Privacy-Preserving Weighted Manhattan Distance-based Similarity Query

by

Rhoda Tani Mairabo

Bachelor of Engineering, Ahmadu Bello University Zaria, 2003
Bachelor of Science, Belarusian State University of Informatics and Radioelectronics, 2018

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor:	Rongxing Lu, Phd, Faculty of Computer Science
Examining Board:	Saqib Hakak, Phd, Faculty of Computer Science, Chair
	Sajjad Dadkhah, Phd, Faculty of Computer Science
	Zhen Lei, Phd, Department of Civil Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2023

Abstract

Computing over outsourced, encrypted data in an efficient, secure and privacy-preserving way continues to be a challenge as the quantity, complexity and applications of the data continue to grow. Both data owners and data users require solutions that address the continued evolution of security and privacy needs in the presence of determined malicious actors and threats. The basic framework for similarity queries has remained relatively unchanged, but we continue to design more efficient, secure and privacy-preserving schemes that combine different encryption techniques and data structures to better address the needs of the users and changing landscape of data complexity and its attendant threats. We propose and design a similarity query scheme that uses a weighted Manhattan distance-based metric, a symmetric homomorphic encryption (SHE) technique, a kd-tree to index our data and a 2-cloud server model. Security analysis shows that our proposed scheme can achieve the desirable privacy requirements.

Dedication

For Dorcas Mairabo, who always inspired and encouraged me to be better.

Acknowledgements

I would like to express my gratitude for the guidance, patience and encouragement of my supervisor, Dr. Rongxing Lu. Dr. Yunguo Guan, whose assistance was invaluable and consistently pushed me towards excellence in this work. I would also like to thank Dr. Pulei Xiong, Dr. Yandong Zheng, and Guanjuo Tang for their support, advice and contributions. Many thanks to my family, Dad, Es, Ruthie, Rabi, Glor, for your love and support. So many others too numerous to mention gave time, advice, encouragement and assistance in various ways on this journey, thank you. Most importantly, I thank God for His mercies, which have never failed.

Table of Contents

List of Tables

List of Figures

List of Symbols, Nomenclature or Abbreviations

AES	Advanced Encryption Standard
DO	Data Owner
CS	Cloud Server
CSP	Cloud Service Provider
CKA	Chosen Keyword Attack
CPA	Chosen Plaintext Attack
DO	Data Owner
HE	Homomorphic Encryption
IND-CCA	Indistinguishability under Adaptive Chosen Ciphertext Attack
IND-CPA	Indistiguishability under Chosen Plaintext Attack
NIST	National Institute of Standards and Technology
PEKS	Public Encryption with Keyword Search
PK	Public Key
SE	Searchable Encryption
SHE	Symmetric Homomorphic Encryption
SK	Secret Key
SSE	Symmetric Searchable Encryption

Chapter 1

Introduction

Data in today's world has become an increasingly valuable commodity. Over 300 million terabytes of data are produced daily from mobile devices, industrial plants, IoT devices, vehicles, government, medical facilities, military installations, satellites, sensors and so much more. [64],[13] As the sources, quantity and rate of output of data has continued to increase so have its features, attributes, dimensions and formats. This has led to the need for continuous improvements in data collection, processing, storage and security. This has opened up opportunities and challenges for data owners, data brokers and data users. Finding cost-effective, efficient and secure methods of transporting, storing, accessing, and encrypting the data remains an area of constant research and improvement. The introduction of cloud computing has provided data access, storage, collaboration and other variables at a cost, without having to handle the costs of expensive hardware and software maintenance and upgrades. Primary data storage and retrieval have advanced to more complex processes that improve security, search efficiency, privacy, and computational ability. Advances in cryptography have led to more secure encryption techniques that have led to computing on encrypted data. Various theories, schemes and applications have been proposed and developed to continuously address the challenges and threats that

have evolved along with the changing data landscape and its associated operations. Choosing appropriate encryption, index, storage, and search methods to efficiently and securely retrieve results can become computationally expensive, vulnerable to security breaches and loss of data privacy, confidentiality and integrity. Our work looks at the use of homomorphic encryption for efficient, privacy-preserving and secure computation on outsourced data, to retrieve encrypted data records that meet specified similarity range query criteria.

1.1 Searchable Encryption

Searching encrypted data was pioneered by the work of Song et al., [59] who laid the foundations of what has evolved into computing on encrypted data with advances in homomorphic encryption as proposed by Gentry [25]. Handa et al. [34] describe searchable encryption as a multidisciplinary field combining *Information Retrieval* for searchable index generation, *Algorithms* for devising an efficient search scheme and *Cryptography* to maintain security and privacy requirements. Searchable encryption finds applications in encrypted images, encrypted audit logs, encrypted medical data, encrypted genomic data, encrypted smart grid data, IoT collected data, secure biometric authentication, and secure databases. Searchable encryption (SE) is a system that allows a group of data users or owners to store, share, and retrieve remotely stored data [17]. The type of searchable encryption scheme used for a specified application can depend on any number of variables which include frequency of queries, size and type of data, sensitivity of data, number of users etc.
The model of a searchable encryption scheme generally consists of three basic entities, the data owner, the cloud server and the user [49].

- *The data owner*, might be an individual or organization who owns the data and is responsible for building the encrypted searchable index and outsourcing

the encrypted data to the cloud server.

- *The data user* is the entity that wants access to the outsourced data and generates a query which is sent to the cloud server for processing. The query results are then returned to the data user.

- *The cloud server* receives the encrypted data from the data owner and stores the encrypted data. It performs search and computation operations on the encrypted data and returns the results to the end user.

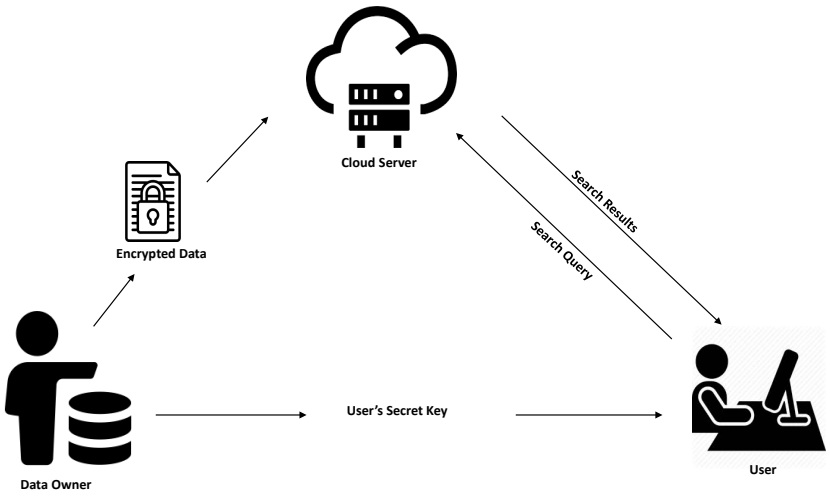

Figure 1.1: Searchable Encryption Model

Searchable encryption schemes usually apply four basic algorithms as described below [34].

- *KeyGen*: This is the system initialization algorithm usually run by the data owner. Here, the secret keys, public and private parameters are generated, based on the type of encryption method (symmetric or asymmetric) used.

- *BuildIndex*: This is the algorithm used to generate the encrypted searchable index that corresponds to the data to be searched. The data owner takes a

secret key K and a set of documents D as input and outputs an encrypted keyword index $\mathcal{I}$.

- *Trapdoor*: This is the algorithm used by the end users to generate a search query. In some cases, the data owner may generate the query on behalf of the users. A secret key K and query keyword w are taken as inputs to output a trapdoor T_w for the query keyword.

- *Search*: this is the algorithm used by the cloud server after it receives the encrypted query and compares it to the index entries and generate a list of matching documents.

1.1.1 Security Models for Searchable Encryption

Security for SE was first introduced by Goh et al. [26], who defined security for indexes as semantic security (indistinguishability) against adaptive chosen keyword attacks *IND1-CKA*. This was where an adversary, given 2 encrypted documents of equal size, cannot deduce which document's index is encoded, but did not require trapdoors to be secure. Goh et al. [26], later introduced *IND2-CKA* which protected document size i.e. documents of unequal size, but still did not provide security for trapdoors. Curmotola et al. [17] reviewed and improved on the work done by Goh et al. on searchable encryption and addressed security for trapdoors. They introduced two adversarial models, *IND2-CKA1*, which is non-adaptive and *IND-CKA2*, which is adaptive indistinguishability against chosen keyword attacks. Other security models have been proposed for SE schemes outlined below [34].

- **IND-CPA**(Indistinguishability against Chosen plaintext attacks): It is desired that it should not be possible for an adversary who knows the encryption algorithm, the cipher text corresponding to a message m, and the length plain-text attacks of the message.

- **IND-CKA**(Indistinguishability against Chosen Keyword attacks): it is desired that the index provides security with respect to the contents of the documents, ie, no information about the document is revealed to the chosen-keyword adversary from the encrypted index.

- **IND2-CKA**: It is desired that the index for two unequal-sized documents should also be indistinguishable.

- **Non-adaptive IND-CKA1**: It is assumed that the queries of the adversaries are independent of the previous queries, ie, the adversary issues all the queries at once and obtains the desired encrypted documents.

- **Adaptive IND-CKA2**: It is a variant of IND-CKA1 where the adversary can utilize the information of previous queries (i.e, results and trapdoors) to generate the new queries and to gain insights into the encrypted documents and indexes (without the assumption of all queries being executed at once).

- **Known cipher-text model**: Here, the adversary (or cloud server) has access to the encrypted document collection and the encrypted index. The adversary tries to gain insights into the collection using the available information only.

- **Known background Model**: Here, the adversary (or cloud server) has access to the encrypted document collection and encrypted index and knowledge about the document collection(in the form of keyword frequency, etc). This background knowledge, together with the encrypted document collection and index, can be used to launch statistical attacks.

- **Internal attacks**: Initiated by malicious insiders, such as cloud users, cloud providers,and internal organization users, and involve alteration of the data.The capability to withstand such an internal attack is used to measure the security of the scheme.

- **External attacks**: Initiated by outside users like unauthorized adversaries and involve unauthorized access to the data.The capability to withstand such an external attack is used to measure the security of the scheme.

1.2 Similarity Queries

Similarity queries are usually used to find data records that are similar or related to a given query record. Equal or exact matches of data are only sometimes effective or efficient, in cases where we are dealing with noisy data, trying to detect small differences between data objects or comparing complex objects [6]. Similarity queries find application in recommendation systems [39], computer vision [38], information retrieval [4], natural language processing [23], bioinformatics [48], healthcare databases [68], fraud detection [57] and more. There are two major types of similarity queries, the range query and the nearest neighbor query.

1.2.1 Range Query

The range query searches a given set of data to find all the data objects that lie within a given distance from a specified data object [63]. Given a dataset X, a query value q and a query threshold r, the range query finds and retrieves data objects from X whose distances to q are bounded by r, and can be defined as:

$$R(q, r) = \{x \in X | d(q, x) \leq r\}$$

Fig. 1.2 gives an example of using the two-dimensional Euclidean distance-based range query described above, where the query aims to find all objects constrained by a distance r from the query point q.

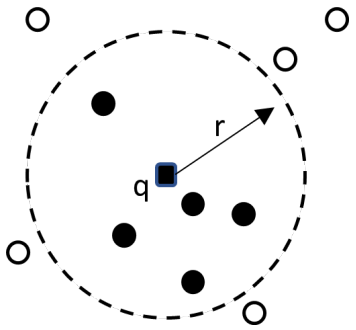

Figure 1.2: Similarity Range Query Example

1.2.2 Nearest Neighbor Query

The nearest neighbor query searches a data-set and finds the data objects that are closest to a specified query object. Given a query object q, a dataset X and a distance function $d(\cdot, \cdot)$, the query finds and retrieves data objects that are closest to the specified query object [65].

$$x \in X, \forall y \in X, d(q, x) \leq d(q, y)$$

This query can be extended to find k data objects in a dataset closest to a specified query object and is called the $k-$nearest neighbor query. Figure 1.3 is an example of a two-dimensional Euclidean distance-based k-nearest neighbor query, that indicates all objects closest to the query object q, where $k = 5$ [65].

$$\text{Given, } k\text{−NN}(q, k) = A, \text{ where } A \subseteq X, |A| = k, \text{ such that}$$
$$\forall x \in A, y \in X - A, d(q, x) \leq d(q, y)$$

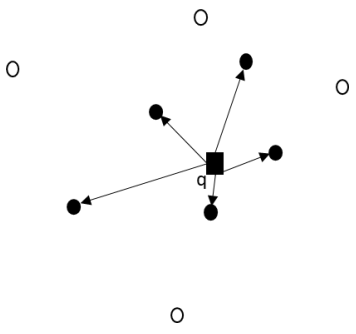

Figure 1.3: k-NN Query Example

1.2.3 Distance-based Similarity Queries

A distance-based similarity query problem is made up of a dataset characterized by some features, a query that specifies the value to be searched, and the similarity metric to measure the relevance between the query and the dataset. Distance can be defined as a measure of the absolute value of the difference between two points and this value can be used to gauge the similarity of data objects. The distance function gives us a dissimilarity criterion to compare database objects. For a small distance between a pair of data objects in a given domain, the pair of data objects are considered similar while a large distance between them shows dissimilarity. Therefore, the distance-based similarity query retrieves data records whose similarity when measured using the distance metric is greater than a predetermined threshold.

Definition 1.2.1. *A distance metric d, is defined over objects in a set X as a function $d : X \times X \to [0, \infty)$, such that for each $x, y, z \in X$ the following properties are satisfied [45].*

$$d(x, y) \geq 0 \text{ (non-negativity)}$$
$$d(x, y) = 0 \Leftrightarrow x = y \text{ (identity)}$$
$$d(x, y) = d(y, x) \text{ (symmetry)}$$
$$d(x, y) \leq d(x, y) + d(y, z) \text{ (triangle inequality)}$$

Definition 1.2.2. *A similarity function s over objects in a set X is a function: $s : X \times X \to [0, u]$, where u is an upper bound (i.e., the maximum similarity value, usually $u = 1$), and where for each $x, y \in X$ the following properties are satisfied [45].*

$$d(x, y) \geq 0 \text{ (non-negativity)}$$
$$d(x, y) \leq u \text{ (boundedness)}$$
$$s(x, y) = u \Leftrightarrow x = y \text{ (identity)}$$
$$s(x, y) = s(y, x) \text{ (symmetry)}$$

1.2.4 Distance Functions for Similarity Queries

The choice of appropriate distance function for use in a similarity query tends to improve the results of distance-based similarity query algorithms. When similarity is modeled with a distance function that satisfies the triangle inequality, the set of objects is called a metric space.

Definition 1.2.3. *If the distance d is a metric, and it satisfies:* $\forall x, y, z \in \mathbb{X}, d(x, y) \leq d(x, z) + d(z, y)$ *traingle inequality, then the pair* $(\mathbb{X}, d)$ *is called a metric space [15].*

This section will discuss some standard distance functions used to measure similarity in metric space, specifically the different instantiations of the Minkowski distance.

- *Minkowski Distance*, is a metric in normed vector space, i.e. a space where distances can be represented as vectors that have a length. It is a distance between two points x and y, with an order p (where p is an integer), such that: $d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$ for $p > 0$.

- *Euclidean distance* is a straight line distance between two points x and y, for $p = 2$.

$$d_E(x, y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

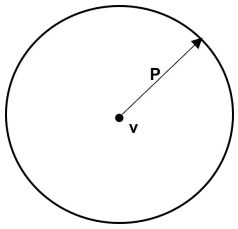

Figure 1.4: Euclidean Distance in 2-dimensions

- *Manhattan distance*, also known as the city block distance, the taxicab distance, or the rectilinear distance is the distance between two points measured along axes at right angles, it follows a grid-like path. For $p = 1$, the Manhattan distance is given as:

$$d_M = \sum_{i=1}^{n} |x_i - y_i|$$

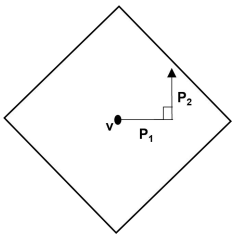

Figure 1.5: Manhattan Distance in 2-dimensions

- The *Chebyshev distance* also known as the maximum value distance, the chessboard distance or the L-infinity distance, is the absolute magnitude of the differences between coordinates of a pair of objects, for $p = \infty$.

$$d_C = max|x_i - y_i|$$

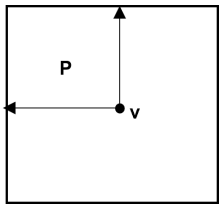

Figure 1.6: Chebyshev Distance in 2-dimensions

Table 1.1: Common Instantiations of Minkowski Distance

Name	Power Parameter(p)	Formula
Euclidean distance	$p = 2$	$d(x,y) = (\sum_{i=1}^{n} \|x_i - y_i\|^p)^{\frac{1}{p}}$
Manhattan distance	$p = 1$	$d_M = \sum_{i=1}^{n} \|x_i - y_i\|$
Chebyshev distance	$p = \infty$	$d_C = max\|x_i - y_i\|$

1.3 Data Structures for Similarity Queries

Data structures are a crucial element in similarity queries, as the choice of an appropriate data structure can significantly impact the scalability, efficiency and ef-

fectiveness of the similarity query. Techniques and data structures are required to organize feature vectors and manage the search process so that objects relevant to the query can be located quickly. A large number of index structures have been developed for efficient query processing in some multidimensional space. In general, the index structures can be classified into two main groups: data organizing structures such as R-trees and space organizing structures such as kd-trees. [11] Below we list commonly used data structures for indexing multidimensional data.

- R-trees: Guttman [33] proposed the R-tree, a dynamic index multidimensional spatial data. The R-tree organizes data into a hierarchical structure consisting of nodes. It abstracts data objects into minimum bounding rectangles (MBRs). Each node in the tree represents a bounding box that encompasses a group of spatial objects or other nodes.

- Kd-trees: Also known as a k-dimensional tree, it is a type of binary tree, proposed by Bentley [10]. They are space-partitioning trees that recursively divide the data space into regions along the axes of the dimensions. They are effective for range searches and nearest neighbor queries in multidimensional spaces.

- M-trees: Ciaccia et al. [16], introduced a paged metric tree. It is a balanced tree, able to deal with dynamic data files, and as such it does not require periodical reorganizations. M-tree can index objects using features compared by distance functions which either do not fit into a vector space or do not use an Lp metric.

- LSH: Local sensitivity hashing [36],[41], The basic method uses a family of locality-sensitive hash func- tions to hash nearby objects in the high-dimensional space into the same bucket. Two similar data would be mapped into one bucket with a very high probability, and two dissimilar data would be mapped into

one bucket with a very low probability.

High-dimensional data is subject to the "curse of dimensionality" [11],[8], [37], where as the number of dimensions increase, the cost of an algorithm grows exponentially with dimension and efficiency reduces.

1.4 Cloud Computing

The authors, P.Mell and T. Grance in a NIST publication [43], define cloud computing as *"a model for enabling ubiquitous, convenient,on-demand access to a shared pool of configurable resources(e.g., networks,servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*. Individual and organizations are able to take advantage of cloud services to outsource data, storage and infrastructure needs with the added benefits of lower costs, scalability, flexibility and anywhere-anytime access. NIST recommends five essential characteristics for the cloud, On-demand self-service, Broad network access, Resource pooling, Rapid elasticity and Measured service.

1.4.1 Cloud Models

We outline the general classifications for cloud models that defines them according to the services they provide.

- *Software as a Service* (SaaS): a service where users access software and other programs in the cloud and eliminate the need for on-site data storage, applications and application support.

- *Platform as a Service* (PaaS): a service where users develop cloud applications and services without having to purchase their own equipment. It supports a full software life cycle and delivers the developed applications to clients over the internet.

- *Infrastructure as a Service* (IaaS): a service where users have access to servers, data centers, network equipment in a virtualized environment.

- *Container as a Service* (CaaS): this service makes cloud applications independent of the PaaS environment and is based on the concept of container virtualization [3].

1.4.2 Cloud Deployment Models

NIST also identifies four deployment models for the cloud: Private cloud, Community cloud, Public cloud, and Hybrid cloud, they may be owned, managed and operated by a single organization, a third party or a combination of them.

- *Private Cloud*: this is usually used or owned by a single organization,with infrastructure on or off premises of the cloud service provider (CSP). It is usually not accessible to the public. It is considered the most secure cloud model as all its data processes are controlled and managed by the organization [18], [53].

- *Community Cloud*: for use by a community of customers from organizations with shared concerns, projects or share infrastructure demands for software and hardware to help reduce running costs of IT services. It may be situated on or off the premises of the CSP.

- *Public Cloud*: for open use by the general public, usually exists on the premises of CSP. This model offers applications, data storage and different services to the users. It is usually a pay-as-you-go type model that offers availability, reliability, freedom of self-service with a flexible and elastic environment.

- *Hybrid Cloud*: a combination of two or more distinct cloud architectures, usually private and public cloud, that work together to enable data and appli-

cation portability, but remain unique entities. It is usually shared between organizations that have similar interests and requirements. Consideration for standardization and interoperability of clouds will need to be considered in this model.

1.5 Related Works on Privacy-Preserving Similarity Queries

The need for ever-increasing efficient and privacy-preserving queries on encrypted data has produced a vast body of research. These queries have been applied to encrypted time series data [31], access control in healthcare [68], exact set similarity search [67], encrypted spatial data [60]. Query efficiency has been approached from the use of encryption technique, the type of data indexing applied,data size and related attributes, to the type of distance metric used to achieve more accurate query results and reduce the cost of computation. Guo et al. [32] explored a weighted similarity search scheme based on locality-sensitive hashing (LSH) and searchable symmetric encryption (SSE), with a view to extend their research application for multiple data owners. Peng et al. [47] designed a secure kd-tree for efficient and secure range queries over encrypted multi-dimensional data using comparable encryption (CE) and AES.Their scheme was designed to be more applicable to location-based services, geographical data-sets and resource constrained devices. Shi et al. [56] designed an encryption scheme, multi-dimensional range query over encrypted data (MRQED) to specifically address the privacy of shared network audit logs,with further applicability to financial audit logs and medical privacy. [66] used Euclidean distance for similarity range queries to pre-compute distances in order to reduce the cost of distance computation for the queries. Zheng et al. [40] designed a privacy-preserving multi-dimensional range query (PRQ) scheme using an R-tree to

address the challenge of single-dimensional range query leaks. In their work Zheng et al. [69], [70], used the Euclidean distance as a metric for their similarity range calculations, proposing to achieve greater query result accuracy in a one-server system model.

1.6 Problem Statement

To efficiently perform a similarity query on an outsourced, encrypted database while preserving the privacy of the data, query process and query results continues to be an ongoing research challenge.

- Data Privacy: In our proposed scenario, the data owner encrypts data before outsourcing it to the cloud server, ensuring the data's privacy and security. However, the data owner no longer has full control of the data and the computation operations that occur on it. The cloud servers are considered honest-but-curious, meaning they are assumed to follow assigned roles and protocols without deviation. However, the possibility of a curious cloud server sensing, linking or leaking information from the stored data and the queries remains a possibility and concern for the data owner.

- Query privacy and efficiency: The query user's query is encrypted before being sent to the cloud for processing and eventual return of results. The cloud servers, still being honest-but-curious may also be curious about the contents of the query, the results, and the query user who generated the specific query. The possibility of inferring preferences, features and search patterns of data concerns the query user. The query user wants to get the results through an optimized process that ensures efficiency and privacy.

Most distance-based similarity queries use Euclidean distance to calculate similarity [62],[68]. The Manhattan distance has been shown to be more effective in

high-dimension applications compared to other distance metrics[2]. The use of the weighted Manhattan Distance has been applied by Anyaiwe et al. [5] for analyzing data on Alzheimer's disease, for the analysis of metabolomic data in botanical research [19] and nearest neighbor search in sublinear time [35]. The weighting of distances allows the query user to emphasize the importance of different dimensions or data features, where such features are more relevant to the similarity query. Our goal is to preserve the privacy of our data and ensure that our queries are processes are also privacy-preserving and efficient. Based on our research, there has been no work that has specifically combined a kd-tree, a two-server model, a Somewhat Homomorphic Encryption (SHE) technique and a weighted Manhattan distance to improve the efficiency and privacy of a similarity query.

1.7 Main Contributions

This work proposes a privacy-preserving weighted Manhattan distance-based similarity query scheme (*WMDSQ*) whose privacy and security is based on Symmetric Homomorphic Encryption (SHE). Our proposed scheme aims to show that the user's queries are efficient and privacy-preserving. Our main contributions are as follows.

- We index and build a multi-dimensional dataset using a kd-tree for efficient searching of data. We encrypt this dataset using the SHE technique and outsource it to cloud server one for storage and further computation and processing of user-generated queries as proposed by our scheme.
- We design a weighted Manhattan distance-based similarity query (WMDSQ) scheme based on the SHE technique and a set of privacy-preserving protocols that meet our efficiency and privacy preservation goals.
- We prove our scheme's security and simulate its performance. We then analyze the results and present them.

1.8 Thesis Organization

This is organized as follows: Chapter 2 is a background of the basics of cryptography as applies to its application to the design of our scheme. Chapter 3 looks at our proposed design goals, system and security models. In chapter 4, we discuss and describe the preliminaries of our scheme. Chapter 5 details our proposed scheme and chapter 6 looks at our security analysis. Chapter 7 is our experimental settings, analysis and results. Chapter 8 presents our conclusion and possibilities for future work.

Chapter 2

Background

This chapter gives an overview of cryptography, its definitions and concepts.

2.1 Cryptography

According to NIST (National Institute of Standards and Technology), cryptography is the discipline that embodies the principles, means and methods for the transformation of data in order to hide their semantic content, prevent their unauthorized use, or prevent their undetected modification [21]. Rivest [50] defines cryptography as communication in the presence of adversaries, that provides methods that enable a communicating party to develop trust that his communications have the desired properties like privacy, authentication and more. Menezes et al.[44] define cryptography as the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication. Information and communication security is based on cryptographic algorithms of encryption and decryption, which rely on the secrecy of the keys to achieve the goals of modern cryptographic systems. Some applications of cryptography include data storage, secure end-to-end communication,storing of passwords, digital currency and much more.

The fundamental goals of cryptography that form the security framework are defined as follows.

- Confidentiality: only authorized entities can access the message/data.
- Data Integrity: ensuring that the message/data has not been modified, substituted or manipulated in any way during transit to its destination.
- Authentication: this identifies the message/data and the communicating entity as authentic; its identity and origins are ascertained.
- Non-repudiation: the sender/receiver of a message should not be able to deny previous commitments or actions.
- Access Control: this ensures that only users with the right permissions have access to protected information or systems.

2.2 Encryption

Encryption is the principal application of cryptography and a technique for privacy preservation and confidentiality. An unencrypted message, referred to as plaintext M is encrypted to become a ciphertext C. Decryption turns the ciphertext back into plaintext. [61] Tilborg et al. define encryption (also called enciphering) as a mapping of plaintext to ciphertext based on some chosen key.

The encryption function E encodes a message M as a string of bits $M \in \{0,1\}^l$ with a key $k \in \{0,1\}^n$ into a ciphertext $C \in \{0,1\}^L$, such that $C = E_k(M)$.The decryption function D reverses the encryption to reveal the original message $M = D_k(E_k(M))$. There are two main types of encryption, symmetric (secret key) and asymmetric (public key) encryption.

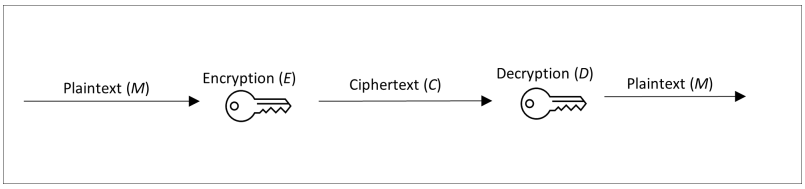

Figure 2.1: Encryption Process

2.2.1 Symmetric Encryption

Symmetric encryption also known as secret-key encryption is where the same key is used to encrypt and decrypt the message. This secret key must be shared ahead of time by the communicating parties. Symmetric encryption works by using a stream or block cipher for encryption and decryption. A stream cipher converts the plaintext into ciphertext one byte at a time, while the block cipher converts blocks of plaintext with predetermined key lengths which could be 128, 192 0r 256 bits. Types of symmetric encryption include Data Encryption Standard (DES), Advanced Encryption Standard (AES), Triple Data Encryption Standard (Triple DES), International Data Encryption Algorithm (IDEA) and the TLS/SSL protocol.

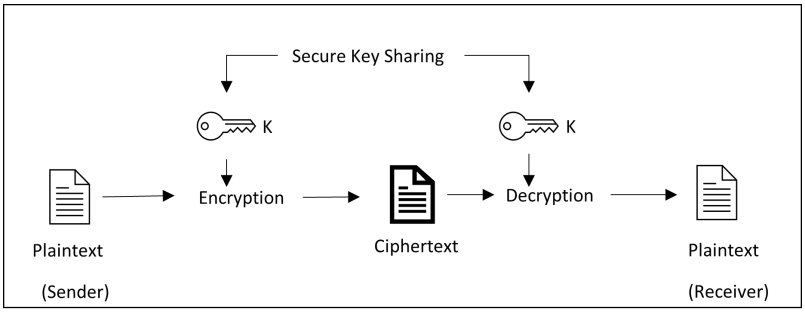

Figure 2.2: Symmetric Encryption

Symmetric encryption tends to be faster to execute due to its simplicity and short key lengths and are generally secure. However, its disadvantage lies in the use of a single, secret cryptographic key for encryption and decryption, which is susceptible to access by malicious actors.

2.2.2 Asymmetric Encryption

Asymmetric encryption also known as public key encryption is where a public key is used to encrypt the message and a private key is used to decrypt the message. The communicating parties give their unique public keys to each other to encrypt messages to be sent. These messages can only be decrypted by an individual's private key which is only known to the intended receiver of the message. Asymmetric encryption can also be used for digital signature authentication, Examples of symmetric encryption include Rivest-Shamir-Adleman (RSA), Elliptical Curve Cryptography (ECC), Digital Signature Standard (DSS) which incorporates the Digital Signature Algorithm (DSA), the Diffie-Hellman exchange method.

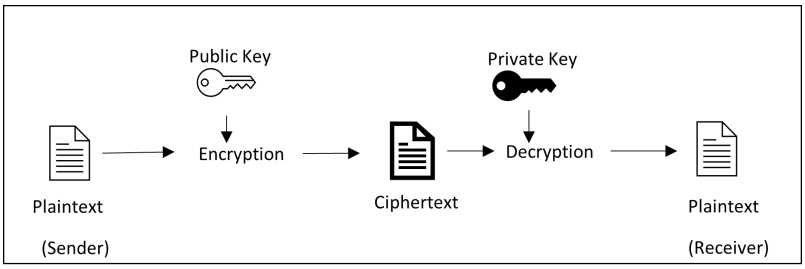

Figure 2.3: Asymmetric Encryption

One of the key advantages of asymmetric encryption is that there is no need for private key exchange and distribution, thereby ensuring the security and integrity of the encrypted messages. The ability to digitally sign and verify messages is also an advantage. The major drawback for asymmetric encryption is speed, due to longer key lengths and more complex algorithms.

2.3 Security of an Encryption Scheme

Shannon proved that perfect security against an arbitrarily capable adversary is impossible unless the number of possible keys is as large as the number of possible messages encrypted with a single key [54], this makes true perfect secrecy practically

impossible. In their paper [7], two goals of encryption scheme security, indistinguishability of encryption and non-malleability were considered. Indistinguishability [27] means that an adversary, given a single encryption of a known message chosen from the set $\{m_0, m_1\}$, should not be able to determine which message the ciphertext represents with probability significantly greater than $\frac{1}{2}$. A formal definition of indistinguishability is given below.

Definition 2.3.1. *A cryptosystem is said to be indistinguishable under chosen plaintext attack, or IND-CPA, if a probabilistic polynomially bounded adversary cannot win the following game with probability greater than* $\{\frac{1}{2}+\mathcal{V}\epsilon\}$*, where* $\mathcal{V}$ *is a pre-defined security parameter and* ϵ *is a negligible function:*

In the scenario:

1. The challenger creates an instance of the cryptosystem using security parameter and sends the public key to the adversary.

2. The adversary may perform a polynomially-bounded number of encryptions or other calculations.

3. The adversary chooses two different messages $m_0, m_1 \in P$, and sends them to the challenger.

4. The challenger randomly chooses a bit $b \in \{0, 1\}$, and sends the ciphertext $eK(m_b)$ to the adversary.

5. The adversary may perform a polynomially-bounded number of encryptions or other calculations, then responds with either 0 or 1.

6. The adversary wins the game if the bit-value chosen by the adversary is the same value chosen by the challenger.

In a situations in where an adversary can modify a ciphertext such that it affects the plaintext in a deterministic way, these cryptosystems are said to be malleable, while cryptosystems that resist this behavior are said to be non-malleable.

Definition 2.3.2. *A cryptosystem is said to be malleable if, given a ciphertext c representing plaintext m, it is feasible to compute functions f and g, where f is not the identity function, such that $f(c)$ decrypts to $g(m)$. If it is not feasible to calculate any such f and g, the cryptosystem is said to be non-malleable.*

Dolev et al's. [20] work on non-malleability formalized the adversary's inability given an ciphertext to output a different ciphertext such that that the plaintexts of both ciphertexts are meaningfully related. Goldwasser and Micali [29] proposed the widely used definition of semantic security and indistinguishability of encryption where an adversary is unable to learn any information about the plaintext from the ciphertext. Semantic security is defined in the Encyclopedia of cryptography and security [52] as the security of an encryption scheme, it also refers to indistinguishability of encryption. Here, the adversary chooses one of two plaintexts m_1 or m_2, and receives the encryption of either one. An encryption scheme is said to be semantically secure if the adversary cannot guess with a probability greater than $\frac{1}{2}$ if the given ciphertext is an encryption of m_1 or m_2.

Indistinguishability under chosen plaintext attack is usually the weakest form of security expected from a cryptosystem. The strongest form of indistinguishability assumes that the adversary retains access to the decryption oracle after the challenge ciphertext is received, but that the decryption oracle will not respond if the adversary queries the challenge ciphertext. [24] Basically, the adaptive chosen ciphertext indistinguishability IND-CCA2 is the strongest requirement for an encryption as it implies non-malleability. Homomorphic cryptosystems cannot be IND-CCA2 secure. This follows from the fact that any homomorphic cryptosystem is malleable. They are however able to achieve IND-CCA1 security.

2.3.1 Security Attack Models for Encryption Schemes

The security models for symmetric and asymmetric encryption are mainly based on the scenario where an adversary attempts to compromise the access and security of the ciphertext by recovering the plaintext from ciphertext or deducing the decryption key. Outlined below are possible security attack models for encryption schemes.

- A Ciphertext-Only Attack: an attack where the adversary tries to find the plaintext or decryption key from the ciphertext.
- A Known-Plaintext Attack: an attack where the adversary has some plaintext with the corresponding ciphertext.
- Chosen-Plaintext Attack (CPA): an attack where the adversary chooses the plaintext and is given the ciphertext and is required to recover the plaintext of a given ciphertext.
- Adaptive Chosen-Plaintext Attack: an attack where the choice of plaintext may depend on the ciphertext received from previous requests.
- Chosen-Ciphertext Attack (CCA or CCA1): an attack where the adversary selects the ciphertext and is given the corresponding plaintext and is then required to find the plaintext when given a different ciphertext.
- Adaptive Chosen-Ciphertext Attack (CCA2): an attack where the choice of ciphertext may depend on the plaintext received from previous requests. [44]

2.4 Homomorphic Encryption

Homomorphic encryption (HE) is a type of encryption where users can perform computations on encrypted data without decrypting it, where the decrypted result has to be equal to the intended computed value if it was done on the plaintext. Acar

et. al. [1] define homomorphic encryption as a kind of encryption scheme which allows a third party (e.g., cloud, service provider) to perform certain computable functions on the encrypted data while preserving the features of the function and format of the encrypted data. Homomorphism in cryptography was introduced by Rivest, Adleman and Dertouzous [50] as they attempted to address computing on encrypted data. There were subsequent proposals and schemes designed by other researchers to address the limitations of Rivest et al.'s work, culminating in Gentry's 2009 HE proposal [25]. An encryption scheme [1] is called homomorphic over an operation " $*$ " if it supports the following equation:

$$E(m_1) * E(m_2) = E(m_1 * m_2), \forall m_1, m_2 \in M$$

Where E is the encryption algorithm and M is the set of all possible messages.
A HE scheme usually is made up of four operations:

- KeyGen: this generates a secret key for symmetric encryption and a secret and public key pair for asymmetric encryption.
- Enc: this is the encryption function, given a public key and a plaintext, it outputs a ciphertext
- Dec: this is the decryption function, given a secret key and a ciphertext, it outputs a plaintext
- Eval: this is a HE operation where the inputs are ciphertext (c_1, c_2) and the output is a ciphertext that corresponds to a plaintext function $f()$.

2.4.1 Types of Homomorphic Encryption

Homomorphic encryption can be differentiated depending on the kind and number of mathematical operations that can be computed on the encrypted data.

- **Fully Homomorphic Encryption (FHE)** – allows an unlimited number of Eval operations (addition and multiplication) on the encrypted data with unlimited number of times. Work done in this area includes Gentry's research [25].

- **Partially Homomorphic Encryption (PHE)** – supports only one type of operation (multiplication or addition), an unlimited number of times. Examples of this include RSA [50] and El-Gamal [22], which only support multiplication operations. Goldwasser and Micali [28], Paillier [46] and Benaloh [9] support addition operations.

- **Somewhat Homomorphic Encryption (SWHE)** – allows both addition and multiplication operations, but with a limited number of times, as the size of the ciphertexts grows with each homomorphic operation. An example is Boneh-Goh-Nissim (BGN) [12].

Table 2.1: Types of Homomorphic Encryption

Type	Homomorphic Operation	No. of Homomorphic Operations	Application
Fully Homomorphic Encryption (FHE)	Addition AND Multiplication	Unlimited	Gentry's scheme
Partially Homomorphic Encryption (PHE)	Addition OR Multiplication	Unlimited	RSA, Paillier
Somewhat Homomorphic Encryption (SWHE)	Addition AND Multiplication	Limited	BGN

2.5 Hash Functions

Cryptographic hash functions are of two types,*Keyed hash Functions (Message Authentication Code, MAC)* , which use a secret key and *Unkeyed Hash Functions (referred to as Hash Functions)*, which do not use a secret key [58]. A hash function (H) converts a message (M) of arbitrary size into a fixed size called a digest or hash

value (h), given as $h = H(M)$. Hash functions are one-way, meaning it is computationally impossible to reverse the computation or find the original message given the hash value.

Cryptographic hash functions are expected to have the following properties [55].

- *Collision Resistance:* it is considered computationally infeasible to find messages such that $m \neq m'$ with $h(m) = h(m')$,basically makes it hard for an attacker to find two input values with the same hash.

- *Preimage Resistance:*sometimes referred to as the one-way nature of the hash function, given a random hash function h_0,it is computationally infeasible to find a message m for which $h(m) = h_0$, it would be hard to reverse a hash function.

- *Second Preimage Resistance:*with an input m that has a hash $h(m)$, it should be computationally infeasible to find a different value m' with $h(m) = h(m')$; so given an input and its hash, it would be difficult to find another input with the same hash.

Hash functions can be used for achieving integrity and authentication, storing passwords, implementing digital signatures, digital time stamping, pseudo-random number generators, session key derivations and many more. The most commonly used hash functions are from the SHA (Secure Hash Algorithm) family which includes SHA-1, SHA-2, SHA-3 and RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest).

2.6 Digital Signature

A digital signature is a cryptographic output used to verify the authenticity of data. It uses a private key to create a signature and a corresponding public key to verify

the signature. There are three algorithms in digital signatures: key generation, signing and signature verification algorithms. Digital signatures use hash functions and public-key cryptography. A sender generates a hash value from the message and encrypts it with its private key, then the receiver of the message uses the public key and the hash function to verify if the signature matches the message.

Chapter 3

Models and Design Goals

This chapter outlines the system model, security model and design goals for our proposed scheme.

3.1 System Model

Our system model conceptualizes the 3 main entities and the processes that work together to produce our proposed scheme. Our scheme assumes that our dataset is static and does not consider the addition of new data,the deletion of data or data updates. Our dataset, dimensions and query values are integers. Our scheme is designed to show one query user sending a query request to the cloud servers, our scheme can be extended for multiple users.

Data Owner: The data owner (DO), has a k-dimensional dataset given as $X = \{x_i = x_{i1}, x_{i2}, x_{i3}, \ldots, x_{ik} | i = 1, 2, 3....n\}$ where each data record has a unique identity id_i. The data owner builds and encrypts a kd-tree based index for the data. Due to the large amount of data and the computing resources required, the data owner outsources the encrypted dataset and index to the cloud. The data owner is assumed to be trusted for the proposed scheme and not in any way compromising the security of the outsourced data.

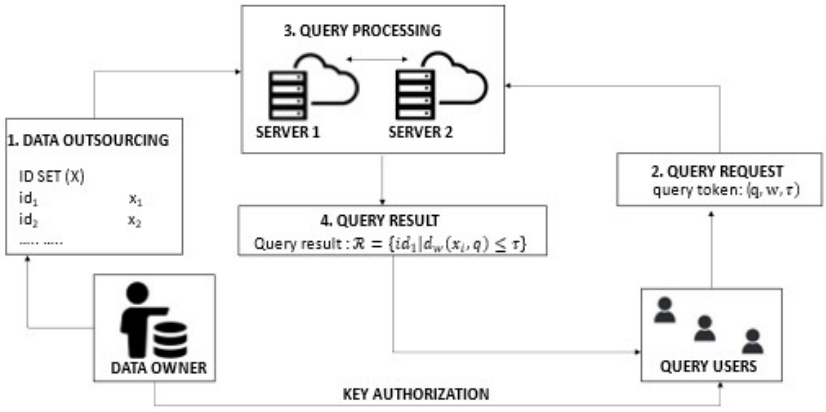

Figure 3.1: System Model

Cloud Servers: Our model has two cloud servers, Cloud Server 1 ($CS1$) and Cloud Server 2 ($CS2$). Each cloud server has powerful computing abilities and high storage capacity that enables them to perform the necessary operations and computations. The $CS1$ stores the encrypted dataset and $CS2$ holds the private key. When a query request (q, w, τ) is sent to the cloud, $CS1$ and $CS2$ work together to search through the encrypted data to perform the necessary computations required to find data records whose distance is less than or equal to the distance threshold (τ). The results are then returned to the appropriate query user.

Query Users: The query users $U_i = \{U_1, U_2, U_3, \ldots, U_n\}$, are registered and authorized by the data owner. Users send an encrypted query request to the cloud server for processing, after which the result is returned to the query user. Query users are independent and query results can only be decrypted by the intended recipient.

3.2 Security Model

The security model assumes that the data owner is trusted as the provider of the encrypted dataset. The dataset is also considered uncorrupted for the proposed scheme. The query users are also assumed to be honest, without malicious intent,

as they launch query requests to the servers. The cloud servers $CS1$ and $CS2$ are assumed to be *honest-but-curious*. Honest-but-curious means that they may be curious about the encrypted contents but will not collude, look into or launch an attack against the encrypted data and associated query processes. We assume the two servers will not collude in any way but will strictly only operate as designed. Since our scheme focuses on preserving the privacy of the similarity query, we do not consider other attacks like denial of service (DoS) attack, access pattern attack, data pollution attack etc. These attacks and others are considered outside the scope of this work and may be considered for future work.

3.3 Design Goals

Our scheme design aims to show that it is privacy-preserving and efficient.

- **Privacy preservation in the proposed scheme:** The first goal of our design is to ensure that the privacy of the outsourced data, the query requests, the query processing and query results are preserved and secure against our honest-but-curious servers $CS1$ and $CS2$.

- **Efficiency in the proposed scheme:** In our quest for efficient and more cost-effective computation, that preserves our above goal of privacy, we will work towards minimizing costs associated with computing and processing our similarity range queries.

Chapter 4

Preliminaries

In this chapter, we introduce the building blocks of our scheme,the Kd-tree, the weighted Manhattan distance, the weighted Manhattan-distance similarity query and the SHE privacy protocols used to implement our scheme.

4.1 KD-Tree

The kd-tree, proposed by Bentley, [10] also known as k-dimensional tree, as it stores points in k-dimensional space. It is a space-partitioning data structure that generalizes the concept of a binary search tree for multidimensional data. The cost of distance calculations on multidimensional data is very high, the kd-tree improves the efficiency of our search algorithm. It does so by pruning whole branches or (subtrees), thereby reducing the number of distance calculations needed to answer a query.

The structure of the kd-tree is made up of the root, the internal nodes and the leaf nodes. The root of the tree represents the entire data-set. Every node is a k-dimensional point. Each internal node v has a cutting dimension (C_d), cutting value (C_v), left sub-tree (T_L), right subtree (T_R). The cutting dimension (C_d) is randomly chosen from all available dimensions and the cutting value (C_v) is the median value

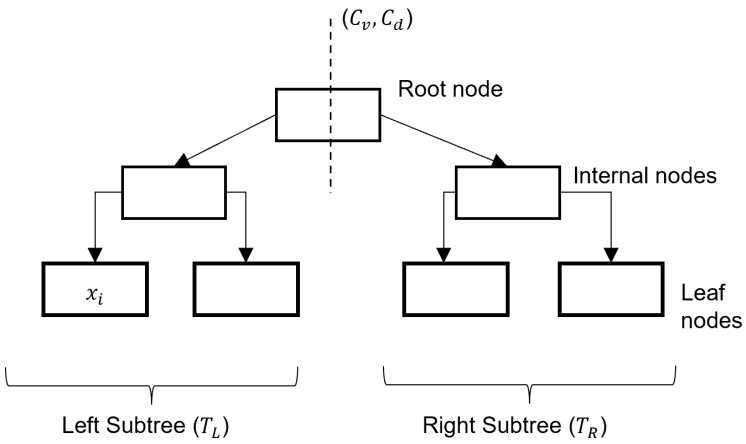

Figure 4.1: KD-Tree Structure

of all data records in the cd^{th} dimension. Each leaf node contains a data record x_i. Algorithm 1 shows the construction of a two-dimensional kd-tree, which can be extended to kd-trees with more dimensions.

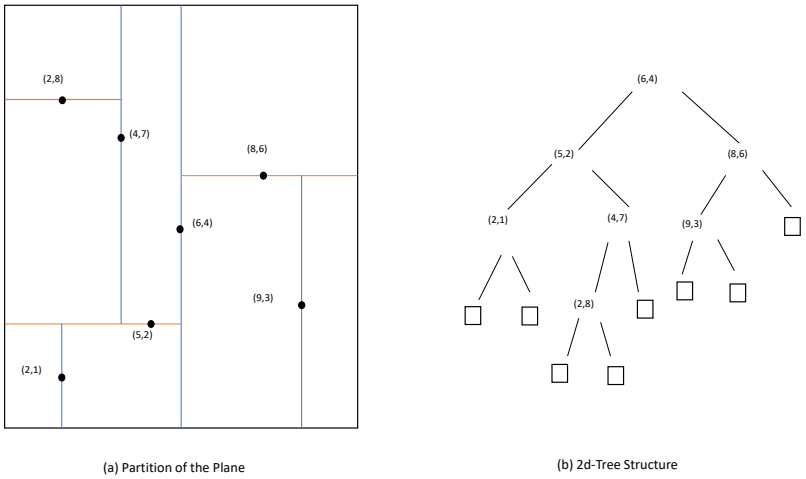

Figure 4.2: 2d Kdtree Range Search

The kd-tree is constructed by recursively dividing the dataset, such that every internal node generates a splitting hyperplane that divides the space into two subspaces.

Algorithm 1 Build Kd-Tree

Input:A set of data records X
Output: The node x

1: **if** X==null **then**
2: **return** null
3: **end if**
4: if X contains only one point
5: **return** x_i (leaf node)
6: **else**
7: C_d, $C_v \leftarrow$ choose cutting dimension and cutting value (X)
8: **for** $x \in X$ do
9: **if** $x_{C_d} < x_{C_v}$ then
10: put x in T_L
11: **else**
12: put x in T_R
13: end if
14: end for
15: x_l = BUILD KD-TREE (T_L, x_l)
16: x_r = BUILD KD-TREE (T_R, x_r)
17: **return** node x

The hyperplane direction is chosen such that every node split to sub-trees is associated with one of the k-dimensions, such that the hyperplane is perpendicular to that dimension vector. So, if for a particular split the x-axis is chosen, all data records in the left sub-tree (T_L) have a value in the cd^{th} dimension that is less than the cutting value (C_v) i.e., $x_{C_d} < C_v$. All the data records whose values in the cd^{th} dimension are greater or equal to the cutting value (C_v), will be in the right sub tree i.e., $x_{C_d} \geq C_v$.

4.1.1 KD-Tree Range Query

A kd-tree range query searches the kd-tree to find a set of data records that lie within a given query range. Suppose we have a k-dimensional dataset $X = \{x_i = x_{i1}, x_{i2}, \ldots, x_{ik} \mid |i = 1, 2, 3, \ldots, n\}$, where n is the number of data records. Let $Q_R = ([x_1^l, x_1^r], [x_2^l, x_2^r] \ldots, [x_k^l, x_k^r])$ denote a range query, where x_i^l and x_i^r are the lower bound and upper bound of the ith query range for $i = 1, 2, \ldots, k$ respectively. The range search algorithm has two stages, filtering and verification. In the filtering

stage, we search the kd-tree to find a set of candidate records $\mathcal{C}$ that are probably with in the query range Q_R. If the searched node is a leaf node, it is added to the set $\mathcal{C}$ of candidate records. If the searched node is an internal node $\{C_d, C_v, T_L, T_R\}$, then we check and compare the query range in the cd^{th} dimension $\{x^l_{C_d}, x^r_{C_d}\}$ with the cutting value C_v as shown in algorithm 2. If $x^l_{C_d} < C_v$, we search the left subtree, if $x^r_{C_d} \geq C_v$, we search the right subtree. In the verification stage, we verify that the

Algorithm 2 Kd-Tree Range Query

Input: Node (*node*), Query Range (Q_R)
Output: Candidate Records($\mathcal{C}$)

```
1: Set C = 0; //initialize candidate record list
2: if node is a leaf node, then;
3: add node's data record to C;
4: else
5: if x^l_{C_d} < C_v then
6: filter (node.T_L, Q_R);
7: if x^r_{C_d} ≥ C_v then
8: filter (node.T_R, Q_R);
9: return C;
```

candidate records x_i in $\mathcal{C}$ are within the query range Q_R.

4.2 Manhattan Distance

The Manhattan distance, was proposed by Hermann Minkowski, [14] who defined it as the sum of the absolute differences of the cartesian coordinates x, y.

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Manhattan Distance also known as L_1-norm, taxicab distance or city block distance calculates the distance traveled if a grid-like path is taken, to get from one data point to another.

Weighted Manhattan Distance: Given a k-dimensional weight vector $w = w_1, w_2, \ldots, w_k$, which satisfies $w_j > 0$ for $1 \leq j \leq k$ and $\sum_{j=1}^{k} w_j = 1$, the weighted Manhattan distance is given as:

$$d_{wm}(x_i, q) = \sum_{j=1}^{k} w_j^* |x_{ij} - q_j|.$$

4.3 Weighted Manhattan Distance Similarity Query

Let $X = \{x_i = x_{i,1}, x_{i,2} \ldots, x_{i,k} | i = 1, 2, 3, \ldots, n\}$ be a data-set with n data records in k-dimensional space. Let (q, w, τ) be a query request, where $q = \{q_1, q_2, \ldots q_k\}$ is a k-dimensional query vector, $w = \{w_1, w_2, \ldots, w_k\}$ is a k-dimensional weight vector, which satisfies $w_j > 0$ for $1 \leq j \leq k$, such that $\sum_{j=1}^{k} w_j = 1$. τ is a distance threshold. A weighted Manhattan distance similarity query searches the data-set and returns all points x_i such that the weighted Manhattan distance is $d_{wm}(x_i, q) = \sum_{j=1}^{k} w_j^* |x_{ij} - q_j| \leq \tau$.

4.4 The SHE Technique

Symmetric Homomorphic Encryption (SHE) [42] provides additive and multiplicative homomorphic operations and basically consists of three algorithms as defined below.

- **SHEKeyGen**(k_0, k_1, k_2): where (k_0, k_1, k_2) are 3 security parameters with $k_1 \ll k_2 < k_0$, the key generation algorithm selects two large prime numbers p and q such that $|p| = |q| = k_0$. A random number $\mathcal{L}$ is chosen such that $|\mathcal{L}| = k_2$. The algorithm calculates the values for $N = pq$, the secret key $(sk) = (p, \mathcal{L})$ and public parameter $(pp) = (k_0, k_1, k_2, N)$, for a message space of $M = \{m | m \in [-2^{k_1-1}, 2^{k_1-1}]\}$.

- **SHEEnc**(m, sk):Given a message $m \in M$, it can be encrypted by a secret key (sk) to become $E(m) = (r\mathcal{L} + m)(1 + r'p) mod N$, where $r \in \{0,1\}^{k_2}$ and $r' \in \{0,1\}^{k_0}$ are random numbers.

  SHE can also encrypt messages with a public key (pk), denoted as

$SHEEnc(pk, m)$. Given two ciphertexts $E(0)_1$ and $E(0)_2$, and random numbers r_1 and $r_2 \in \{0,1\}^{k_2}$. We can encrypt the message with the public key $(pk) = \{E(0)_1, E(0)_2\}$. It outputs a ciphertext given as:

$E(m) = m + r_1 \cdot E(0)_1 + r_2 \cdot E(0)_2 \, mod \, N$

This encryption is IND-CPA secure as proved in [30].

- **SHEDec**$(sk, E(m))$: To decrypt using the sk and E(m), we have: $m' = (E(m) mod \, p) \, mod \, \mathcal{L} = (m + \mathcal{L}) \, mod \, \mathcal{L}$. If $m' < \mathcal{L}/2$,then $m \geq 0$ and $m = m'$. If $m' > \mathcal{L}/2$, then $m < 0$ and $m = m' - \mathcal{L}$.

SHE has the following properties:

(i) Homomorphic addition(I): $E(m_1) + E(m_2) \rightarrow (m_1 + m_2)$

(ii) Homomorphic multiplication(I): $E(m_1) * E(m_2) \rightarrow (m_1 * m_2)$

(iii) Homomorphic addition(II): $E(m_1) + m_2 \rightarrow (m_1 + m_2)$

(iv) Homomorphic multiplication(II): $E(m_1) * m_2 \rightarrow (m_1 * m_2), when \, m_2 > 0$

4.4.0.1 Multiplicative Depth of SHE

The SHE technique is fully homomorphic but it can only support a limited number of homomorphic multiplication(I).If we have two cipher texts:

$E(m_1) = (r_1\mathcal{L} + m_1)(1 + r'_1 p) mod \; N$

$E(m_2) = (r_2\mathcal{L} + m_1)(1 + r'_2 p) mod \; N$

where r_1, r'_1, r_2, r'_2 are random numbers that satisfy the conditions $r_1, r_2 \in \{0,1\}^{k_2}$ and $r_2, r'_2 \in \{0,1\}^{k_0}$.

Applying homomorphic multiplication(I) to $E(m_1)$ and $E(m_2)$:

$E(m_1 * m_2) = E(m_1) * (m_2)$

$= E(r_1\mathcal{L} + m_1)(1 + r'_1 p)(r_2 L + m_2)(r'_2 + r'_2 p)$

$= m_1 * m_2 + r_1 r_2 \mathcal{L}^2 + r_1 \mathcal{L} m_2 + r_2 \mathcal{L} m_1 + \Delta * p$

where Δ is the coefficient of p. We then decrypt the ciphertext $E(m_1 * m_2)$ as:

$(E(m_1 * m_2)\ mod\ p)\ mod\ \mathcal{L}$

$= ((m_1 * m_2 + r_1 r_2 \mathcal{L}^2 + r_1 \mathcal{L} m_1 + r_2 \mathcal{L} m_2 + \Delta * p)\ mod\ p)\ mod\ \mathcal{L}$

$= ((m_1 * m_2 + r_1 r_2 \mathcal{L}^2 + r_1 \mathcal{L} m_1 + r_2 \mathcal{L} m_2)\ mod\ p)\ mod\ \mathcal{L}$

The decryption is correct $iff\ m_1 * m_2 + r_1 r_2 \mathcal{L}^2 + r_{\mathcal{L}} L m_1 + r_2 \mathcal{L} m_2 < p$. Since $m_1, m_2 \in \{0,1\}^{k_1}$,$r_1, r_2, \mathcal{L} \in \{0,1\}^{k_2}$ and $k_1 \ll k_2$, we can deduce that $m_1 * m_2 + r_1 \mathcal{L} m_1 + r_2 \mathcal{L} m_2 \ll r_1 r_2 \mathcal{L}^2$. To guarantee that the decryption is correct,$r_1 r_2 \mathcal{L}^2 < p$. If the required multiplicative depth (the maximum number of consecutive multiplications) is θ , then $\theta = [\frac{k_0}{2k_2} - 1]$.

4.4.1 SHE Privacy-Preserving Protocols

In this subsection we review some secure protocols built upon SHE which involve two cloud servers, cloud server 1 and cloud server 2.

- **Absolute Value Computation Protocol**

  Using this protocol, the cloud servers compute the distance between two points: the authorized query user's query record and the outsourced data. Based on the SHE technique, the absolute value computation protocol computes $E(|m_1 - m_2|)$. Cloud Server 1 ($CS1$) holds the ciphertexts $E(m_1)$ and $E(m_2)$ and Cloud Server 2 ($CS2$) holds the private key sk. $CS1$ preserves the privacy of m_1 and m_2 by permuting the values as it calculates $E(|m_1 - m_2|)$ while also working with $CS2$.

  Step 1: Based on the homomorphic properties of SHE, $CS1$ computes $E(m) = E(m_1 - m_2) = E(m_1) + E(m_2) * E(-1)$ and also computes $E(x) = (r_1 * m + r_2)$. Here, r_1, r_2 are random numbers where $r_1, r_2 \in \{0,1\}^{k_1}$ such that $r_1 > r_2 > 0$. $CS1$ then sends $E(x)$ to $CS2$.

  Step 2: When $CS2$ receives $E(x)$, it uses the secret key sk to recover the

plaintext x. $CS2$ returns an encrypted value $E(b)$ to $CS1$ which depends on the sign of x, such that $E(b) = E(-1)$ if $x < 0$ and $E(b) = E(1)$ if $x > 0$.

Step 3: When $CS1$ receives $E(b)$, it then computes $E(|m_1 - m_2|) = E(b) * E(m_1 - m_2)$.

Correctness: the absolute value computation protocol is correct iff $E(|m_1 - m_2|) = E(b) * E(m_1 - m_2)$, such that $|m_1 - m_2| = m_2 - m_1$ when $m_1 < m_2$ and $m_1 - m_2$ otherwise.

Proof: When $m_1 < m_2$, we have

$m = m_1 - m_2 < 0$

$x = r_1 * m + r_2 < 0$

When $b = -1, E(|m_1 - m_2|) = E(b * E(m_1 - m_2)) = E(m_2 - m_1)$

When $m_1 \geq m_2, E(|m_1 - m_2|) = E(m_1 - m_2)$

Therefore the absolute value computation protocol is correct.

- **Comparison Protocol 1**

The comparison protocol is used in both the kd-tree technique and the comparison of distances. From this protocol, we can only obtain a comparison of two ciphertexts $E(m_1)$ $E(m_2)$ without leaking any plaintext values of m_1 and m_2. We compute a message's encrypted sign given a ciphertext $E(m)$, when $m < 0$, the sign of m is -1, encrypted as $E(-1)$. Otherwise, when $m \geq 0$, the encrypted sign of m will be $E(1)$. Servers $CS1$ and $CS2$ can find $E(sign(m))$ and preserve the privacy of m, where $CS1$ has the ciphertext $E(m)$ where $M = \{m|m \in [-2^{k_1-1}, 2^{k_1-1}]\}$ and $CS2$ has the corresponding private key sk.

Step 1: Flip a coin to randomly pick a boolean value b. If b is true, then swap $E(m_1)$ and $E(m_2)$.

Step 2: Calculate $E(m) = E(m_1 - m_2) = E(m_1) + E(m_2) \times E(-1)$.

Step 3: $CS1$ chooses r_1 and r_2 which are two random numbers where $r_1, r_2 \in$

$\{0,1\}^{k_1}$ satisfy the condition $r_1 > r_2 > 0$.

Step 4: $CS1$ sends $E(x)$ to $CS2$, where $E(x) = E(r_1 * m + r_2)$.

Step 5: $CS2$ uses the sk to recover x from $E(x)$. From $CS2$, based on the sign of x, $sign(m) = -1$ if $x < 0$ and $sign(m) = 1$ otherwise. $CS2$ then sends $sign(m)$ to $CS1$.

Step 6: If b is true, $CS1$ returns $-sign(m)$, otherwise it returns $sign(m)$.

Correctness: this computation is correct iff

$$m < 0, sign(m) = -1 \tag{4.1}$$

$$m \geq 0, sign(m) = 1 \tag{4.2}$$

(4.1) and (4.2) are equivalent to each other, therefore we will prove the equivalence of $m < 0$, $sign(m) = -1$.

Proof: If $CS1$ computes $E(x) = E(r_1 * m + r_2)$ and sends it to $CS2$ which decrypts $E(x)$ to get plaintext $x = r_1 * m + r_2$. If $r_1 > r_2 > 0$ when $m < 0$, $r_1 * m + r_2 < 0$, then $CS2$ will send $sign(m) = -1$ to $CS1$. Since $sign(m) = -1$ means that $x < 0$, it means $r_1 * m + r_2 < 0$. Therefore, the protocol is correct and $m < 0 \Leftrightarrow sign(m) = -1$ holds.

- **Comparison Protocol 2**

  We can modify comparison protocol 1 to obtain a different version where the output obtained by $CS1$ is encrypted. Steps 1 - 4 remain the same as comparison protocol 1.

  - *Step 5:* $CS2$ uses the sk to recover x from $E(x)$. Based on the sign of x, it sets $\sigma = 0$ if $x < 0$, and $\sigma = 1$, otherwise. It sends $E(\sigma)$ to $CS1$.
  - *Step 6:* If b is true, $CS1$ returns $1 + E(-1) \cdot E(\sigma)$, otherwise, it returns, $E(\sigma)$.

Chapter 5

Proposed Scheme

This chapter describes the details of our proposed scheme, showing how the different entities work together to return the query result to the query user.

5.1 Kd-Tree Based Weighted Manhattan Distance Similarity Query

The kd-tree based weighted Manhattan distance-based similarity query algorithm takes a user generated query request (q, w, τ) and searches the kd-tree (T_{kd}) to obtain data records whose weighted Manhattan distance $\{d_{wm}(x_i, q) = \sum_{j=1}^{k} w_j^* |x_{ij} - q_j|\}$ from a specified query record (q_j) are within a specified distance threshold (τ) i.e. $d_{wm}(x_i, q) \leq \tau$.

For a kd-tree based weighted Manhattan distance-based similarity query, we have a multidimensional dataset $\{X = (x_i = x_{i1}, x_{i2}, \ldots, x_{ik}) | i = 1, 2, 3, \ldots, n\}$, that stores our data records and is indexed using a kd-tree (T_{kd}), and a query request (q, w, τ) is sent to the cloud servers for query processing. The algorithm searches through the internal nodes of the kd-tree to find the query results in plaintext. As the algorithm searches the kd-tree's internal nodes u, it first checks to find the minimum weighted

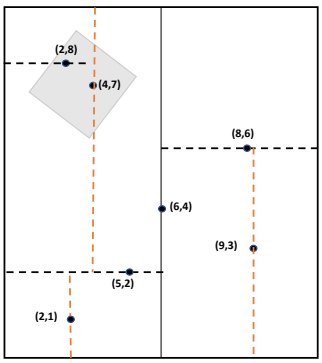

Figure 5.1: 2d Kd-Tree Range Query Using Manhattan Distance

Manhattan distance $\{d_{min}(q, M_{C_d,C_v}) = w_k * |q_k - C_v|\}$ from the query vector q to the median M of the internal node u, which is a value that divides the records attached to u. We then check to see if our weighted Manhattan distance is greater than our query threshold $\{d_{wm}(q, M_{Cd,Cv}) > \tau\}$, to achieve efficient searching.

1. If $C_v - q_{C_d} > \frac{\tau}{w_{C_d}}$, then prune the right sub-tree T_R and search the left sub-tree $T_{kd}.T_L$.

2. If $q_{C_d} - C_v > \frac{\tau}{w_{C_d}}$,we prune the left sub-tree T_L and search the right sub-tree $T_{kd}.T_R$.

3. Otherwise, we search both left and right subtrees.

4. If searching a leaf node, we determine if $d_w(x_i, q) \leq \tau$, if yes, we add that leaf node to our $\mathcal{R} = \mathcal{R} \cup (x_i)$.

Algorithm 2 is correct if the pruning conditions $C_v - q_{C_d} > \frac{\tau}{w_{C_d}}$ and $q_{C_d} - C_v > \frac{\tau}{w_{C_d}}$

Algorithm 3 Kd-tree Based Weighted Similarity Range Query on Plaintext

Input: Kd-Tree (T_{kd}), Query (q, w, τ)
Output: Query Result (R)

Set $R = \emptyset$; //initialize query result
Search $(T_{kd}, (q, w, \tau), \mathcal{R})$;
return $\mathcal{R}$;
Search ((T_{kd}), Query (q, w, τ))
if node is a leaf with x_i, then
if $d_w(x_i, q) \leq \tau$ then
$\mathcal{R} = \mathcal{R} \cup \{x_i\}$;
else if node is an internal node with (C_d, C_v, T_L, T_R) then
if $C_v - q_{C_d} > \frac{\tau}{w_{C_d}}$ then
Search $T_{kd}.T_L, (q, w, \tau), \mathcal{R}$;
else if $q_{C_d} - C_v > \frac{\tau}{w_{C_d}}$ then
Search $T_{kd}.T_R, (q, w, \tau), \mathcal{R}$;
else
Search $T_{kd}.T_L, (q, w, \tau), \mathcal{R}$;
Search $T_{kd}.T_R, (q, w, \tau), \mathcal{R}$;

are correct, such that:

$$C_v - q_{C_d} > \frac{\tau}{w_{C_d}} \Leftrightarrow w_{C_d} * |C_v - q_{C_d}| > \tau \tag{5.1}$$

Given a hyperplane M_{C_d,C_v} that is perpendicular to the ${C_d}^{th}$-dimension with cutting values given as C_v, the weighted Manhattan distance between q and m_{C_d,C_v} is given as:

$$d_w(q, M_{C_d,C_v}) = w_{C_d} * |C_v - q_{C_d}| \tag{5.2}$$

Based on eqn. 5.1, $d_w(q, M_{C_d,C_v}) > \tau$, we can infer for $C_v - q_{C_d} > \frac{\tau}{w_{C_d}}$, all query results in the left sub-tree, and we will therefore prune the right subtree. Similarly, when $q_{C_d} - C_v > \frac{\tau}{w_{C_d}}$ for all query results in the right sub-tree, we will prune the left subtree. Therefore our algorithm 2 is correct.

5.2 Description of our Scheme

There are four stages in our scheme, namely the system initialization, data outsourcing, query token generation and query processing stages.

Main Idea. Calculating the Manhattan distance over ciphertext is computationally expensive. We therefore employ the kd-tree weighted Manhattan distance similarity query to achieve a filtering and verification approach. Specifically, in the filtering phase, we obliviously traverse the encrypted kd-tree based on the SHE based absolute value computation protocol and the comparison protocols as seen in section 4.4.1 and add all visited leaf nodes to the candidate set. In the verification phase, we refine the candidate set by removing all records that do not satisfy the Manhattan distance condition.

5.2.1 System Initialization

The data owner (DO) initializes the system by selecting three security parameters k_0, k_1, k_2 and applying the SHEKeyGen algorithm to generate a private key (sk) and a public key (pk). The DO also generates three ciphertexts $E(0)_1, E(0)_2$ and $E(-1)$. The ciphertexts $\{E(0)_1, E(0)_2\}$ give us the public key (pk) and $\{E(-1)\}$ is a public parameter (pp). The DO then chooses a secure hash function $H(\cdot)$ and randomly generates a secret key (K) for the AES scheme, which is used to authorize the query users (U_i). It then publishes $(pk, E(0)_1, E(0)_2, E(-1))$ and the private key sk is sent to cloud server 2 ($CS2$).

5.2.2 Data Outsourcing

To outsource our dataset to cloud server one ($CS1$), we take the following steps.

Step 1: We build a kd-tree from the dataset $X = \{x_i = (x_{i1}, x_{i2}, \ldots, x_{ik})\}_{i=1}^{n}$.

Step 2: Let $V = \{v_1, v_2, \ldots, v_n\}$ be an array of all internal nodes in the kd-

tree. Each internal node $v_i \in V$ has key values $\{C_d, C_v, T_L, T_R\}$, and may be represented as a $(k+3)$-dimensional vector $v_i = (C_{d,1}, C_{d,2}, \ldots, C_{d,k}, C_v, T_L, T_R)$, where $C_{d,i} = 1$ when $i = C_d$ otherwise $C_{d,i} = 0$. Encrypt the internal nodes in the kd-tree so each v_i is encrypted by the SHE public key (pk) to give $\{E(v_i) = \{E(C_d)\}_{i=1}^{k}, E(C_v), E(T_L), E(T_R)\}$.

Step 3: The leaf nodes x_i are represented as k-dimensional vectors $\{X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})\}$. The data owner then encrypts leaf nodes (x_i) of the kd-tree with the SHE public key to give the ciphertext $E(x_i) = \{E(x_{i,j})\}_{j=1}^{k}$.

Step 4: The data owner then sends the encrypted kd-tree $E(T_{kd})$ to cloud server 1 ($CS1$).

5.2.3 Query Token Generation

In this phase, the query user U_i generates a plaintext query request (q, w, τ) and encrypts it using the $SHEEnc(pk, m)$ to produce the encrypted query request ($E(q)$, $E(w), E(\tau)$). The query user then constructs a query range $Q_R = \{q_i - \lceil \frac{\tau}{w_i} \rceil, q_i + \lceil \frac{\tau}{w_i} \rceil\}_{i=1}^{k}$ and also encrypts it to $E(Q_R)$. The query user also chooses a session key K which assists the user in recovering the query result. The encrypted query token $\{E(Q_T) = (E(q), E(w), E(\tau), E(Q_R), E(K)\}$ is sent to cloud server one ($CS1$) through a secure channel.

5.2.4 Processing the Query

In processing the query, the scheme uses a filter-and-verify approach as described in the following sections.

5.2.4.1 Filtering Phase

In the filtering phase of the query process, the following steps are taken to find a set of candidate data records that may be within the required distance threshold.

- *Step 1:* $CS1$ receives the encrypted query token $E(Q_T)$ and confirms that the query user is authorized, if not the query is discarded. Initialize an empty candidate set $\mathcal{C} = \emptyset$.

- *Step 2:* $CS1$ traverses the encrypted kd-tree from its root node. For each encrypted internal node $E(v_i)$, $CS1$ collaborates with $CS2$ to determine whether to visit the left or right child.

  (i) $CS1$ extracts the query range for the ${C_d}^{th}$ dimension by homomorphically calculating the lower bound for $C_{d,i}$ for $E(q_{C_d} - \lceil \frac{\tau}{w_{C_d}} \rceil) = E(\sum_{i=1}^{k}(q_i - \lceil \frac{\tau}{w_i} \rceil))$. Similarly, $CS1$ also extracts the upper bound for $C_{d,i}$ by calculating $E(q_{C_d} + \lceil \frac{\tau}{w_{C_d}} \rceil) = E(\sum_{i=1}^{k}(q_i + \lceil \frac{\tau}{w_i} \rceil))$.

  (ii) $CS1$ sends the calculated lower bound $E(q_{C_d} - \lceil \frac{\tau}{w_{C_d}} \rceil)$ and upper bound $E(q_{C_d} + \lceil \frac{\tau}{w_{C_d}} \rceil)$ to $CS2$.

  (iii) $CS2$ uses sk to decrypt the lower and upper bounds for the ${C_d}^{th}$ dimension and runs a simple comparison.

  (iv) If $E(C_v) \geq E(q_{C_d} - \lceil \frac{\tau}{w_{C_d}} \rceil)$, the left subtree is traversed

  (v) If $E(C_v) \leq E(q_{C_d} + \lceil \frac{\tau}{w_{C_d}} \rceil)$, the right subtree is traversed.

  (vi) $CS2$ sends the comparison results for left or right traversal of the tree to $CS1$

- *Step 3:* $CS1$ chooses two random numbers r_{1,C_d}, r_{2,C_d} for the encrypted difference $E(q_{C_d}) - E(C_v)$, $CS1$ also chooses two random numbers $r_{1,\tau}, r_{2,\tau}$ for $\frac{E(\tau)}{E(w_{C_d})}$.

- *Step 4:* $CS1$ computes $E(x_{C_d}) = E(r_{1,C_d}) \cdot (q_{C_d} - C_v + r_{2,C_d})$ and $E(x_\tau) = E(r_{1,\tau} \cdot (\frac{\tau}{w_{C_d}}) + r_{2,\tau})$. $CS1$ sends both $E(x_{C_d})$ and $E(x_\tau)$ to $CS2$.

- *Step 5:* $CS2$ uses the SHE private key (sk) to decrypt $E(x_{C_d})$ and $E(x_\tau)$ to recover x_{C_d} and x_τ.

- *Step 6:* $CS2$ deduces the sign for x_{C_d} and determines if $(q_{C_d} - C_v) \geq 0$ or $(q_{C_d} - C_v) < 0$.

- *Step 7:* For $E(x_\tau)$, $CS2$ deduces if $\frac{\tau}{w_{C_d}} \geq 0$ or $\frac{\tau}{w_{C_d}} < 0$. $CS2$ then returns $E(sign(x_{C_d}))$ and $E(sign(s_\tau))$ to $CS1$.

- *Step 8:* $CS1$ decrypts both values to obtain the encrypted signs to determine if $E(C_v) - E(q_{C_d}) > \frac{E(\tau)}{E(w_{C_d})}$. Since the values are permutated, $CS1$ recovers the actual values.

- *Step 9:* The filtering results give a set of candidate data records $\mathcal{C} = E(x_i)$, which are potentially within the query result range.

Algorithm 4 Filtering Phase

Input: Kd-Tree $E(T_{kd})$, Encrypted Query Token (Q_T)
Output: Candidate Records($\mathcal{C}$)

1: Set $\mathcal{C} = 0$; //initialize candidate result
2: if node is a leaf node, then add data record to $\mathcal{C}$;
3: else
4: if $E(C_v) < E(q_{C_d} - \frac{\tau}{w_{C_d}})$
5: Filtration $(E(node.T_L), E(Q_T))$
6: if $E(C_v) > E(q_{C_d} + \frac{\tau}{w_{C_d}})$
7: Filtration $(E(node.T_R), E(Q_T))$
8: return $\mathcal{C}$;

5.2.4.2 Result Verification Phase

The verification phase confirms that these candidate records meet the criteria as defined by the range query. In the verification phase, we compute to verify if for every $E(x_i) \in \mathcal{C}$, where it's weighted Manhattan distance (d_{wm}) to q is calculated, if it satisfies the condition $d_{wm}(x_i, q) \leq \tau$ or not.

- *Step 1:* $CS1$ has a list of encrypted candidate records, $\mathcal{C} = E(x_i)$ with which may be within the required threshold distance $d_{wm}(x_i, q) \leq \tau$.

- *Step 2:* $CS1$ calculates the encrypted weighted Manhattan distance between each candidate record p_i and the query q using the absolute value protocol as described in section 4.4.1.

- *Step 3:* $CS1$ and $CS2$ use the comparison protocol as shown in section 4.4.1 to compare the inequality $d_w(x_i, q) \leq \tau$. If (x_i) satisfies the query condition, it returns the encrypted result bit as $E(b_i) = E(1)$ otherwise $E(b_i) = E(0)$.

- *Step 4:* For each $E(p_i) \in \mathcal{C}$, $CS1$ calculates $E(p_i)' = E(b_i) \cdot E(p_i)$, such that those values are returned as verified records $\mathcal{C}' = \{E(p_i)'\}$.

5.2.4.3 Result Re-encryption

After the results have been verified, they need to be returned to the query user in a privacy-preserving way.

- *Step 1:* For every $E(p_i)' \in \mathcal{C}'$, $CS1$ generates a k-dimensional random vector such that $r_i = \{r_{i1}, r_{i2}, \ldots, r_{i,k}\}$ to get $\mathcal{C}''$. $CS1$ sends $\mathcal{C}'' = \{E(p_i)'' = (E(p_i)' + r_i)\}, E(K)$ to $CS2$.

- *Step 2:* $CS2$ receives $\mathcal{C}''$ from $CS1$ and uses the sk to recover the SHE plaintext, $\mathcal{C}_{pt} = \{(p_i)''\}, K$. $CS2$ then re-encrypts C_{pt} with K to obtain $C_{AES} = \{K(p_i'')\}$ and returns it to $CS1$, which in turn sends it along with the random vectors to the query user through a secure channel.

- *Step 3:* The query user will decrypt using K and subtract the random vectors to recover the query results.

Thus we satisfy the query conditions based on the comparison of the weighted distance, which ensures the correct identification and retrieval of the results, $R = \{x_i | d_w(x_i, q) \leq \tau\}$.

Algorithm 5 Weighted Manhattan Distance-Based Similarity Query over Ciphertext

Input: Encrypted Kd-Tree $E(T_{kd})$, Query Token $E(Q)$
Output: Query Result$\mathcal{R}$

Set $\mathcal{R} = 0$; //initialize query result
for $E(q, w, \tau)$
Search $E(T_{kd})$
Find $d_{wmin}(q, m_{C_d,C_v}) = w_k * |q_k - C_v| > \tau$
Filter results
$E(C_v) < E(q_{C_d} - \frac{\tau}{w_{C_d}})$, search $E(T_L)$
$E(C_v) > E(q_{C_d} + \frac{\tau}{w_{C_d}})$, search $E(T_R)$
$\mathcal{C}$
Verify results
for each $x_i \in C$ do
if $d_w(x_i, q) \leq \tau$
Return result $\mathcal{R} = \{x_i | d_w(x_i, q) \leq \tau\}$

Chapter 6

Security Analysis

This chapter presents the security of the proposed scheme and the protocols used to implement it.

6.1 Security of Our Privacy Preserving Protocols

- The absolute value protocol computes the value for $E(m) = E(|m_1 - m_2|)$ based on the SHE technique. The absolute value protocol is run between $CS1$ and $CS2$, the two servers cannot obtain the information about the encrypted values $E(m_1)$ and $E(m_2)$ as well as their difference $m_1 - m_2$ and $sign(m_1 - m_2)$. Cloud server 1 ($CS1$) holds the ciphertexts $E(m_1)$, $E(m_2)$ and $E(b)$, and is unable to decrypt those values. They have been encrypted by SHE which has been shown to be semantically secure against CPA (chosen plaintext attack). Cloud server 2 ($CS2$) holds the private key (sk) and is able to recover the plaintext x from $E(x) = (r_1 * m + r_2)$, where $m = m_1 - m_2$ or $m = m_2 - m_1$. However, due to the randomization of x using r_1 and r_2, $CS2$ is unable to obtain the true value of x or m.

- The comparison protocol 1 uses the SHE technique to encrypt the ciphertexts

$E(m_1)$ and $E(m_2)$ and compare their result using $CS1$ and $CS2$ to obtain the encrypted sign of the given ciphertext $E(sign(m))$. $CS1$ is further able to obfuscate the relationship between m_1 and m_2 by randomly flipping a coin to pick a boolean value b, such that $CS2$ cannot know which value is assigned to m_1 or m_2. Thus when $CS2$ returns $sign(m)$ to $CS1$ only $CS1$ can assign the right encrypted values to ensure the security and correctness of $E(m)$. $CS1$ has the ciphertext $E(m) = E(m_1 - m_2) = E(m_1) + E(m_2) \times E(-1)$ and is unable to obtain any information about $E(m)$. $CS1$ sends $E(m)$ with random values r_1 and r_2 to $CS2$ as $E(x)$. $CS2$ has the private key sk and can recover plaintext x where $E(x) = E(r_1 * m + r_2)$, but is unable to obtain any information about x due to the added random values.

- Comparison protocol 2 similar to comparison protocol 1 also has $CS1$ hold the encrypted values of $E(m_1)$, $E(m_2)$ and $E(m)$ and sends $E(x)$ to $CS2$ for further computation. However, instead of computing $sign(m)$, $CS2$ recovers x and based on its sign, flags it as $\sigma = 0$ or 1 and returns $E(\sigma)$ to $CS1$. Our comparison protocol 2 is able to securely compute our encrypted results such that $CS1$ and $CS2$ are unable to obtain plaintext values of our ciphertexts.

6.2 Security of our Query Scheme

- Cloud server 1 $CS1$ holds the encrypted Kd-tree and is unable to know the values of the data it holds as our SHE encryption technique has been shown to be semantically secure over CPA. Similarly, when $CS1$ receives the encrypted query token $E(Q_T)$, it is also unable to obtain or infer any information about its plaintext values. All operations on $CS1$ are done over SHE ciphertexts, therefore,$CS1$ cannot see or recover the plaintexts of $E(q), E(w), E(\tau)$ or infer which dimension is selected from the query request.

- Cloud server 2 $CS2$ holds the private key sk and is able to decipher the encrypted values sent to it for computation. $CS2$ cannot however decipher the random values r_1 and r_2 to obtain the plaintext values and due to the obfuscation of $E(m_1)$ and $E(m_2)$ by $CS1$, it cannot know which unencrypted value correctly corresponds to $E(m_1)$ or $E(m_2)$.

Chapter 7

Performance Evaluation

This chapter shows the results of our performance evaluation from the simulation of our weighted Manhattan distance-based similarity query. The performance of our scheme is evaluated based on the computational costs of outsourcing our data, query token generation, and processing our query results, as we vary the dataset size n and the number of data dimensions d.

7.1 Experiment Settings

Our proposed scheme was implemented in java and run on a Windows 10 PC with 16GB RAM, 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 1.69 GHz, 64-bit operating system, x64-based processor. We evaluated the performance using the EEG Eye State dataset from the UCI machine learning depository [51], with 14980 records, each with 15 dimensions. The security parameters were set as $k_0 = 2048$, $k_1 = 160$, $k_2 = 80$. Our maximum multiplicative depth is then given as $\theta = [\frac{k_0}{2k_2} - 1]$, such that $\theta = [\frac{1024}{2(60)} - 1]$, giving us $\theta \approx 9$. Our scheme does not exceed this depth.

Table 7.1: Experiment Environment Table

Parameter	Value
Operating System	Windows 10
CPU	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
RAM	16GB
Programming Language	Java
SHE Security Parameters	$k_0 = 2048$, $k_1 = 160$, $k_2 = 80$

7.2 Experiment Results and Analysis

We present our results and the analysis of the results in the sections below.

7.2.1 Data Outsourcing

In evaluating the computational cost of outsourcing our dataset, we consider the process of building and encrypting our kd-tree. We take into consideration two parameters, the size of our dataset n, and the number of dimensions d. We set $n = \{1000, 4000, 6000, 8000, 10000, 12000\}$, and $d = \{4, 6, 8, 10, 12, 14\}$. The figure below shows the result of our experiment data. As we increase the dimensinality

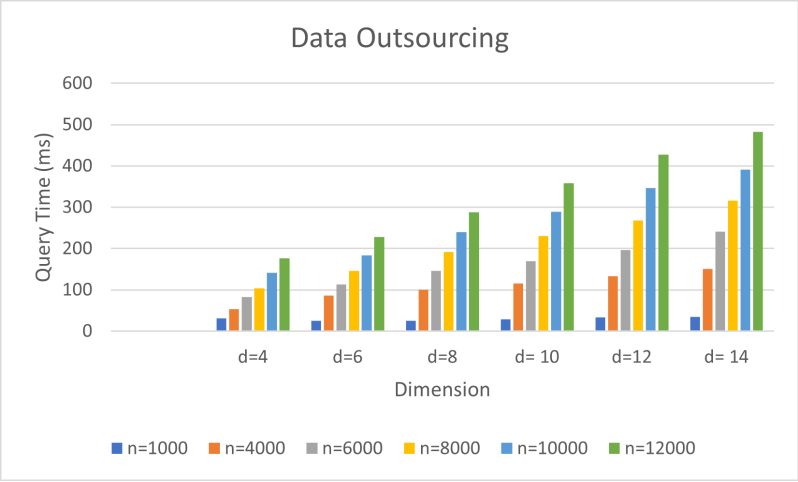

Figure 7.1: Computational Cost of Data Outsourcing

and size of our dataset, we see a corresponding linear increase over time. There is

a significant time cost increase as the number of dimensions d increase. The size of the dataset also impacts the time cost significantly, with higher data sizes showing a marked increase. Overall, the relationship is linear and indicates an efficient process.

7.2.2 Query Token Generation

We evaluate the computational cost of query token generation as our dimensions, $d = \{4, 6, 8, 10, 12, 14\}$, increase. There is a corresponding increase in time as we increase the number of dimensions.

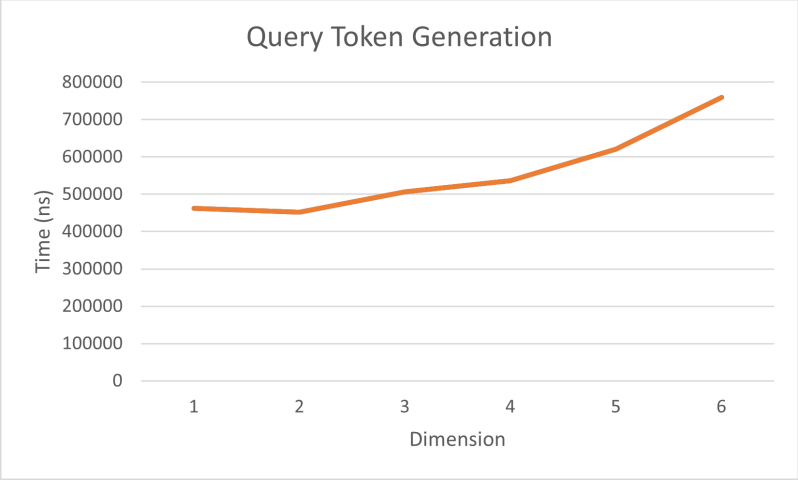

Figure 7.2: Query Token Generation

7.2.3 Query Processing

In evaluating the filtering process, our experiment filters through a sample set of records,$\{100, 200, 300, 400, 500, 600\}$ for a set number of dimensions, $d = \{4, 6, 8, 10, 12, 14\}$ and measures the time taken. We simulate the filtration process as shown in the figure 7.3 below.

There is a corresponding linear increase in time as the number of dimensions increase. We evaluate our scheme's verification process by verifying which candidate record matches the query request. We measure the time taken to verify a set num-

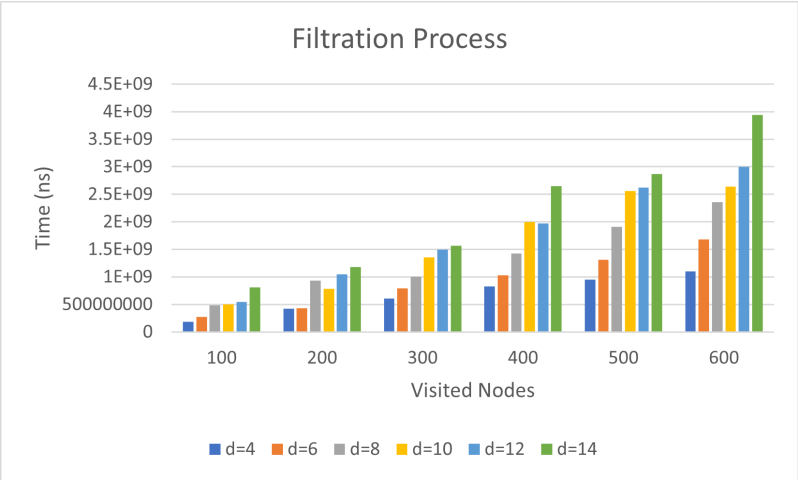

Figure 7.3: Filtration Process

ber of records, $VR = \{50, 100, 200, 500, 1000, 5000\}$ for each dimension, $d = \{4, 6, 8, 10, 12, 14\}$. Figure 7.4 below shows our verification process over the specified dimensions.

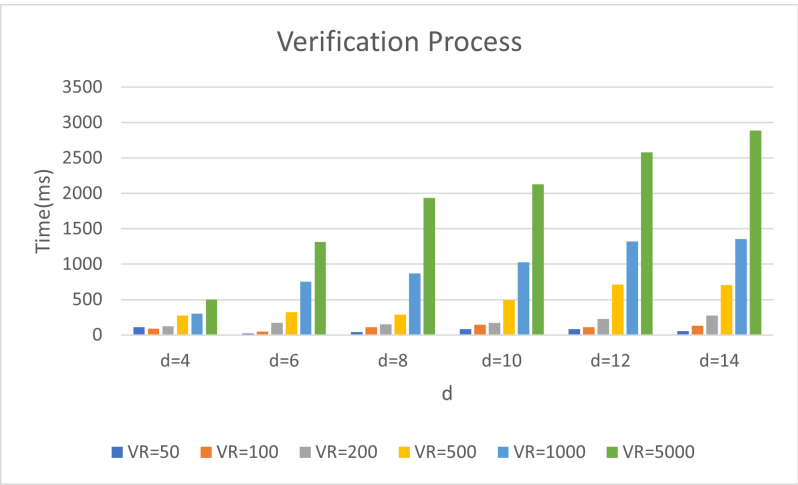

Figure 7.4: Verification Process

Chapter 8

Conclusion and Future Work

In our concluding chapter, we will discuss the summary of our proposed scheme and its performance, and discuss possible directions for future work.

8.1 Conclusion

In this paper, we proposed an efficient and privacy-preserving weighted Manhattan based similarity query for multidimensional data, using a kd-tree to index the data. We applied the SHE technique to various privacy-preserving protocols that enhanced the security of our scheme. Our security analysis demonstrated the security and privacy-preserving nature of our scheme. We further performed a performance evaluation and analyzed the results to validate our scheme.

8.2 Future Work

The study of similarity queries on encrypted data and its applications to various fields of research especially healthcare is important as researchers continually look for more efficient, secure and privacy-preserving ways of computation.

Directions for further research on this topic include looking at the use of dynamic

datasets, as our work focused on the use of a static dataset. Considerations can be made for adding, deleting, and updating the dataset, while ensuring the security, privacy, and integrity of the data structure, computations and other entities involved in the query process.

Homomorphic encryption still has many challenges in terms of its usability and operational limitations. The use of efficient and cost-effective computation using secure encryption techniques are also aspects of future work.

There are also challenges with securing the outsourced data. With the rise of artificial intelligence and machine learning techniques, honest-but-curious servers may no longer be assumed to be free from collusion, as machines are increasing being trained on models that enable them to learn ways of bypassing established controls and security protocols. Other threats exist to cloud architecture, storage and secure data communication channels that can be considered for future work.

Future work can also look at the limitations of the kd-tree in high-dimensional spaces due to the curse of dimensionality, where efficiency reduces at higher dimensions. Future work may consider the use of a more efficient data structure.

In terms of practicality, our scheme was not implemented in a real world scenario. Future work can look at practically deploying our scheme in a production environment, and the resources required to fully implement it.

Bibliography

[1] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti, *A survey on homomorphic encryption schemes: Theory and implementation*, ACM Computing Surveys (Csur) **51** (2018), no. 4, 1–35.

[2] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim, *On the surprising behavior of distance metrics in high dimensional space*, International conference on database theory, Springer, 2001, pp. 420–434.

[3] Bader Alouffi, Muhammad Hasnain, Abdullah Alharbi, Wael Alosaimi, Hashem Alyami, and Muhammad Ayaz, *A systematic literature review on cloud computing security: threats and mitigation strategies*, IEEE Access **9** (2021), 57792–57807.

[4] Mutasem K Alsmadi, *Content-based image retrieval using color, shape and texture descriptors and features*, Arabian Journal for Science and Engineering **45** (2020), no. 4, 3317–3330.

[5] Oriehi Edisemi Destiny Anyaiwe, Gautam B Singh, George D Wilson, and Timothy J Geddes, *Weighted manhattan distance classifier; seldi data for alzheimer's disease diagnosis*, 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 257–262.

[6] Nikolaus Augsten, *A roadmap towards declarative similarity queries*, 21st International Conference on Extending Database Technology, EDBT 2018, OpenProceedings. org, 2018, pp. 509–512.

[7] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway, *Relations among notions of security for public-key encryption schemes*, Annual International Cryptology Conference, Springer, 1998, pp. 26–45.

[8] Richard Bellman, *Dynamic programming*, Science **153** (1966), no. 3731, 34–37.

[9] Josh Benaloh, *Dense probabilistic encryption*, Proceedings of the workshop on selected areas of cryptography, 1994, pp. 120–128.

[10] Jon Louis Bentley, *Multidimensional binary search trees used for associative searching*, Communications of the ACM **18** (1975), no. 9, 509–517.

[11] Christian Böhm, Stefan Berchtold, and Daniel A Keim, *Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases*, ACM Computing Surveys (CSUR) **33** (2001), no. 3, 322–373.

[12] D Boneh, EJ Goh, and K Nissim, *Evaluating 2-dnf formulas on ciphertexts. ts. theory of cryptography: Proceedings of the 2nd theory of cryptology conference (tcc'05), feb 10- 12, 2005, cambridge, ma, usa. lncs 3378*, 2005.

[13] Pedro Ramos Brandao and Manuel Rezende, *Data: The most valuable commodity*, Kriativ. tech **8** (2020), no. 1, 1–7.

[14] Sung-Hyuk Cha, *Comprehensive survey on distance/similarity measures between probability density functions*, City **1** (2007), no. 2, 1.

[15] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín, *Searching in metric spaces*, ACM computing surveys (CSUR) **33** (2001), no. 3, 273–321.

[16] Paolo Ciaccia, Marco Patella, Pavel Zezula, et al., *M-tree: An efficient access method for similarity search in metric spaces*, Vldb, vol. 97, Citeseer, 1997, pp. 426–435.

[17] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky, *Searchable symmetric encryption: improved definitions and efficient constructions*, Proceedings of the 13th ACM conference on Computer and communications security, 2006, pp. 79–88.

[18] Tinankoria Diaby and Babak Bashari Rad, *Cloud computing: a review of the concepts and deployment models*, International Journal of Information Technology and Computer Science **9** (2017), no. 6, 50–58.

[19] Philip M Dixon, L Wu, MP Widrlechner, and ES Wurtele, *Weighted distance measures for metabolomic data*, Bioinformatics (2009), 1–8.

[20] Danny Dolev, Cynthia Dwork, and Moni Naor, *Non-malleable cryptography*, Proceedings of the twenty-third annual ACM symposium on Theory of computing, 1991, pp. 542–552.

[21] CSRC Content Editor, *Cryptography - glossary:csrc*.

[22] Taher ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE transactions on information theory **31** (1985), no. 4, 469–472.

[23] Hongchao Fang, Sicheng Wang, Meng Zhou, Jiayuan Ding, and Pengtao Xie, *Cert: Contrastive self-supervised learning for language understanding*, arXiv preprint arXiv:2005.12766 (2020).

[24] Caroline Fontaine and Fabien Galand, *A survey of homomorphic encryption for nonspecialists*.

[25] Craig Gentry, *Fully homomorphic encryption using ideal lattices*, Proceedings of the forty-first annual ACM symposium on Theory of computing, 2009, pp. 169–178.

[26] Eu-Jin Goh, *Secure indexes*, Cryptology ePrint Archive (2003).

[27] Oded Goldreich, *A uniform-complexity treatment of encryption and zero-knowledge*, Journal of Cryptology **6** (1993), no. 1, 21–53.

[28] Shafi Goldwasser and Silvio Micali, *Probabilistic encryption & how to play mental poker keeping secret all partial information*, Proceedings of the fourteenth annual ACM symposium on Theory of computing, 1982, pp. 365–377.

[29] Shafi Goldwasser and Silvio Micali, *Probabilistic encryption*, Journal of Computer and System Sciences **28** (1984), no. 2, 270–299.

[30] Yunguo Guan, Rongxing Lu, Yandong Zheng, Songnian Zhang, Jun Shao, and Guiyi Wei, *Toward privacy-preserving cybertwin-based spatiotemporal keyword query for its in 6g era*, IEEE Internet of Things Journal **8** (2021), no. 22, 16243–16255.

[31] Yunguo Guan, Yandong Zheng, Rongxing Lu, Jun Shao, and Hui Zhu, *Efficient and privacy-preserving similarity range query over encrypted time series data*, IEEE Transactions on Dependable and Secure Computing **19** (2021), no. 4, 2501–2516.

[32] Cheng Guo, Pengxu Tian, and Chin-Chen Chang, *Privacy preserving weighted similarity search scheme for encrypted data*, IET Information Security **13** (2019), no. 1, 61–69.

[33] Antonin Guttman, *R-trees: A dynamic index structure for spatial searching*, Proceedings of the 1984 ACM SIGMOD international conference on Management of data, 1984, pp. 47–57.

[34] Rohit Handa, C Rama Krishna, and Naveen Aggarwal, *Searchable encryption: a survey on privacy-preserving search schemes on encrypted outsourced data*, Concurrency and Computation: Practice and Experience **31** (2019), no. 17, e5201.

[35] Huan Hu and Jianzhong Li, *Sublinear time nearest neighbor search over generalized weighted manhattan distance*, arXiv preprint arXiv:2104.04902 (2021).

[36] Piotr Indyk and Rajeev Motwani, *Approximate nearest neighbors: towards removing the curse of dimensionality*, Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998, pp. 604–613.

[37] Frances Y Kuo and Ian H Sloan, *Lifting the curse of dimensionality*, Notices of the AMS **52** (2005), no. 11, 1320–1328.

[38] Yutian Lin, Lingxi Xie, Yu Wu, Chenggang Yan, and Qi Tian, *Unsupervised person re-identification via softened similarity learning*, Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 3390–3399.

[39] R Logesh, V Subramaniyaswamy, D Malathi, N Sivaramakrishnan, and Varadarajan Vijayakumar, *Enhancing recommendation stability of collaborative filtering recommender system through bio-inspired clustering ensemble method*, Neural Computing and Applications **32** (2020), 2141–2164.

[40] Rongxing Lu, Yandong Zheng, Yunguo Guan, Jun Shao, and Hui Zhu, *Towards practical and privacy-preserving multi-dimensional range query over cloud*, IEEE Transactions on Dependable and Secure Computing (2021).

[41] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li, *Multi-probe lsh: efficient indexing for high-dimensional similarity search*, Proceedings of the 33rd international conference on Very large data bases, 2007, pp. 950–961.

[42] Hassan Mahdikhani, Rongxing Lu, Yandong Zheng, Jun Shao, and Ali A Ghorbani, *Achieving o ($log^3 n$) communication-efficient privacy-preserving range query in fog-based iot*, IEEE Internet of Things Journal **7** (2020), no. 6, 5220–5232.

[43] P Mell and T Grance, *Special publication 800-145 the nist definition of cloud computing recommendations of the national institute of standards and technology*, URL: https://nvlpubs. nist. gov/nistpubs/Legacy/SP/nistspecialpublication800-145. pdf (2011).

[44] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone, *Handbook of applied cryptography*, CRC press, 2018.

[45] Santiago Ontanon, *An overview of distance and similarity functions for structured data*, Artificial Intelligence Review **53** (2020), no. 7, 5309–5351.

[46] Pascal Paillier, *Public-key cryptosystems based on composite degree residuosity classes*, International conference on the theory and applications of cryptographic techniques, Springer, 1999, pp. 223–238.

[47] Yanguo Peng, Hui Li, Jiangtao Cui, Yixiao Zhu, and Changgen Peng, *An efficient range query model over encrypted outsourced data using secure kd tree*, 2016 International Conference on Networking and Network Applications (NaNA), IEEE, 2016, pp. 250–253.

[48] Braslav Rabar, Maja Zagorscak, Strahil Ristov, Martin Rosenzweig, and Pavle Goldstein, *Igloss: iterative gapless local similarity search*, Bioinformatics **35** (2019), no. 18, 3491–3492.

[49] Sarah Louise Renwick and Keith M Martin, *Practical architectures for deployment of searchable encryption in a cloud environment*, Cryptography **1** (2017), no. 3, 19.

[50] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al., *On data banks and privacy homomorphisms*, Foundations of secure computation **4** (1978), no. 11, 169–180.

[51] Oliver Roesler, *Eeg eye state*, UCI Machine Learning Repository, 2013, DOI: https://doi.org/10.24432/C57G7J.

[52] Kazue Sako, *Semantic security*, pp. 1176–1177, Springer US, Boston, MA, 2011.

[53] Laura Savu, *Cloud computing: Deployment models, delivery models, risks and research challenges*, 2011 International Conference on Computer and Management (CAMAN), IEEE, 2011, pp. 1–4.

[54] Claude E Shannon, *Communication theory of secrecy systems*, The Bell system technical journal **28** (1949), no. 4, 656–715.

[55] Thomas R Shemanske, *Modern cryptography and elliptic curves*, vol. 83, American Mathematical Soc., 2017.

[56] Elaine Shi, John Bethencourt, TH Hubert Chan, Dawn Song, and Adrian Perrig, *Multi-dimensional range query over encrypted data*, 2007 IEEE Symposium on Security and Privacy (SP'07), IEEE, 2007, pp. 350–364.

[57] Pedro Silva, Catarina Maçãs, Evgheni Polisciuc, and Penousal Machado, *Visualisation tool to support fraud detection*, 2021 25th International Conference Information Visualisation (IV), IEEE, 2021, pp. 77–87.

[58] Rajeev Sobti and Ganesan Geetha, *Cryptographic hash functions: a review*, International Journal of Computer Science Issues (IJCSI) **9** (2012), no. 2, 461.

[59] Dawn Xiaoding Song, David Wagner, and Adrian Perrig, *Practical techniques for searches on encrypted data*, Proceeding 2000 IEEE symposium on security and privacy. S&P 2000, IEEE, 2000, pp. 44–55.

[60] Fuyuan Song, Zheng Qin, Liang Xue, Jixin Zhang, Xiaodong Lin, and Xuemin Shen, *Privacy-preserving keyword similarity search over encrypted spatial data in cloud computing*, IEEE Internet of Things Journal **9** (2021), no. 8, 6184–6198.

[61] Henk CA Van Tilborg and Sushil Jajodia, *Encyclopedia of cryptography and security*, Springer Science & Business Media, 2014.

[62] Chang Xu, Ningning Wang, Liehuang Zhu, Chuan Zhang, Kashif Sharif, and Huishu Wu, *Reliable and privacy-preserving top-k disease matching schemes for e-healthcare systems*, IEEE Internet of Things Journal **9** (2021), no. 7, 5537–5547.

[63] Keyu Yang, Xin Ding, Yuanliang Zhang, Lu Chen, Baihua Zheng, and Yunjun Gao, *Distributed similarity queries in metric spaces*, Data Science and Engineering **4** (2019), 93–108.

[64] Pan Yang, Naixue Xiong, and Jingli Ren, *Data security and privacy protection for cloud storage: A survey*, IEEE Access **8** (2020), 131723–131740.

[65] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko, *Similarity search: the metric space approach*, vol. 32, Springer Science & Business Media, 2006.

[66] Yandong Zheng and Rongxing Lu, *Efficient privacy-preserving similarity range query based on pre-computed distances in ehealthcare*, GLOBECOM 2020-2020 IEEE Global Communications Conference, IEEE, 2020, pp. 1–6.

[67] Yandong Zheng, Rongxing Lu, Yunguo Guan, Jun Shao, and Hui Zhu, *Achieving efficient and privacy-preserving exact set similarity search over encrypted data*, IEEE Transactions on Dependable and Secure Computing **19** (2020), no. 2, 1090–1103.

[68] Yandong Zheng, Rongxing Lu, Yunguo Guan, Songnian Zhang, Jun Shao, and Hui Zhu, *Efficient and privacy-preserving similarity query with access control in ehealthcare*, IEEE Transactions on Information Forensics and Security **17** (2022), 880–893.

[69] Yandong Zheng, Rongxing Lu, and Songnian Zhang, *Achieving privacy-preserving weighted similarity range query over outsourced ehealthcare data*, ICC 2022-IEEE International Conference on Communications, IEEE, 2022, pp. 1251–1256.

[70] Yandong Zheng, Rongxing Lu, Songnian Zhang, Hui Zhu, and Fengwei Wang, *Efficiency-improved privacy-preserving weighted similarity query over outsourced ehealthcare data*, GLOBECOM 2022-2022 IEEE Global Communications Conference, IEEE, 2022, pp. 4866–4871.

Appendix A

Correctness of SHE

A.1 Correctness of Homomorphic Properties

Proof of correctness is as shown below for our homomorphic addition(I & II) and multiplication (I & II) protocols.

A.1.1 The Correctness of Homomorphic Addition-I

Given pp and two ciphertexts:

$E(m_1) = (r_1 L + m_1)(1 + r'_{1p})\, modN$ and

$E(m_2) = (r_2 L + m_2)(1 + r'_{2p})\, modN$, we have

$E(m_1) + E(m_2) modN \rightarrow E(m_1 + m_2)$

Proof:

$E(m_1) + E(m_2)\, modN$

$= (r1\mathcal{L} + m_1)(1 + r'_{1p}) + (r_2\mathcal{L} + m_2)(1 + r'_{2p})\, modN$

$= ((r_1 + r_2)\mathcal{L} + m_1 + m_2 + \alpha p)\, modN$

where $\alpha = (r_1 r'1 + r_2 r'2)\mathcal{L} + (m_1 r'1 + m_2 r'2)$

When we try to decrypt $E(m_1) + E(m_2)\, modN$, we have:

$SHEDec(sk, E(m_1) + E(m_2))$

$= ((((r_1 + r_2)\mathcal{L} + m_1 + m_2 + \alpha p)\ mod N)\ mod p)\ mod \mathcal{L}$

$= ((r_1 + r_2)\mathcal{L} + m_1 + m_2)\ mod \mathcal{L} (2k_2 < k_0)$

$= m_1 + m_2 (k_1 \ll k_2)$

We decrypt it to $m_1 + m_2$

Therefore,

$E(m_1) + E(m_2)\ mod N \rightarrow E(m_1 + m_2)$

A.1.2 The Correctness of Homomorphic Multiplication-I

Given pp and two ciphertexts:

$E(m_1) = (r_1\mathcal{L} + m_1)(1 + r'_2 p)\ mod N$ and

$E(m_2) = (r_2\mathcal{L} + m_2)(1 + r'_2 p)\ mod N$,

we apply the homomorphic multiplication:

$E(m_1 * m_2)\ mod N$

$= E(m_1) * E(m_2)\ mod N$

$= (r_1\mathcal{L} + m_1)(1 + r'_1 p)(r_2\mathcal{L} + m_1)(1 + r'_2 p)\ mod N$

$= (m_1 * m_2 + r_1 r_2 \mathcal{L}^2 + r_1\mathcal{L} m_2 + r_2\mathcal{L} m_1 + \Delta * p)\ mod N$

α denotes the coefficient of p

We decrypt the ciphertext of $E(m_1 * m_2)$:

$((E(m_1 * m_2)\ mod N)\ mod p)\ mod \mathcal{L})$

$= ((((m_1 * m_2 + r_1 r_2 \mathcal{L}^2 + r_1\mathcal{L} m_2 + r_2\mathcal{L} m_1 + \Delta * p)\ mod N)\ mod p)\ mod \mathcal{L})$

$= ((((m_1 * m_2 + r_1 r_2 \mathcal{L}^2 + r_1 L m_2 + r_2 L m_1)\ mod N)\ mod p)\ mod \mathcal{L})$

To decipher the ciphertext correctly, $m_1 * m_2 + r_1 r_2 \mathcal{L}2 + r_1\mathcal{L} m_2 + r_2\mathcal{L} m_1) < p$.

According to $m_1, m_2 \in \{0,1\}k_1, r_1, r_2, \mathcal{L} \in \{0,1\}k_2$ and $k_2 \gg k_1$, we can deduct that $r_1 r_2 \mathcal{L}^2$ is the largest noise term.

To guarantee the correctness of decryption, we are supposed to ensure $r_1 r_2 \mathcal{L}^2 < p$.

We set up the maximum depth of homomorphic multiplication-I, denoted as $\theta = [\frac{k_0}{2k_2} - 1]$.

If the multiplicative depth is within θ, we have:

$((((m_1 * m_2 + r_1 r_2 \mathcal{L}^2 + r_1 \mathcal{L} m_2 + r_2 \mathcal{L} m_1)\ mod N)\ mod p)\ mod \mathcal{L})$

$= ((r_1 r_2 \mathcal{L} + r_1 m_2 + r_2 m_1)\mathcal{L} + m_1 * m_2) mod \mathcal{L}$

$= m_1 * m_2 (k_1 \ll k_2)$

Homomorphic multiplication-I is established if the multiplicative depth is involved within θ.

A.1.3 The Correctness of Homomorphic Addition-II

Given pp and a ciphertext $E(m_1) = (r_1 \mathcal{L} + m_1)(1 + r_1' p) mod N$ and a message m_2, we have $E(m_1) + m_2 mod N \rightarrow E(m_1 + m_2)$

Proof:

$E(m_1) + m_2 mod N$

$= (r_1 \mathcal{L} + m_1)(1 + r_1' p) + m_2 mod N$

$= (r_1 \mathcal{L} + m_1 + m_2 + (r_1 \mathcal{L} + m_1) r_1' p) mod N$

We decrypt $E(m_1) + m_2 mod N$:

$SHEDec(sk, E(m_1) + m_2)$

$= (((r_1 \mathcal{L} + m_1 + m_2 + (r_1 \mathcal{L} + m_1) r_1' p)\ mod N)\ mod p)\ mod \mathcal{L}$

$= (r_1 \mathcal{L} + m_1 + m_2) mod \mathcal{L} (2k_2 < k_0)$

$= m_1 + m_2 (k_1 \ll k_2)$

We decrypt it to $m_1 + m_2$

Therefore, $E(m_1) + m_2 mod N \rightarrow E(m_1 + m_2)$

A.1.4 The Correctness of Homomorphic Multiplication-II

Given pp and a ciphertext $E(m_1) = (r_1 \mathcal{L} + m_1)(1 + r_1' p)\ mod N$ and a message $m_2 (m_2 > 0)$, we have $E(m_1) * m_2 mod N \rightarrow E(m_1 * m_2)$.

Proof:

$E(m_1) * m_2 mod N = (r_1\mathcal{L} + m_1)(1 + r'_1 p) * m_2 mod N$

$= (r_1 m_2 \mathcal{L} + m_1 * m_2 + (r_1\mathcal{L} + m_1) r'_1 p m_2) mod N$

We decrypt $E(m_1) * m_2 \, mod N$, we have:

$SHEDec(sk, E(m_1) + m_2)$

$= (((r_1 m_2 \mathcal{L} + m_1 * m_2 + (r_1\mathcal{L} + m_1) r'_1 p m_2) \, mod N) \, mod p) \, mod \mathcal{L}$

$= (r_1 m_2 \mathcal{L} + m_1 * m_2) \, mod \mathcal{L} (2k_2 < k_0)$

$= m_1 * m_2 (k_1 \ll k_2)$

We decrypt it to $m_1 * m_2$.

Therefore, $E(m_1) * m_2 mod N \rightarrow E(m_1 + m_2)$ if $m_2 > 0$.

A.2 Correctness of Decryption

When given a ciphertext $E(m) = (r\mathcal{L} + m)(1 + r'p) \, mod N$, it can be decrypted to plaintext m with a secret key $sk \leftarrow (p, \mathcal{L})$.

Proof:

$SHEDec(sk, E(m)) = (E(m) \, mod p) \, mod \mathcal{L}$

$= (((r\mathcal{L} + m)(1 + r'p) \, mod N) \, mod p) \, mod \mathcal{L}$

$= (r\mathcal{L} + m) \, mod \mathcal{L} (2k_2 < k_0)$

$= m(k_1 \ll k_2)$

Appendix B

Java Implementation of Core Modules

This section shows snippets of code implementing part of our scheme.

B.1 Implementation of CloudServer1

The CloudServer1 class implements a cloud server that stores our encrypted kd-tree and works together with CloudServer2 to implement our privacy protocols and compute the results of our scheme.

Listing B.1: Java Implementation CloudServer1 Class

```
public class CloudServer1 {

  private SHEPublicParameter pp;
  private BigInteger E_minus_1;
  private EncKdTree encKdTree;
  private CloudServer2 cs2;
  private final SecureRandom rnd = new SecureRandom();
```

```
public CloudServer1(SHEPublicParameter pp,
BigInteger E_minus_1,
EncKdTree kdTree, CloudServer2 cs2) {
  this.pp = pp;
  this.E_minus_1 = E_minus_1;
  this.encKdTree = kdTree;
  this.cs2 = cs2;
}
public SHEPublicParameter getPp() {
  return pp;
}
public BigInteger getE_minus_1() {
  return E_minus_1;
}
public int compare(BigInteger left, BigInteger right) {
  SecureRandom rnd = new SecureRandom();
  boolean b = rnd.nextBoolean();
  if (b) {
    BigInteger temp = left;
    left = right;
    right = temp;
  }

  BigInteger m = left.add(right.multiply(E_minus_1)).mod(pp.
     getN());
  BigInteger[] r1r2 = generateRandomR1R2(pp.getK1());
  BigInteger r1 = r1r2[0];
  BigInteger r2 = r1r2[1];
```

```
    BigInteger E_x = r1.multiply(m).add(r2).mod(pp.getN());
    int sign_m = cs2.compare(E_x);
    if (b) {
      return -sign_m;
    }
    return sign_m;
  }
  public BigInteger compare2(BigInteger left, BigInteger right
     ) {
    SecureRandom rnd = new SecureRandom();
    boolean b = rnd.nextBoolean();
    if (b) {
      BigInteger temp = left;
      left = right;
      right = temp;
    }

    BigInteger m = left.add(right.multiply(E_minus_1)).mod(pp.
       getN());
    BigInteger[] r1r2 = generateRandomR1R2(pp.getK1());
    BigInteger r1 = r1r2[0];
    BigInteger r2 = r1r2[1];
    BigInteger E_x = r1.multiply(m).add(r2).mod(pp.getN());
    BigInteger sign_m = cs2.compare2(E_x);
    if (b) {
      return BigInteger.ONE.add(E_minus_1.multiply(sign_m)).
         mod(pp.getN());
    }
    return sign_m;
```

```
}

public static BigInteger[] generateRandomR1R2(int k1) {
  SecureRandom rnd = new SecureRandom();
  BigInteger r1, r2;
  do {
    r1 = new BigInteger(k1, rnd);
    r2 = new BigInteger(k1, rnd);
  } while (!(r1.compareTo(r2) > 0 && r2.compareTo(BigInteger
     .ZERO) > 0));
  return new BigInteger[]{r1, r2};
}

public BigInteger absVal(BigInteger left, BigInteger right)
   {
  SecureRandom rnd = new SecureRandom();
  boolean b = rnd.nextBoolean();
  if (b) {
    BigInteger temp = left;
    left = right;
    right = temp;
  }

  BigInteger m = left.add(right.multiply(E_minus_1)).mod(pp.
     getN());
  BigInteger[] r1r2 = generateRandomR1R2(pp.getK1());
  BigInteger r1 = r1r2[0];
  BigInteger r2 = r1r2[1];
  BigInteger E_x = r1.multiply(m).add(r2).mod(pp.getN());
```

```
    BigInteger absVal = cs2.absVal(E_x);
    return absVal.multiply(m).mod(pp.getN());
}

public List<ArrayList<BigInteger>> Filtration(EncQueryToken
    encQueryToken, EncKdNode node) {
    List<ArrayList<BigInteger>> candidateList = new ArrayList
        <>();
    if (node instanceof EncKdLeaf) {

        candidateList.addAll(((EncKdLeaf) node).encVectorList.
            values());

    } else if (node instanceof EncKdInterNode) {
        EncKdInterNode interNode = (EncKdInterNode) node;
        BigInteger[] cd = interNode.cipherCd;

        BigInteger upperBound = BigInteger.ZERO;
        BigInteger lowerBound = BigInteger.ZERO;

        EncQueryRange encQueryRange = encQueryToken.
            getEncQueryRange();
        BigInteger[] upperBounds = encQueryRange.getUpperBounds
            ();
        BigInteger[] lowerBounds = encQueryRange.getLowerBounds
            ();
        for (int i = 0; i < cd.length; i++) {
```

```
        upperBound = upperBound.add(upperBounds[i].multiply(cd
            [i])).mod(pp.getN());
        lowerBound = lowerBound.add(lowerBounds[i].multiply(cd
            [i])).mod(pp.getN());
      }
      if (compare(interNode.cipherCv, lowerBound) > 0) {
        candidateList.addAll(Filtration(encQueryToken,
            interNode.left));
      }
      if (compare(interNode.cipherCv, upperBound) < 0) {
        candidateList.addAll(Filtration(encQueryToken,
            interNode.right));
      }
    }
    return candidateList;
  }

  public List<BigInteger[]> Verification(EncQueryToken
     encQueryToken, List<BigInteger[]> candidateList) {

    List<BigInteger[]> VerifiedRecords = new ArrayList<>();
    for (BigInteger[] p : candidateList) {
      BigInteger dist = calEncWeightedManhattanDist(p,
          encQueryToken.getEncQ(), encQueryToken.getEncW());
      BigInteger E_bi = compare2(encQueryToken.getEncTau(),
          dist) ;
      BigInteger [] P_prime = new BigInteger[p.length];
      for (int i = 0; i < p.length; i++) {
        P_prime[i] = E_bi.multiply(p[i]).mod(pp.getN());
```

```
    }
    VerifiedRecords.add(P_prime);
  }
  return VerifiedRecords;
}

private BigInteger calEncWeightedManhattanDist(BigInteger[]
   p, BigInteger[] q, BigInteger[] w) {
  BigInteger dist = BigInteger.ZERO;
  for (int i = 0; i < w.length; i++) {
    BigInteger d_i = absVal(p[i], q[i]).multiply(w[i]).mod(
       pp.getN());
    dist = dist.add(d_i).mod(pp.getN());
  }
  return dist;
}

public Pair<List <BigInteger[]>, List<byte[]>> randomize(
   List<BigInteger []> VerifiedRecords, SecretKey aesKey) {
  int size = VerifiedRecords.size();
  List<BigInteger []> C_Prime2 = new ArrayList<>(size);
  List<BigInteger []> random_vectors = new ArrayList<>(size)
     ;
  for (int i = 0; i < size; i++) {
    BigInteger [] p = VerifiedRecords.get(i);
    BigInteger [] r_vector = new BigInteger[p.length];
    BigInteger [] P_prime = new BigInteger[p.length];

    for (int j = 0; j < p.length; j++) {
```

```
        r_vector[j]= new BigInteger(pp.getK1(), rnd);
        P_prime [j] = r_vector[j].add(p[j]).mod(pp.getN());
      }
      C_Prime2.add(P_prime);
      random_vectors.add(r_vector);
    }
    return new Pair<>(random_vectors,cs2.reencryption(C_Prime2
       , aesKey));
  }

}
```

B.2 Implementation of CloudServer2

The CloudServer2 class implements a cloud server that stores our private key (sk) and works together with CloudServer1 to implement our privacy protocols and compute the results of our scheme.

Listing B.2: Java Implementation CloudServer2 Class

```
public class CloudServer2 {

  private final SHEPrivateKey sk;
  public CloudServer2(SHEPrivateKey sk) {
    if (sk == null) {
      throw new IllegalArgumentException("Private␣key␣cannot
         ␣be␣null");
    }
    this.sk = sk;
```

```
}
public SHEPrivateKey getSk() {
  return sk;
}
public int compare(BigInteger e_x) {
  BigInteger result = SymHomEnc.dec(e_x, sk);
  if (result.compareTo(sk.getL().shiftRight(1)) > 0) {
    return -1;
  }
  return 1;
}
public BigInteger compare2(BigInteger e_x) {
  BigInteger result = SymHomEnc.dec(e_x, sk);
  if (result.compareTo(sk.getL().shiftRight(1)) > 0) {
    return SymHomEnc.enc(1, sk);
  }
  return SymHomEnc.enc(0, sk);
}

public BigInteger absVal(BigInteger e_x) {
  BigInteger ZERO = SymHomEnc.enc(0, sk);
  BigInteger result = SymHomEnc.dec(e_x, sk);
  if (result.compareTo(sk.getL().shiftRight(1)) > 0) {
    return ZERO.add(sk.getL()).subtract(BigInteger.ONE);
  }
  return ZERO.add(BigInteger.ONE);
}
```

```
public List<byte[]> reencryption(List<BigInteger[]>
   c_prime2, SecretKey aesKey) {
  ArrayList<byte []> C_AES = new ArrayList<>();

  for (int i = 0; i < c_prime2.size(); i++) {
    BigInteger[] Ep_prime = c_prime2.get(i);
    BigInteger [] p_prime = new BigInteger[Ep_prime.length
       ];

    for (int j = 0; j < Ep_prime.length; j++) {
      p_prime[j] = SymHomEnc.dec(Ep_prime[j],sk);
    }
    byte [] barray = new byte[0];
    try ( ByteArrayOutputStream baos = new
       ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(baos))
       {

      oos.writeObject(p_prime);
      barray = baos.toByteArray();

    } catch (IOException e) {
      e.printStackTrace();
    }

    byte[] cipher = AES.encrypt(barray,aesKey);
    C_AES.add(cipher);
  }
```

```
        return C_AES;

    }
  }
```

B.3 Implementation of EncQueryToken

The EncQueryToken class defines the parameters and methods that are used to generate an encrypted query token.

Listing B.3: Java Implementation EncQueryToken Class

```
public class EncQueryToken {

  private BigInteger[] EncQ;
  private BigInteger[] EncW;
  private BigInteger EncTau;
  private EncQueryRange encQueryRange;
  private static final int AES_KEY_SIZE = 128;
  private SecretKey ssk;
 public EncQueryToken(BigInteger[] encQ, BigInteger[] encW,
    BigInteger encTau, EncQueryRange encQueryRange) {

  EncQ = encQ;
  EncW = encW;
  EncTau = encTau;
  this.encQueryRange = encQueryRange;
  this.ssk = secretKey(AES_KEY_SIZE);
}
```

```
private SecretKey secretKey(int keySize) {
  return AES.getRandomKey(keySize);
}

public static EncQueryToken genEncQueryToken(QueryToken
   queryToken, SHEPublicKey pk){
  BigInteger[] encQ = new BigInteger[queryToken.getDimension()
     ];
  for (int i = 0; i < encQ.length; i++){
    encQ[i] = SymHomEnc.enc(queryToken.getQ()[i],pk);
  }

  BigInteger[] encW = new BigInteger[queryToken.getDimension()
     ];
  for (int i = 0; i < encW.length; i++) {
    encW[i] = SymHomEnc.enc(queryToken.getW()[i], pk);
  }

  BigInteger encTau;
  encTau = SymHomEnc.enc(queryToken.getTau(), pk);

  EncQueryRange encQueryRange =  new EncQueryRange(queryToken.
     getQueryRange(),pk);
  return new EncQueryToken(encQ,encW, encTau, encQueryRange);
}

public BigInteger[] getEncQ() {
  return EncQ;
```

```
}

public BigInteger[] getEncW() {
  return EncW;
}

public BigInteger getEncTau() {
  return EncTau;
}

public EncQueryRange getEncQueryRange() {
  return encQueryRange;
}

public SecretKey getSsk() {
  return ssk;
}
}
```

Vita

Candidate's Full Name: Rhoda Tani Mairabo
Bachelor of Engineering, Ahmadu Bello University Zaria, 2003.
Bachelor of Science, Belarusian State University of Informatics and Radioelectronics, 2018.