_{pk}(DB_1 + DB_2) = E_{pk}(DB_1) * E_{pk}(DB_2) \bmod N^2$. We emphasize that the addition is attribute-wise and the result is known to only CSP_1. Table I and II show the values of DB_1 and DB_2 after randomly splitting DB in Example 1 from [8].

Table 2: Sample Heart Disease Dataset (DB_1)

record-id	age	sex	cp	tresbps	chol	fbs	slope	ca	thal	num
t_1	63 + $r_{1,1}$	1 + $r_{1,2}$	1 + $r_{1,3}$	145 + $r_{1,4}$	233 + $r_{1,5}$	1 + $r_{1,6}$	3 + $r_{1,7}$	0 + $r_{1,8}$	6 + $r_{1,9}$	0 + $r_{1,10}$
t_2	56 + $r_{2,1}$	1 + $r_{2,2}$	3 + $r_{2,3}$	130 + $r_{2,4}$	256 + $r_{2,5}$	1 + $r_{2,6}$	2 + $r_{2,7}$	1 + $r_{2,8}$	6 + $r_{2,9}$	2 + $r_{2,10}$
t_3	57 + $r_{3,1}$	0 + $r_{3,2}$	3 + $r_{3,3}$	140 + $r_{3,4}$	241 + $r_{3,5}$	0 + $r_{3,6}$	2 + $r_{3,7}$	0 + $r_{3,8}$	7 + $r_{3,9}$	1 + $r_{3,10}$
t_4	59 + $r_{4,1}$	1 + $r_{4,2}$	4 + $r_{4,3}$	144 + $r_{4,4}$	200 + $r_{4,5}$	1 + $r_{4,6}$	2 + $r_{4,7}$	2 + $r_{4,8}$	6 + $r_{4,9}$	3 + $r_{4,10}$
t_5	55 + $r_{5,1}$	0 + $r_{5,2}$	4 + $r_{5,3}$	128 + $r_{5,4}$	205 + $r_{5,5}$	0 + $r_{5,6}$	2 + $r_{5,7}$	1 + $r_{5,8}$	7 + $r_{5,9}$	3 + $r_{5,10}$

Table 3: Sample Heart Disease Dataset (DB_2)

record-id	age	sex	cp	tresbps	chol	fbs	slope	ca	thal	num
t_1	$N - r_{1,1}$	$N - r_{1,2}$	$N - r_{1,3}$	$N - r_{1,4}$	$N - r_{1,5}$	$N - r_{1,6}$	$N - r_{1,7}$	$N - r_{1,8}$	$N - r_{1,9}$	$N - r_{1,10}$
t_2	$N - r_{2,1}$	$N - r_{2,2}$	$N - r_{2,3}$	$N - r_{2,4}$	$N - r_{2,5}$	$N - r_{2,6}$	$N - r_{2,7}$	$N - r_{2,8}$	$N - r_{2,9}$	$N - r_{2,10}$
t_3	$N - r_{3,1}$	$N - r_{3,2}$	$N - r_{3,3}$	$N - r_{3,4}$	$N - r_{3,5}$	$N - r_{3,6}$	$N - r_{3,7}$	$N - r_{3,8}$	$N - r_{3,9}$	$N - r_{3,10}$
t_4	$N - r_{4,1}$	$N - r_{4,2}$	$N - r_{4,3}$	$N - r_{4,4}$	$N - r_{4,5}$	$N - r_{4,6}$	$N - r_{4,7}$	$N - r_{4,8}$	$N - r_{4,9}$	$N - r_{4,10}$
t_5	$N - r_{5,1}$	$N - r_{5,2}$	$N - r_{5,3}$	$N - r_{5,4}$	$N - r_{5,5}$	$N - r_{5,6}$	$N - r_{5,7}$	$N - r_{5,8}$	$N - r_{5,9}$	$N - r_{5,10}$

4.2 Stage 2 - Query Outsourcing

The steps of query outsourcing are similar to the Database Outsourcing Stage. Briefly, an end-user randomly splits the query into Q_1 and Q_2, and outsources them to CSP_1 and CSP_2, respectively. CSP_2 encrypts its share and sends $E_{pk}(Q_2)$ to CSP_1. CSP_1 obtains $E_{pk}(Q_1)$ and then utilizes the additive homomorphic property to obtain $E_{pk}(Q)$.

Algorithm 2 shows the complete steps for data preparation: Database Outsourcing (stage1) and Query outsourcing (stage 2). We emphasize that the randomization and the encryption of data are computed attribute-wise.

Algorithm 2 Data Preparation (DB_1, DB_2, Q_1, Q_2) $\rightarrow$ $E_{pk}(DB), E_{pk}(Q)$

1: The data owner sends DB_1 to CSP_1 and DB_2 to CSP_2
2: CSP_1 and CSP_2:
   - CSP_2 encrypts DB_2 and sends $E_{pk}(DB_2)$ to CSP_1
   - CSP_1 encrypts DB_1
3: **for** i = 1 to n **do:**
   - **for** j = 1 to m **do:**
     - CSP_1 performs:
       - $E_{pk}(t_{i,j}) = E_{pk}(t_{i,j})_1 * E_{pk}(t_{i,j})_2 \mod N^2$
5: Bob sends Q_1 to CSP_1 and Q_2 to CSP_2
   - CSP_2 encrypts Q_2 and send $E_{pk}(Q_2)$ to CSP_1
   - CSP_1 encrypts Q_1
6: **for** j = 1 to m **do:**
   - CSP_1 **do**
     - $E_{pk}(Q) = E_{pk}(Q_j)_1 * E_{pk}(Q_j)_2 \mod N^2$;

4.3 Stage 3 - Secure Computation of kNN

This paper slightly modifies the proposed scheme of computing SkNN in [8] and focuses on proposing a probabilistic approach for verifying SkNN returned results. For the computation phase, we will adopt Algorithm 6 proposed in [8]; however, we do not require the data owner and the end-user to encrypt DB and Q. Specifically, we outsource the encryption task to the cloud to reduce the workload on the data owner and the end-user. Besides, we added a few steps to Algorithm 6 in [8] to combine the computation and the verification phases. The following are the various stages in the secure computation of kNN.

4.3.1 Secure Computation of the Euclidean Distance d_i

The clouds start computing the Euclidean Distance between $E_{pk}(Q)$ and $E_{pk}(t_i)$ where $1 \leq i \leq n$. The protocols used to compute the $E_{pk}(d_i)$ are briefly explained below. Interested readers can refer to [8] for more details.

- Secure Multiplication (SM):

  Consider CSP_1 sends $(E_{pk}(a), E_{pk}(b))$ to CSP_2 for computing secure multiplication. It selects different random numbers, r_a and r_b, to randomize $E_{pk}(a)$ and $E_{pk}(b)$. Then, it sends the randomized encrypted values to CSP_2 that decrypts, multiplies the numbers, encrypts the result and sends it to CSP_1. Subsequently, CSP_1 utilizes the additive homomorphic property to remove the randomness from the result as shown below:

  $$a * b = (a + r_a) * (b + r_b) - a * r_b - b * r_a - r_a * r_b$$

  The output $E_{pk}(a * b)$ is known only to CSP_1.

- Secure Squared Euclidean Distance (SSED):

  Consider CSP_1 has a tuple, $E_{pk}(t_i) = \langle E_{pk}(t_1), E_{pk}(t_2), \ldots, E_{pk}(t_m)\rangle$ and CSP_2 has a query $E_{pk}(Q) = \langle E_{pk}(Q_1), E_{pk}(Q_2), \ldots, E_{pk}(Q_m)\rangle$, then, the Squared Euclidean Distance between $E_{pk}(t_i)$ and $E_{pk}(Q)$ is:

$$\prod_{j=1}^{m} E_{pk}(t_{i,j} - Q_j)^2$$

  CSP_1 computes $E_{pk}(t_{i,j} - Q_j)$, and sends the result to CSP_2 for secure multiplication, SM($E_{pk}(t_{i,j} - Q_j), E_{pk}(t_{i,j} - Q_j)$). At the end of this protocol, CSP_1 adds the squared differences and computes the final distance between $E_{pk}(t_i)$ and $E_{pk}(Q)$.

4.3.2 Secure Sorting of the Euclidean Distance d_i

The following protocols are utilized to sort the Euclidean Distances:

- Secure Bit-Decomposition (SBD):

  This protocol is adopted from [18]. It takes the encryption of a digit as input and outputs the encryption of individual bits that represent this digit. Let CSP_1 has $E_{pk}(x)$ and CSP_2 has sk. SBD should outputs a vector that contains the encryption of individual bits that represent x, $[x] = \langle E_{pk}(x_1), ..., E_{pk}(x_l)\rangle$ where 1 and l represent the most and the least significant bits of $[x]$. The final result is only known to CSP_1.

- Secure Minimum (SMIN):

  This algorithm takes two encrypted vectors as input and outputs the minimum value. Consider $[x]$ and $[y]$ are two vectors that represent the encryption of individual bits of two values, x and y, respectively. CSP_1 has $([x], [y])$ and CSP_2 holds sk. The

protocol outputs [min(x, y)], and it is only known to CSP_1.

- Secure Minimum out of n Numbers (SMIN$_n$)

  This algorithm takes all the distances, in the encrypted format of each individual bit, between the query and each t_i as input and outputs the smallest distance d_i. Consider CSP_1 has ($[d_1]$,...,$[d_n]$) and CSP_2 has sk where $1 \leq i \leq n$, and for each vector $[d_i]$ = ($E_{pk}[d_{i,1}]$,...,$E_{pk}[d_{i,l}]$) , in which, 1 and l represent the most and the least significant bits of $[d_i]$, respectively. The output [min$(d_1, ..., (d_n)$] is known to only CSP_1.

It is important to mention that the above protocols are directly adopted from the literature. In SVC-kNN, we added Stage (1) and Stage (2) to outsource the encryption of DB and Q to the cloud. The purpose of adding these stages is to maximize the use of the clouds and reduce the workload on the data owner and the end-user which is more efficient.

After the data preparation phase explained in Algorithm 2, CSP_1 and CSP_2 will start the Secure Computation of kNN. In general, for each encrypted tuple $E_{pk}(t_i)$, CSP_1 and CSP_2 will compute the SSED, that is:

1. CSP_1 has $E_{pk}(Q)$ and $E_{pk}(t_i)$.
2. CSP_1 computes the difference between the query and each tuple attribute-wise.
3. CSP_1 sends the computed differences to CSP_2 for secure multiplication, SM($E_{pk}(t_{i,j} - Q_j)$, $E_{pk}(t_{i,j} - Q_j)$).
4. Upon receiving the SM results from CSP_2, CSP_1 will use the additive homomorphic property to obtain $E_{pk}(d_i)$, which is equivalent to $E_{pk}(t_i - Q)^2$. This result is only known to CSP_1.

Then, CSP_1 and CSP_2 start the Secure Sorting of the Euclidean Distance d_i. For each distance d_i, the clouds execute SBD to compute the encryption of individual bits that represent the encryption of the distance, in which $E_{pk}[d_i] = \langle E_{pk}(d_{i,1}), E_{pk}(d_{i,2}),...., E_{pk}(d_{i,l})\rangle$

where 1 and l represent the most and the least significant bits of the distance. The clouds execute this protocol n times, and pass the outputs as parameters to the SMIN$_n$ protocol. We emphasize that all data is encrypted, all the computations are on the encrypted data, and the outputs of all the protocols are only known to CSP_1.

After that, CSP_1 and CSP_2 will compute the k-nearest records to $E_{pk}(Q)$ in an iterative way. The steps are similar to Algorithm 6 in [8] with slight variations and are shown in Algorithm 3.

Briefly, CSP_1 and CSP_2 executes SMIN protocol n times instead of k times in order obtain the complete sorted distances. Then, CSP_1 computes the difference between the minimum distance d_{min} and every other distance d_i, permutes them and sends the encrypted vector β to CSP_2. Upon receiving β, CSP_2 decrypts and observes only one attribute with the value 0. Then, it computes vector U such that the attribute containing value 0 is updated as $E_{pk}(1)$ and the other attributes as $E_{pk}(0)$, and sends U to CSP_1.

CSP_1 performs the inverse permutation after receiving U. It then raises each encrypted value in vector V to the power of its respective index and then applies the summation. Using the additive homomorphic property of pailler cryptosystem, CSP_1 performs this operation by itself to obtain the sorted encrypted indices. The distance vectors are updated for every iteration using SBOR protocol.

CSP_1 obtains the encryption of all the sorted indices, permutes them and sends them to CSP_2 for the verification step. Besides, during the verification step, CSP_1 obtains the hashes of the indices of fake tuples $H(\xi)$ from CSP_2. It is important to mention that the SVC-kNN requires this information to obtain the final k nearest records. Specifically, CSP_1 checks if any of the hashes of fake tuples $H(\xi_i)$ is equal to any of the hashes of sorted indices i'_s. If this statement were executed, CSP_1 adds 1 to ctr. At the end of the iterations, CSP_1 updates k; that is, k' = k + ctr.

Finally, CSP_1 obtains the top k' records, discards the fake tuples, and sends k real records to user. More Specifically, CSP_1 executes SM protocol of vector V and actual tuple, and then applies a summation. This yields only one tuple that corresponds to the minimum distance and the process is repeated for k' iterations where k' is the updated value of k based on the existence fake tuples. Finally, CSP_1 removes the tuples that correspond to the hashes of fake indices, and sends the actual k nearest records to the end-user.

4.4 Stage 4 - Secure Verification of kNN

This stage concentrates on verifying the integrity of the computed k-nearest records. Similar to the Secure Computation of kNN, CSP_1, CSP_2, the data owner, and the end-user participate in the Secure Verification of kNN. As mentioned earlier, during the Database Outsourcing Stage, the data owner generates auxiliary information, randomly splits it, and sends the shares to CSP_1 and CSP_2. Moreover, the clouds require a variety of information to construct the proofs δ_1, δ_2, $H(\zeta_1)$, and $H(\zeta_2)$. In detail, the information used in evidence construction is, the encrypted Euclidean Distance $E_{pk}(d_i)$, four random vectors, ϵ_{a_1} and ϵ_{a_2},ϵ_{b_1}, and ϵ_{b_2}, vector μ that represents the aggregate values of the randomized fake tuples, and vector β that represents the aggregate of the product of all possible combinations of μ in two pairs. We emphasize that the Secure Verification of kNN involves CSP_1 and CSP_2 in constructing evidence such that each evidence requires different steps and distinct variables. Our solution in this paper incorporates several verification steps: Proof Generation, Evidence Preparation, Euclidean Distances Verification, and Sort Verification, which we will discuss in the next section. It is important to mention that the computation and the verification steps are closely associated. However, in this paper, they are explained separately for more clarity and convenience.

Algorithm 3 SkNN$_m$ $(d_i) \rightarrow \mathrm{I}_{sorted}$, $\langle\ (t'_1)$, ..., $(t'_{k'})\ \rangle$

1: CSP_1 and CSP_2:

(a). CSP_1 obtains $E(Q)$ in Algorithm 2

(b). **for** i = 1 to n **do:**

- $E_{pk}(d_i) \leftarrow$ SSED($E_{pk}(Q)$, $E_{pk}(t_i)$)
- $[d_i] \leftarrow$ SBD($E_{pk}(d_i)$)

2: **for** s = 1 to n **do:**

(a). CSP_1 and CSP_2:

- $[d_{min}] \leftarrow$ SMIN$_n$([d_1], ..., [d_n])

(b). CSP_1:

- $E_{pk}(d_{min}) \leftarrow \prod_{\gamma=0}^{l-1} E_{pk}(d_{min}, \gamma+1)^{2^{l-\gamma-1}}$
- **if** $s \neq 1$ **then,** for $1 \leq i \leq n$
  - $E_{pk}(d_i) \leftarrow \prod_{\gamma=0}^{l-1} E_{pk}(d_i, \gamma+1)^{2^{l-\gamma-1}}$
- **for** i = 1 to n **do:**
  - $\tau_i \leftarrow E_{pk}(d_{min})$ * $E_{pk}(d_i)^{N-1}$
  - $\tau'_i \leftarrow \tau_i^{r_i}$, where $r_i \in_R \mathbb{Z}_N$
- $\beta \leftarrow \pi(\tau')$; send β to CSP_2

(c). CSP_2 :

- Receives β from CSP_1
- $\beta'_i \leftarrow D_{sk}(\beta_i)$, for $1 \leq i \leq n$
- Computes U, for $1 \leq i \leq n$
  - **if** β'_i = 0 **then** U_i = $E_{pk}(1)$
  - **else** U_i = $E_{pk}(0)$
- Send U to CSP_1

(d). CSP_1:

- Receive U from CSP_2 and compute $V \leftarrow \pi^{-1}(U)$
- $V'^i = V_i^i$
- $E_{pk}(i_s) \leftarrow \prod_{i=1}^{n} V'_i$

(e). CSP_1 and CSP_2, for $1 \leq i \leq n$:

- $E_{pk}(d_i, \gamma) \leftarrow$ SBOR(V_i, $E_{pk}(d_i, \gamma)$), for $1 \leq \gamma \leq 1$

3: CSP_1 :

(a). Based on the sort verification step, the value of k' is determined.

- **for** i = 1 to k **do:**
- **for** j = 1 to n_f **do:**
- **if** $H(i_s) = H(\xi_j)$
- $ctr++$
- k' = k + ctr

4: **for** s = 1 to k' **do:**

(a). CSP_1 :

- $V'_{i,j} \leftarrow$ SM(V_i, $E_{pk}(t_{i,j})$), for $1 \leq i \leq n$ and $1 \leq j \leq m$
- $E_{pk}(t'_{s,j}) \leftarrow \prod_{i=1}^{n} V'_{i,j}$
- $E_{pk}(t'_s)$ = $\langle\ E_{pk}(t'_{s,1})$, ..., $E_{pk}(t'_{s,m})\ \rangle$

5 Verification Scheme

In this section, we design a probabilistic verification approach that verifies the kNN results. More specifically, this approach captures any unexpected behavior that may return wrong k-nearest records. The proposed scheme utilizes the computational resources of the cloud and avoids assigning heavy computations to the data owner. Figure 2 shows the Verification Scheme. The following explains the verification stages in detail.

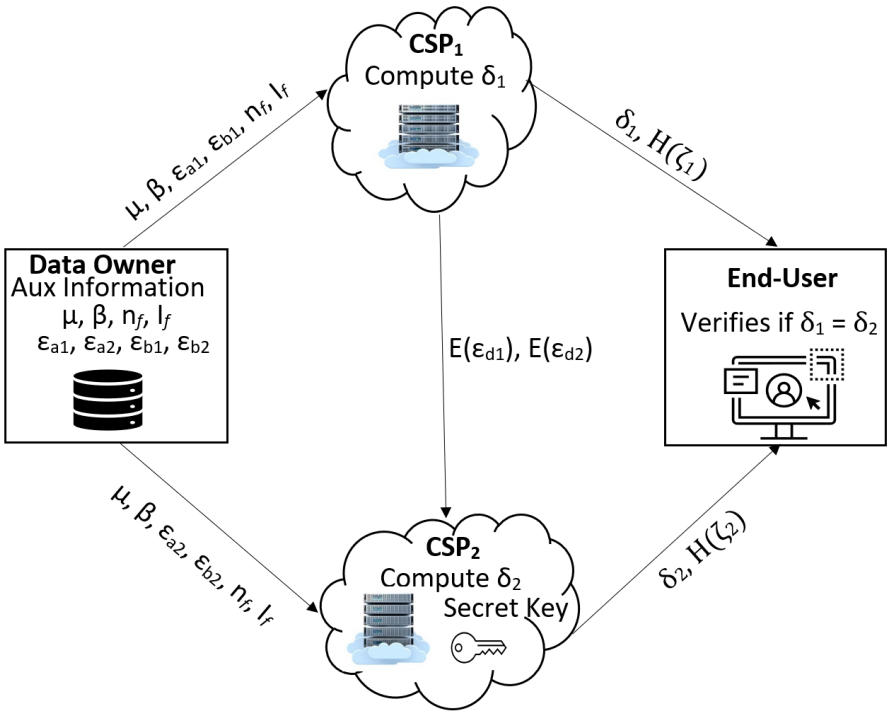

Figure 2: Verification Scheme

5.1 Proof Preparation

As mentioned in the Database Outsourcing Stage, the data owner inserts n_f artificial tuples into the dataset; that is, the randomly split DB includes both the real and the fake tuples.

The data owner records the number of fake tuples in n_f, and the indices of fake tuples in a vector [I_f]. The reason behind adding these tuples is to use them in (1) The Euclidean Distance Verification and (2) Sort Verification. Moreover, the data owner constructs the following:

- A random vector ϵ_{a_1}; that is, $\epsilon_{a_1} = \langle \epsilon_{a_{11}}, ..., \epsilon_{a_{1n}} \rangle$. This vector contains n random numbers for randomizing all the real and the fake tuples. This vector is only known to CSP_1.
- A random vector ϵ_{a_2}; that is, $\epsilon_{a_2} = \langle \epsilon_{a_{21}}, ..., \epsilon_{a_{2nf}} \rangle$. We highlight that ϵ_{a_2} is a subset of ϵ_{a_1}; that is, $\epsilon_{a_2} \subset \epsilon_{a_1}$, and stores n_f random numbers for only the fake tuples. This vector is only known to CSP_2.
- A random vector ϵ_{b_1}; that is, $\epsilon_{b_1} = \langle \epsilon_{b_{11}}, ..., \epsilon_{b_{1n}} \rangle$. This vector stores n random numbers for randomizing all the Euclidean Distances $E_{pk}(d_i)$. This vector is only known to CSP_1.
- A random vector ϵ_{b_2}; that is, $\epsilon_{b_2} = \langle \epsilon_{b_{21}}, ..., \epsilon_{b_{2nf}} \rangle$. This vector stores n_f random numbers, and is a subset of ϵ_{b_1}; that is, $\epsilon_{b_2} \subset \epsilon_{b_1}$. CSP_2 requires this vector to remove the randomness from the randomized Euclidean Distances d_{i_r} between Q and each fake tuple. This vector is only known to CSP_2.
- A verification vector, $\mu = \sum_{i=1}^{n_f} (\epsilon_{a_{2i}} + t_i)$.
- A verification vector, $\beta = \sum_{i=1}^{n-1} (\epsilon_{a_{2i}} + t_i) * \left[\sum_{i=i+1}^{n} (\epsilon_{a_{2i}} + t_i)\right]$

It is important to mention that μ and β are only known to CSP_1. Algorithm 4 shows the construction of μ. Let n be 5 and n_f be 3, meaning that there are only three artificial tuples and two real tuples in DB. Let N be the group size; then, ϵ_{a_1} = $\langle$ $\epsilon_{a_{11}}, \epsilon_{a_{12}}, \epsilon_{a_{13}}, \epsilon_{a_{14}}, \epsilon_{a_{15}}$ $\rangle$ and ϵ_{a_2} = $\langle$ $\epsilon_{a_{21}}, \epsilon_{a_{22}}, \epsilon_{a_{23}}$ $\rangle$. Consider (x_1, y_1), (x_2, y_2), (x_3, y_3) three fake tuples that represent t_1, t_2, and t_3, respectively. μ_1 = (x_1 + $\epsilon_{a_{11}}$, y_1 + $\epsilon_{a_{11}}$), μ_2 = (x_2 + $\epsilon_{a_{12}}$, y_2 + $\epsilon_{a_{12}}$), μ_3 = (x_3 + $\epsilon_{a_{13}}$, y_3 + $\epsilon_{a_{13}}$); then μ is the summation of μ_1, μ_2, and μ_3:

$$\mu = [(x_1 + \epsilon_{a_{11}}) + (x_2 + \epsilon_{a_{12}}) + (x_3 + \epsilon_{a_{13}}), (y_1 + \epsilon_{a_{11}}) + (y_2 + \epsilon_{a_{12}}) + (y_3 + \epsilon_{a_{13}})]$$

Algorithm 4 Construction of μ

1: **Require:** ϵ_{a_2}, t_i
2: **Initialize:** μ
3: **for** i = 1 to n_f **do:**
- **Initialize:** μ_i
- **for** j = 1 to m **do:**
  - $\mu_{i,j} \leftarrow \epsilon_{a_{2i}} + t_{i,j}$
- μ += μ_i

Then, using the information of μ, the data owner construct vector β, such that β = (μ_1 * μ_2) + (μ_1 * μ_3) + (μ_2 * μ_3). We emphasize that the number of combinations of μ depends on the number of fake tuples; and is based on the combination permutation formula, $c(n, r) = \frac{n!}{(n-r)!*r!}$, where n = n_f and r = 2 because the number of pairs is two.

We emphasize that the data owner randomly splits all the auxiliary information between CSP_1 and CSP_2. Then, CSP_1 and CSP_2 consolidate the randomly split auxiliary information to obtain the encryption of each one. For the information known to CSP_1, CSP_2 encrypts its share and sends the encrypted values to CSP_1. CSP_1 also encrypts its share, and then obtains the encryption of the original value using the additive homomorphic property, and vice-versa for the information known to CSP_2.

5.2 Euclidean Distance Verification

Each CSP is responsible for constructing and sending the evidence to the end-user. Based on the auxiliary information, CSP_1 and CSP_2 construct δ_1 and δ_2, respectively. Upon receiving the proofs, the end-user verifies if Δ = δ_1 - δ_2 = 0, that is if, δ_1 = δ_2. Besides, each cloud utilizes different parameters to construct its proof; however, both equations must compute the exact final result. In this section, we produce a novel theorem to verify the computation of $E_{pk}(d_i)$. We show that $\delta_1 \equiv \delta_2$ using Proof by Induction methodology.

Interested readers can refer to Appendix A.

$$\delta_1 = \sum_{i=1}^{m} \left[\mu * (\mu - 2Q) - 2\beta + n_f \left(Q^2\right)\right] \tag{1}$$

$$\delta_2 = \sum_{i=1}^{n_f} \left[d_i + \epsilon_{a_{2_i}} \left(m * \epsilon_{a_{2_i}} + 2\sum_{j=1}^{m} (t_{i,j} - Q_j)\right)\right] \tag{2}$$

Algorithm 5 δ_1 Computation $(\mu, \beta, Q, n_f) \rightarrow \delta_1$

1: **Requie:** CSP_1 has μ, β, Q, n_f and CSP_2 has sk
2: **Initialize:** r
3: CSP_1 computes:
- $E_{pk}(2Q) \leftarrow E_{pk}(Q) * E_{pk}(Q) \bmod N^2$
- $E_{pk}(A) \leftarrow E_{pk}(\mu) * E_{pk}(2Q)^{N-1}$
- $E_{pk}(B) \leftarrow E_{pk}(\beta) * E_{pk}(\beta) \bmod N^2$

4: CSP_1 and CSP_2 performs:
- $E_{pk}(C) \leftarrow$ SM$(E_{pk}(\mu), E_{pk}(A))$, send $E_{pk}(C)$ to CSP_1
- $E_{pk}(D) \leftarrow$ SM$(E_{pk}(Q), E_{pk}(Q))$, send $E_{pk}(D)$ to CSP_1
- $E_{pk}(E) \leftarrow$ SM$(E_{pk}(D), E_{pk}(n_f))$, send $E_{pk}(E)$ to CSP_1

5: CSP_1 computes:
- $E_{pk}(F) \leftarrow E_{pk}(C) * E_{pk}(B)^{N-1}$
- $E_{pk}(\delta_1) \leftarrow E_{pk}(F) * E_{pk}(E) \bmod N^2$
- $E_{pk}(\delta_{1r}) \leftarrow E_{pk}(\delta_1) * E_{pk}(r) \bmod N^2$
- Send $E_{pk}(\delta_{1r})$ to CSP_2, send r to end-user

6: CSP_2 :
- $\delta_{1r} \leftarrow$D$_{sk}(\delta_{1r})$, send δ_{1r} to end-user

7: End-user:
- δ_1 = δ_{1r} - r

5.2.1 Construction of $Proof_1$ (δ_1)

CSP_1 constructs $Proof_1$ and sends it to the end-user. Equation (1) shows δ_1 construction, and Algorithm 5 explains the steps of implementation. Although CSP_2 assists CSP_1, the workload assigned to CSP_2 is minimal. Besides, CSP_2 cannot learn any information because it computes all the operations on randomized data. We emphasize that CSP_1 only

knows $E_{pk}(\delta_1)$ and the final result δ_1 is only known to the end-user.

Algorithm 6 $Proof_2$ Preparation ($E_{pk}(\epsilon_{b_1})$, $E_{pk}(d_i)$, $E_{pk}(t_{i,j} - Q_j)) \rightarrow$ ($\mathrm{E}_{pk}(d_{i_r})$, $E_{pk}(t_i - Q)_r$)

1: **Initialize:** vector ϵ_c
2: CSP_1 :
- $E_{pk}(t_i - Q)$ = $\sum_{j=1}^{m}(E_{pk}(t_{i,j} - Q_j))$
- **for** $i = 1$ to n **do:**
  - $E_{pk}(d_{i_r}) \leftarrow E_{pk}(\epsilon_{b_{1i}})$ * $E_{pk}(d_i) \bmod N^2$
  - $E_{pk}(t_i - Q)_r \leftarrow E_{pk}(\epsilon_{c_i})$ * $E_{pk}(t_i - Q) \bmod N^2$
- Send $E_{pk}(d_{i_r})$ and $E_{pk}(t_i - Q)_r$ to CSP_2

5.2.2 Construction of $Proof_2$ (δ_2)

CSP_2 constructs δ_2 in two steps. Firstly, it computes the randomized proof δ_{2r}; then, it removes the randomness to obtain δ_2.

5.2.2.1 Randomized Proof δ_{2r} :

The reason behind adding this step is that computing δ_2 requires (1) the encrypted Euclidean Distance between the query Q and each fake tuple t_i, and (2) the encrypted differences between the query Q and each fake tuple t_i. Besides, CSP_1 cannot send (1) and (2) because it does not know the indices of fake tuples in DB. Also, since CSP_1 knows the tuple and the query in the encrypted form, it is not secure to reveal the indices of fake tuples to it, as it could cheat by doing the computations properly only for the fake tuples. Furthermore, CSP_2 holds sk; hence, CSP_1 must not send the Euclidean Distances and the differences for all tuples to CSP_2. Therefore, CSP_1 randomizes $E_{pk}(d_i)$ using ϵ_{b_1} and $E_{pk}(t_i - Q)$ using vector ϵ_c. We highlight that ϵ_{b_1} is generated by the data owner and ϵ_c is generated by CSP_1. The reason behind assigning ϵ_{b_1} to the data owner is that we will need to securely share the random numbers added to the distances of fake tuples with CSP_2 as

it requires them to solve equation (4). Therefore, the data owner generates ϵ_{b_1} and ϵ_{b_2}, and sends them to CSP_1 and CSP_2, respectively. Algorithm 6 explains $Proof_2$ Preparation.

After CSP_1 sends $E_{pk}(d_{i_r})$ and $E_{pk}(t_i - Q)_r$ to CSP_2, the latter selects the randomized distances and differences for only the fake tuples based on the indices in I_f. Then, it computes δ_{2r} using (3), which is similar to equation (2) but it outputs δ_{2r}. Algorithm 7 illustrates the computation of δ_{2r}. It is important to mention that CSP_2 decrypts all the randomized data before performing δ_{2r} computation.

$$\delta_{2r} = \sum_{i=1}^{n_f} \left[d_{i_r} + \epsilon_{a_{2_i}} \left(m * \epsilon_{a_{2_i}} + 2 \sum_{j=1}^{m} (t_{i,j} - Q_j)_r \right) \right] \tag{3}$$

Algorithm 7 δ_{2r} Computation (d_{ir}, ϵ_{a_2}, $(t_i - Q)_r$) $\rightarrow \delta_{2r}$

1: **Require:** CSP_2 has m, ϵ_{a_2}, d_{ir}, $(t_i - Q)_r$, and I_f
2: CSP_2 performs:
   - δ_{2r} = 0
3: **for** i = 1 to n_f **do:**
   - CSP_2 computes:
     - $a = (t_i - Q)_r + (t_i - Q)_r$
     - $b \leftarrow m * \epsilon_{a_{2i}}$
     - $c \leftarrow a + b$
     - $d = c * \epsilon_{a_{2i}}$
     - $\delta_{2_r} += d_{i_r} + d$

5.2.2.2 Remove randomness from δ_{2_r} :

CSP_1 receives the auxiliary information from the data owner during the Database Outsourcing Stage. It will use this information to locally perform a set of operations that are required to remove the randomness from all δ_{2_r}. We emphasize that CSP_1 executes the following steps only once for each database:

- Computes R_{t_i} based on ϵ_{a_1}, ϵ_{b_1}, and ϵ_c where $1 \leq i \leq n$, as per equation (4):

$$R_{t_i} = (\epsilon_{b_{1i}} + \epsilon_{a_{1i}}(2 * \epsilon_{c_i})) \tag{4}$$

  - Generates all possible combinations of fake tuples in pair of n_f, and stores them in vector ϵ_{d_1}. Each attribute in ϵ_{d_1} represents one combination of indices. For instance, suppose n = 3 and n_f = 2; then, the vector ϵ_{d_1} = $\langle$(1,2),(1,3),(2,3)$\rangle$.

- Constructs vector ϵ_{d_2} by adding the R_t values for each combination in ϵ_{d_1}. For instance, if a combination stored in $\epsilon_{d_{1i}}$ is $(1, 2)$, the corresponding index in $\epsilon_{d_{2i}}$ contains R_{t_1} + R_{t_2}.
- Sends ϵ_{d_1} and ϵ_{d_2} to CSP_2.

  As mentioned previously, CSP_1 performs the previous steps only once for each database, and sends vectors ϵ_{d_1} and ϵ_{d_2} to CSP_2. Similarly, CSP_2 receives these two vectors only once for each DB, and uses them every time to remove the randomness from the all δ_{2_r}. Specifically, CSP_2 selects the correct combination of fake tuples from ϵ_{d_1}; then, it finds the value in the corresponding index in ϵ_{d_2}. Finally, CSP_2 obtains δ_2 as shown in (5):

$$\delta_2 = \delta_{2r} - \epsilon_{d_{2i}} \tag{5}$$

5.3 Sort Verification

Similar to the Euclidean Distance Verification stage, each CSP is responsible for constructing and sending evidence to the end-user. Using the computed Euclidean distances, each CSP sorts the distances of fake tuples d_i, and constructs $H(\zeta_1)$ and $H(\zeta_2)$. Upon receiving the proofs, the end-user verifies if Z = $H(\zeta_1)$ - $H(\zeta_2)$ = 0; that is, $H(\zeta_1)$ = $H(\zeta_2)$. Each hash value, $H(\zeta_1)$ and $H(\zeta_2)$, represents the concatenation of θ and the hash values

of the sorted indices of fake tuples. The computation of the hash values of all indices is assigned to CSP_2 that generates a random salting value η. CSP_2 concatenates η to each index and computes the hash value over the result as shown in equation (6).

$$H(\xi_i) = \eta|\xi_i \tag{6}$$

We emphasize that η is generated for each query processing operation, used for computing the hash value of each index $H(\xi_i)$ and is only known to CSP_2. We also emphasize that the final hash value $H(\zeta)$ is computed using a different salting value θ that is known to both CSP_1 and CSP_2. $H(\zeta)$ represents the hash values of the sorted and concatenated indices of all fake tuples as per equation (7). Algorithm 8 explains the steps for Sort Verification.

5.3.1 Construction of $H(\zeta_1)$

As mentioned in Algorithm 3, CSP_1 obtains the sorted indices V' at step 2(d). CSP_1 will permute V' and send the permuted vector ρ to CSP_2. Upon receiving ρ, CSP_2 decrypts and computes the hash value for each index using equation (6), and sends ρ' to CSP_1. CSP_1 implements the inverse permutation on ρ' to obtain κ vector that represents the hash values of sorted indices.

Besides, CSP_1 receives another vector of hashes denoted by σ from CSP_2. Specifically, σ represents the hash values of unsorted indices of fake tuples. Next, CSP_1 sorts σ using the information stored in κ. Finally, CSP_1 computes H(ζ_1) and sends it to the end-user:

$$H(\zeta_1) = H[\theta|(H(\sigma_1)|H(\sigma_2)|H(\sigma_3)|.....|H(\sigma_{nf}))] \tag{7}$$

5.3.2 Construction of $H(\zeta_2)$

Based on the Euclidean Distance Verification step, CSP_1 sends $E_{pk}(d_{i_r})$ to CSP_2. Using the information I_f and ϵ_{b_2} shared by the data owner, CSP_2 decrypts $E_{pk}(d_{i_r})$ and $E_{pk}(\epsilon_{b_2})$ for only fake tuples. Then, it obtains the real distances of fake tuples d_i; that is, d_i = d_{i_r} - ϵ_{b_2}. Then, using the distance information d_i, CSP_2 sorts the indices of fake tuples, and computes the hash value for each index in I_f as per equation (6). Finally, CSP_2 concatenates the sorted hashes, adds the salting value θ, computes $H(\zeta_2)$, and sends it to the end-user.

We highlight that ϵ_{b_2} contains the random numbers added to only fake distances and it is impossible for CSP_2 to obtain the distances of real tuples since it does not have the random values.

The end-user concludes that the Euclidean distances were computed and sorted properly with a high probability only if $H(\zeta_1) = H(\zeta_2)$. After that, CSP_1 obtains the top k records as explained in Algorithm 3, randomizes them, and sends them to CSP_2 who is responsible for decryption. Upon receiving the random numbers from CSP_1 and the randomized top k records in the plain format from CSP_2, the end-user will remove the randomness and obtain the k-nearest tuples.

Algorithm 8 Sort Verification $(V') \rightarrow H(\zeta_1), H(\zeta_2)$

1: **Require:** CSP_2 knows η, d_{ir}, CSP_1 and CSP_2 know θ, CSP_1 knows V'

2: **for** $i = 1$ to n **do:**

(a). CSP_1:
- $\rho \leftarrow \pi(V')$; and sends ρ to CSP_2

(b). CSP_2 :
- Decypt ρ to obtain the permuted indices
- $H(\xi)_i = \eta \mid \xi_i$
- $\rho'_i \leftarrow H(\xi)_i$, and sends ρ' to CSP_1

(c). CSP_1 :
- $\kappa \leftarrow \pi^{-1}(\rho')$

3: **for** $i = 1$ to n_f **do:**

(a). CSP_2 :
- $H(\xi)_i = \eta \mid \xi_i$
- $\sigma_i \leftarrow H(\xi)_i$, and sends σ to CSP_1

(b). CSP_1 :
- HeapSort(σ) based on κ
- $\sigma' \leftarrow$ concatenated sorted σ
- $H(\zeta_1) = \theta \mid \sigma'$
- Send $H(\zeta_1)$ to end-user

(c). CSP_2:
- $d_{ir} \leftarrow D_{sk}(d_{ri})$
- $\epsilon_{b_2} \leftarrow D_{sk}(\epsilon_{b_2})$
- $d_i = d_{ir} - \epsilon_{b_2}$
- $d_{i_{sorted}} \leftarrow$ HeapSort(d_i)
- for each index in I_{sorted}, compute:
  - $H(\xi_i) = \eta | \xi_i$
- $H(\zeta_2) = H[\theta | (H(\xi_1) | H(\xi_2) | H(\xi_3) | ..... | H(\xi_{nf}))]$
- Send $H(\zeta_2)$ to the end-user

6 Conclusions

Data owners tend to outsource the computation and the verification of data to the cloud, which raises security and privacy concerns about the confidentiality and the integrity of data. In this thesis, we proposed a novel framework SVC-kNN that combines the computation and the verification of the k-nearest neighbor problem. The SVC-kNN involves four parties: Data Owner, End-User, CSP_1, and CSP_2. It also utilizes the homomorphic properties of the Paillier cryptosystem. Besides, it is secure under the semi-honest model. Additionally, this scheme maximizes the usage of the clouds' computational power due to assigning the encryption of data to CSP_1 and CSP_2. This will reduce the computation overhead on query issuers and data owners. The latter randomly split the databases and send each share to the corresponding CSP; then, the clouds consolidates the data and obtain $E_{pk}(DB)$. Moreover, we adopted the computation of k-nearest records from the literature; however, we slightly changed the existing solution and combined it with our verification protocol. Furthermore, our solution for the Secure Verification and Computation of kNN (SVC-kNN) consists of three stages: Proof Preparation, Euclidean Distance Verification, and Sort Verification. We emphasize that our verification scheme is probabilistic, and the SVC-kNN involves two clouds in verifying the top k records under the assumption that these clouds do not collude with each other. This framework verifies the Euclidean Distance at first where CSP_1 and CSP_2 construct proofs using different parameters, and send δ_1 and δ_2 to the end-user, respectively. Only if the Euclidean Distance verification were successful, the cloud proceeds toward the sort verification where each CSP sends $H(\zeta_1)$ and $H(\zeta_2)$, and the end-user verifies the equality of both values.

Due to time limitations, we could not implement SVC-kNN. Therefore, we left the experiments and the security analysis tasks for future work. Additionally, we will investigate how

to extend the proposed SVC-kNN to other security models.

7 References

[1] G. A. Lewis, "Cloud computing," *Computer*, vol. 50, no. 05, pp. 8–9, may 2017.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, p. 50–58, Apr. 2010. [Online]. Available: https://doi.org/10.1145/1721654.1721672

[3] S. Mehrotra, H. HacigÃ¼mÃ¼s, and B. Iyer, "Providing database as a service," in *Proceedings 18th International Conference on Data Engineering*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2002, p. 0029. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICDE.2002.994695

[4] J. Weis and J. Alves-Foss, "Securing database as a service: Issues and compromises," *IEEE Security Privacy*, vol. 9, no. 06, pp. 49–55, nov 2011.

[5] R. Canetti, B. Riva, and G. N. Rothblum, "Practical delegation of computation using multiple servers," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 445–454. [Online]. Available: https://doi.org/10.1145/2046707.2046759

[6] P. Williams, R. Sion, and B. Carbunar, "Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 139–148. [Online]. Available: https://doi.org/10.1145/1455770.1455790

[7] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," 1989.

[8] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *2014 IEEE 30th International Conference on Data Engineering*, 2014, pp. 664–675.

[9] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *2011 IEEE 27th International Conference on Data Engineering*, 2011, pp. 601–612.

[10] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA:

Association for Computing Machinery, 2009, p. 139–152. [Online]. Available: https://doi.org/10.1145/1559845.1559862

[11] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 2013, pp. 733–744.

[12] Y. Zhu, R. Xu, and T. Takagi, "Secure k-nn computation on encrypted cloud data without sharing key with query users," ser. Cloud Computing '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 55–60. [Online]. Available: https://doi.org/10.1145/2484402.2484415

[13] B. Dong, R. Liu, and H. W. Wang, "Trust-but-verify: Verifying result correctness of outsourced frequent itemset mining in data-mining-as-a-service paradigm," *IEEE Transactions on Services Computing*, vol. 9, no. 1, pp. 18–32, 2016.

[14] H. Rong, H. Wang, J. Liu, W. Wu, and M. Xian, "Efficient integrity verification of secure outsourced knn computation in cloud environments," in *2016 IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 236–243.

[15] W. K. Wong, D. W. Cheung, E. Hung, B. Kao, and N. Mamoulis, "An audit environment for outsourcing of frequent itemset mining," *Proc. VLDB Endow.*, vol. 2, no. 1, p. 1162–1173, Aug. 2009. [Online]. Available: https://doi.org/10.14778/1687627.1687758

[16] I. Molloy, N. Li, and T. Li, "On the (in)security and (im)practicality of outsourcing precise association rule mining," 12 2009, pp. 872–877.

[17] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, vol. 1592. Springer, 1999, pp. 223–238.

[18] B. K. Samanthula, H. Chun, and W. Jiang, "An efficient and probabilistic secure bit-decomposition," 05 2013, pp. 541–546.

Appendices

A Theorem 1

We provided δ_1 and δ_2 equations in Section 4, and mentioned that they are mathematically equivalent. We will prove by induction methodology that, $\delta_1 \equiv \delta_2$ for all $n \in Z$, where Z represents all possible integers.

$$\sum_{i=1}^{m} \left[\mu\left(\mu - 2Q\right) - 2\beta + n_f(Q^2)\right] = \sum_{i=1}^{n_f} \left[d_i + \epsilon_{a_{2i}} \left(m * \epsilon_{a_{2i}} + 2\sum_{j=1}^{m}\left(t_{i,j} - Q_j\right)\right)\right]$$

The random values of fake tuples are exactly the same in ϵ_{a_1} and ϵ_{a_2}. For simplicity reasons, we use r to denote these random values. Also, we assume that m = 1, meaning that each tuple has one attribute.

$\mu = \sum_{i=1}^{nf}\left(\epsilon_{a_{2i}} + t_i\right)$

$\beta = \sum_{i=1}^{n-1}\left(\epsilon_{a_{2i}} + t_i\right) * \left[\sum_{i=i+1}^{n}\left(\epsilon_{a_{2i}} + t_i\right)\right]$

(1) Base Case: Prove that the equation holds when n = 1,

Since n = 1, $\mu = r + t_i = t + r$, β = 0 because the number of combinations of μ is 0; then:

L.H.S $(\delta_1) = \mu(\mu - 2Q) - 2\beta + n_f(Q)^2$

$= (t + r)[(t + r) - 2Q] - 2(0) + (1)Q^2$

$= (t + r)^2 - 2Q(t + r) + Q^2$

$= t^2 + r^2 + 2tr - 2Qt - 2Qr + Q^2$

$= t^2 + r^2 + Q^2 + 2tr - 2Qt - 2Qr$

$= (Q - t - r)^2$

R.H.S $(\delta_2) = \sum_{i=1}^{n_f}[\sum_{j=1}^{m}(t_{i,j} - Q)^2 + r_i(mr_i + 2(t_{i,j} - Q))]$

= $(t-Q)^2 + r[r + 2(t-Q)]$

$= t^2 - 2tQ + Q^2 + r[r + 2t - 2Q]$

$= t^2 - 2tQ + Q^2 + r^2 + 2tr - 2Qr$

$= t^2 + Q^2 + r^2 + 2tr - 2tQ - 2Qr$

$= (Q - t - r)^2$

Therefore, both sides are equal when n = 1.

Induction Step: Assuming $n = k$ is true, that is:

$\sum_{i=1}^{k}(t_i + r_i)[\sum_{i=1}^{k}(t_i + r_i) - 2Q] - 2\left[\sum_{i=1}^{k-1}(t_i + r_i) * \left[\sum_{i=i+1}^{k}(t_i + r_i)\right]\right] + kQ^2$ = $\sum_{i=1}^{k}[(t_i - Q)^2 + r_i[r_i + 2(t_i - Q)]]$

Based on the assumption that $n = k$ is true, we will prove that $n = k + 1$ is also true, that is:

$\sum_{i=1}^{k+1}(t_i+r_i)[\sum_{i=1}^{k+1}(t_i+r_i)-2Q]-2\left[\sum_{i=1}^{k-1+1}(t_i + r_i) * \left[\sum_{i=i+1}^{k+1}(t_i + r_i)\right]\right]+(k+1)Q^2 = \sum_{i=1}^{k+1}[(t_i - Q)^2 + r_i[r_i + 2(t_i - Q)]]$

Proof:

L.H.S = $\sum_{i=1}^{k+1}(t_i + r_i)[\sum_{i=1}^{k+1}(t_i + r_i) - 2Q] - 2\left[\sum_{i=1}^{k-1+1}(t_i + r_i) * \left[\sum_{i=i+1}^{k+1}(t_i + r_i)\right]\right] + (k+1)Q^2$

$= \sum_{i=1}^{k}\mu_i[\sum_{i=1}^{k}\mu_i - 2Q] - 2\left[\sum_{i=1}^{k-1}\mu_i * \left[\sum_{i=i+1}^{k}\mu_i\right]\right] + kQ^2 + [\mu_{k+1}[2(\sum_{i=1}^{k}\mu_i) + \mu_{k+1} - 2Q] - 2[\mu_{k+1} * \sum_{i=1}^{k}\mu_i] + Q^2$

= $\sum_{i=1}^{k}(t_i + r_i)\left[\sum_{i=1}^{k}(t_i + r_i) - 2Q\right] - 2\left[\sum_{i=1}^{k-1}(t_i + r_i) * \left[\sum_{i=i+1}^{k}(t_i + r_i)\right]\right] + kQ^2 + \mu_{k+1}\left[2(\sum_{i=1}^{k}\mu_i) + \mu_{k+1} - 2Q\right] - 2\left[\mu_{k+1} * \left[\sum_{i=1}^{k}\mu_i\right]\right] + Q^2$

Substituting the L.H.S of the induction step with the R.H.S:

= $\sum_{i=1}^{k}[(t_i - Q)^2 + r_i[r_i + 2(t_i - Q)]] + 2[\mu_{k+1} * \sum_{i=1}^{k}\mu_i] + (\mu_{k+1})^2 - 2q\mu_{k+1} - 2[\mu_{k+1} *$

$\sum_{i=1}^{k} \mu_i] + Q^2]$

$=[[(t_1 - Q)^2 + r_1(r_1 + 2(t_1 - Q))] + ... + [(t_k - Q)^2 + r_k(r_k + 2(t_k - Q))]] + (\mu_{k+1})^2 - 2q\mu_{k+1} + Q^2$

$=[[(t_1 - Q)^2 + r_1(r_1 + 2(t_1 - Q))] + ... + [(t_k - Q)^2 + r_k(r_k + 2(t_k - Q))]] + (t_{k+1})^2 + (r_{k+1})^2 + 2 * t_{k+1} * r_{k+1} - 2Qt_{k+1} - 2Qr_{k+1} + Q^2$

$=[[(t_1 - Q)^2 + r_1(r_1 + 2(t_1 - Q))] + ... + [(t_k - Q)^2 + r_k(r_k + 2(t_k - Q))]] + (t_{k+1} - Q)^2 + (r_{k+1})^2 + 2t_{k+1}r_{k+1} - 2Qr_{k+1}$

$=[[(t_1 - Q)^2 + r_1(r_1 + 2(t_1 - Q))] + ... + [(t_k - Q)^2 + r_k(r_k + 2(t_k - Q))]] + (t_{k+1} - Q)^2 + r_{k+1}[r_{k+1} + 2t_{k+1} - 2Q]$

$=[[(t_1 - Q)^2 + r_1(r_1 + 2(t_1 - Q))] + ... + [(t_k - Q)^2 + r_k(r_k + 2(t_k - Q))]] + (t_{k+1} - Q)^2 + r_{k+1}[r_{k+1} + 2(t_{k+1} - Q)]$

$= \sum_{i=1}^{k+1}[(t_i - Q)^2 + r_i[r_i + 2(t_i - Q)]] =$ **R.H.S**