Efficient and Privacy-Preserving Similarity Range Query over Genomic Sequences

by

Jiacheng Jin

Bachelor of Computer Science, University of New Brunswick, 2020

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(s):	Rongxing Lu, Ph.D., Faculty of Computer Science
Examining Board:	Saqib Hakak, Ph.D., Faculty of Computer Science, Chair
	Sajjad Dadkhah, Ph.D., Faculty of Computer Science
	Zhen Lei, Ph.D., Department of Civil Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

April, 2022

Abstract

Personalized medicine is becoming more common and accepted with the development of the economy and the improvement of living standards. Meanwhile, similarity queries, one of the trending topics for researchers, have attracted much attention. A subsection of the trending topic, similarity query over genomic sequences, has played a significant role in personalized medicine and has applications in various fields, including DNA alignment and genomic sequencing. Since handling genomic sequences requires massive storage and considerable computational capacity, service providers prefer to process similarity queries over genomic sequences with outsourced datasets on cloud servers. Furthermore, since genomic sequences are highly sensitive data, preserving the privacy of queries has attracted considerable attention. Although many schemes have been proposed for similarity queries over encrypted genomic data, they are either inefficient or have limitations in supporting the dynamic update of the dataset. To address the challenges, we propose an efficient and privacy-preserving similarity range query scheme. Specifically, we introduce an algorithm to build a hash table to index the dataset and present a similarity range query algorithm based on the hash table. Then, we design two cloud-based privacy-preserving protocols based on homomorphic encryption to support the similarity range query algorithm over the encrypted dataset. After that, we propose the privacy-preserving similarity range query scheme by leveraging the two privacy-preserving protocols. We then analyze the security of our proposed scheme and prove that our scheme is privacy-

preserving. Finally, we perform experiments to evaluate the scheme's performance, and the results indicate that it is computationally efficient.

Dedication

To my family and teachers without whom, I would have achieved nothing.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Dr. Rongxing Lu, for his support, guidance and advice. Without his patient guidance, enthusiasm and immense knowledge, I would not have been able to complete my journey as a student at the University of New Brunswick.

I would like to thank my thesis advisory committee members for patiently reading through my thesis and providing valuable comments and suggestions.

My sincere appreciation goes to Dr. Yandong Zheng and Dr. Pulei Xiong for their precious comments, careful reversion and valuable support to my research study.

I would also express my love and gratefulness to my family members for all their continuous support during my master's study.

My thankfulness also goes to my labmates, Dr. Yandong Zheng, Mr. Yunguo Guan, Mr. Songnian Zhang for their guidance. I would also thank all my fellow in the same research group under the supervision of Dr. Lu and my friends Ms. Ellen Zhang, Ms. Tanda Qi and Ms. XianXing Deng for their support.

Table of Contents

List of Tables

List of Figures

Abbreviations

DNA	Deoxyribonucleic Acid
SNP	Single Nucleotide Polymorphisms
AWAS	Genome-Wide Association Study
KNN	k-Nearest Neighbours
NIST	National Institute of Standards
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
AES	Advanced Encryption Standard
DES	Data Encryption Standard
RSA	Rivest–Shamir–Adleman Cryptosystem
ECC	Elliptic Curve Cryptography
DSA	Digital Signature algorithm
BGN	Boneh, Goh, and Nissim Cryptosystem

Chapter 1

Introduction

We now live in a society where customized services are becoming more popular and well-spread. Personalized medicine is a precise and supportable healthcare service for patients, including services like disease prediction and prevention based on individuals, and has attracted the public's interest. The unique property of human beings, genomic sequences, can perfectly support the service.

Researchers have worked on human beings' genomic sequences for years and have notable findings. For example, the Genome-Wide Association Study (AWAS) is a type of research targeting the relationship between genomic sequences and the risk of having diseases. The study finds that specific mutations in BRCA1 and BRCA2 correlate closely with a high risk of breast cancer [44, 3]. Other research applications on genomic instructions in personalized medicine are Paternity Test and DNA alignment [10, 14, 7, 15]. Nowadays, research on genomic sequences even supports authentic business on personalized medicine. 23andMe [1] announced a relative finder service to find a person's relative based on the person's genomic sequences.

With the sensitivity of the genomic sequences, preserving the privacy of the genomic sequences has attracted considerable attention. However, handling genomic sequences usually requires massive storage and strong computation capability. Due to

[1] https://www.23andme.com

the high requirement, personalized medicine service providers outsource the dataset to cloud servers to provide their services at lower rates. Although cloud servers can reduce storage requirements, preventing security risks like abuse or revealing data becomes a priority [55, 18]. Therefore, preserving the privacy of genomic sequences while providing personalized medicine services becomes a priority.

1.1 Genetic Background

In this section, we recall genetic background relates to genomic instructions, including Deoxyribonucleic Acid (DNA), genome, DNA sequencing, and genomic sequences.

1.1.1 Deoxyribonucleic Acid (DNA)

DNA is a basic chemical compound that contains genomic instructions. Fig. 1.1 demonstrates the structure of DNA.

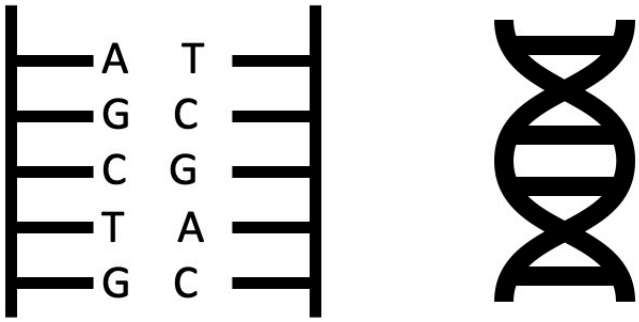

Figure 1.1: Structure of DNA

The right pictorial of Fig. 1.1 is an icon that demonstrates the structure of DNA molecules. From the pictorial, we know that DNA molecules have a double helix structure containing a pair of twisted DNA strands. A DNA strand is the basic

physical and functional unit of heredity. For human beings, DNA strands contain four types of different nucleotides (also known as nitrogen bases or nucleobases) which are two purines (adenine (A) and guanine (G)) and two pyrimidines (thymine (T) and cytosine (C)). Researchers prefer to use sequences containing these four letters (i.e., A, C, G, T) to represent DNA strands. The order of nucleotides in DNA strands encodes genomic instructions in DNA molecules, which lead to the uniqueness between organisms and directs the activities of organisms. In paired DNA strands, an adenine (A) is always paired with a thymine (T), while a cytosine (C) is always paired with a guanine (G) [19]. The left pictorial of Fig. 1.1 shows an example of a piece of paired DNA strands.

A complete set of genomic instructions of human beings contains approximately three billion nucleotides [19]. If we place these nucleotides end-to-end in pairs of DNA strands, these nucleotides can reach a length of six feet [20]. Therefore, the number of nucleotides on human beings' DNA seems to be significant and the number of different orders of nucleotides might be extremely large. However, each person has only 0.5% difference from the reference genome [3, 42]. The common variants in DNA are called single nucleotide polymorphisms (SNP) and are important remarks for research over genomic instructions. The variants inherit from parents to children may lead to inheritable disease. By checking variants in DNA, we can perform disease prediction services and mitigate the disease at an earlier phase.

1.1.2 Genome

A complete genome contains all genetic information of an organism. It consists of DNA strands and can be represented as a nucleotide sequence. Using the genome, we can know every individual's past and present in biology. For example, we can know where an individual's ancestors come from and where they have been from instructions contained in the genome.

Genomes are contained in cells and can be found from various sample types like hair, skin, blood, and even saliva [3]. The structures that contain genomic structures in cells are called chromosomes. The term chromosomes comes from 'chroma' and 'soma' in Greek, meaning colour and body. Chromosomes are particular types of 'body' is packed with proteins and DNA and can be dyed during research.

Each organism has a different number of chromosomes. For example, a dog has 39 chromosomes in cells, while a fruit fly has eight chromosomes. For human beings, we have 23 pairs of chromosomes which are 46 chromosomes in cells.

We human beings inherit a copy of a chromosome from the male parent (i.e., biological father) and another from the female parent (i.e., biological mother). That is why children look similar to their biological parents on appearance in some points. In the 46 chromosomes, the first 22 pairs of chromosomes are called autosomes. The last pair of chromosomes are called sex chromosomes. This pair of chromosomes is a special pair of chromosomes that differ between males and females. A male has a copy of X-chromosome from his female parent and a copy of Y-chromosome from his male parent, while a female has a copy of X-chromosome each from her male parent and female parent, respectively.

Individuals may get serious problems when they do not have exact 46 chromosomes in their cells. For example, women with one X chromosome only have Turner syndrome. They are shorter than others and may have kidney or heart problems [20].

Since individuals inherit genomes from their biological parents, we can determine if an individual is the biological parent of another individual and perform paternity tests by checking the similarity of chromosomes between two individuals. Meanwhile, if two individuals share similar genomic instructions in significant areas of their genome, they will be highly likely to be each other's relatives. As a result, genomes can support the relative finding service in personalized medicine.

1.1.3 DNA Sequencing

DNA sequencing is the fundamental basis of all genomic research. It is a technique to determine the exact order of nucleotides in strands of DNA [19]. To define the orders of nucleotides in an individual's genome, we need a sample from the person. The sample can be hair, skin, blood or even saliva that contains cells and genomes. The most common sequencing type is sequencing by synthesis, and the most common sequencing platform is Illumina Sequencers [19, 3]. The sequencing platform first outputs a sequence of nucleotides with lengths between 25 and 500 from a randomly partitioned genome. To manually get sequences, researchers will use enzymes to emit light sources of nucleotides. After getting orders of nucleotides in different sequences, researchers then analyze the overlaps of these sequences and assemble them into a more extended sequence of nucleotides. Repeating the assembling step, researchers can determine the order of nucleotides at any length. In order to increase the accuracy of the output sequence of nucleotides, each nucleotide will be read multiple times.

1.1.4 Genomic Sequences

Although the genome can perfectly support personalized medicine on disease prediction and paternity tests, there is a limitation on using the genome. Since a complete genome is too big to extract and analyze, it is challenging to use whole genomes in every personalized medicine service. For example, a nearly complete genome of a human being on Genbank [2] can be as large as 254.3 MB. When the number of genomes to handle grows, genomes may bring big challenges to data management, including data storage and data usage. Luckily, not every genomic instruction contained in the genome is essential in personalized medicine services like disease prediction or finding relatives. For example, using SNP or significant areas in the genome for typical

[2]https://www.ncbi.nlm.nih.gov/nuccore/AP023461.1

types of diseases instead of the whole set of the genome is a good solution for disease prediction. Genomic sequences, the shorter version of the complete genome, can be the sequence of nucleotides in some specific area in the whole set of the genome. They are suitable substitutions in these cases in personalized medicine.

Genomic sequences are sequences of nucleotides and are briefer versions of a complete genome. Therefore, handling the same number of genomic sequences requires less storage and better performance than complete genomes. More research relates to genomes like similar sequence queries. DNA alignment prefers to work over genomic sequences rather than the complete genome to achieve better performance and less cost on physical equipment for data management and computation.

1.2 Similarity Queries

Similarity query, a currently popular focus of research, has attracted much attention [58]. It is defined by a query object q and a constant of the form of the extent of proximity required, usually distance. The query result of similarity queries is all objects that satisfy conditions, regarded as objects that are close to the given object [52]. Similarity queries have applications on various fields, including eHealthcare [56], computer vison [28], and smart city [51]. In this section, we recall similarity range query and nearest neighbour query, which are two elementary types of similarity query.

1.2.1 Similarity Range Query

The Similarity range query is a common type of similarity query. It is specified by a query object q with a threshold value τ, which is the distance constraint. The similarity range query finds all objects in a dataset S that the distance $d(\cdot, \cdot)$ between the object and the query object q is within the distance τ, formally [52]:

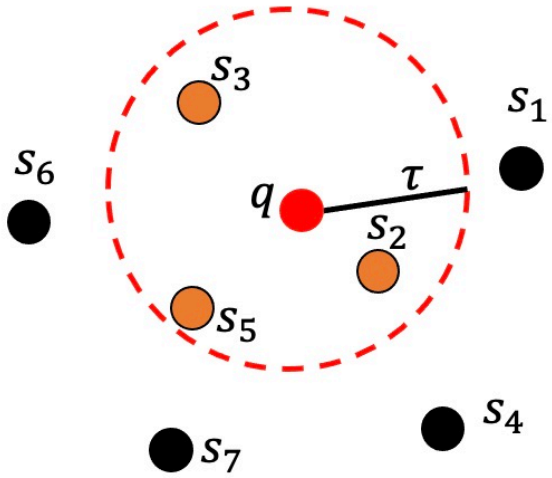

Figure 1.2: Example of similarity range query $R(q, \tau)$

$$R(q, \tau) = \{s_i | s_i \in S; d(s_i, q) \leq \tau\}.$$

Objects in the query result can be sorted by distance to the query object q if required. For example, we can search for all gasoline stations within 500 meters of my current location in geographic applications.

Fig. 1.2 shows an example.The example shows three objects whose distance to the query object q is less than or equal to the threshold value τ. Therefore, the set of the three objects $\{s_2, s_3, s_5\}$ is the query result to the similarity range query request (q, τ) in this example.

When the distance constraint $\tau = 0$, this particular range query of $R(q, 0)$ is called an exact match or point query. In the point query, we search for the copy(copies) of the query object q. The point query is commonly used to delete a specific object from a dataset.

The similarity range query is performed based on knowledge of the dataset because we have to specify the distance constraint when performing the query. If the distance constraint is too small, we may get a query result that contains nothing (i.e., an empty set). Meanwhile, we may get a query result containing non-necessary objects

if the distance constraint is too large.

1.2.2 Nearest Neighbour Query

The nearest neighbour query is another type of similarity query. The nearest neighbour query can be generalized as the KNN query ($KNN(q)$), the response set of which is the k nearest neighbours of the object q in the distance $d(\cdot,\cdot)$ from in a dataset S. The response set of the k-nearest query ($KNN(q)$) is formally defined as follows [52]:

$$KNN(q) = \{X \subseteq S, |X| = k \wedge \forall r_i \in X, r_j \in S - X : d(q, r_i) \leq d(q, r_j)\}$$

When multiple objects in the response set have the same distance to the query object q as the kth closest neighbour, they will be picked randomly. For example, we can search for the two closest gasoline stations to my current location in geographic applications.

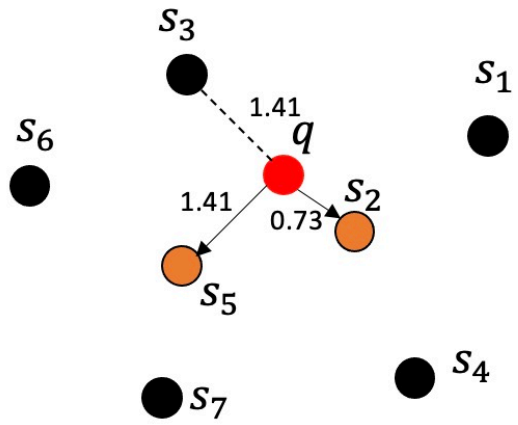

Figure 1.3: Example of 2-nearest neighbour query $2NN(q)$

Fig. 1.3 shows an example. This figure shows that object s_2 has the closest distance to the query object q of 0.73. Meanwhile, the object s_5 and s_3 have the same distance

to q, and the object s_5 is randomly picked. Therefore, $\{s_2, s_5\}$ is the query result to this example's 2-nearest neighbour query $2NN(q)$.

The nearest neighbour query has a special case that the response result will contain less than k objects. If we want to perform a KNN query on a dataset S that contains n objects where $n \leq q$, the query result will be the whole dataset S.

1.3 Cloud Computing

Although using genomic sequences can mitigate the pressure on storage and computation ability, handling genomic sequences still requires considerable storage and strong computation capability during data blooming. Due to the high requirement, personalized medicine service providers prefer to use cloud computing to mitigate the problem. In this section, we introduce the definition and basic features of cloud computing. We also discuss data privacy in the cloud computing environment.

Definition 1 (Cloud computing)**.** *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [4].*

Cloud computing is the most popular and common solution for managing blooming data nowadays. It has been considered a new and essential technique in computation that can provide better computational services at flexible and cheaper rates. Compared to traditional IT that purchases infrastructures, cloud computing provides a more flexible and convenient way to run a business. National Institute of Standards (NIST) defines cloud computing as shown in Definition 1.

Table 1.1: Features of cloud computing

Essential characteristics	Broad network access Rapid elasticity On-demand self-service Measured service Resource pooling
Deployment models	Private cloud Community cloud Public cloud Hybrid cloud
Service models	Software as a Service (SaaS) Platform as a Service (PaaS) Infrastructure as a Service (IaaS)

1.3.1 Basic Features of Cloud Computing

Based on Definition 1, NIST further defines cloud computing to have five essential characteristics, four deployment models, and three service models. All these features are shown in TABLE 1.1 and explained as follows:

• *Essential characteristics:* The five essential characteristics of cloud computing allow cloud providers to abstract and aggregate resources into a pool allocated to different users. Meanwhile, it also allows users to configure resources on-demand and manage them without iterating with service providers.

The board network access allows users to access cloud computing in any situation with the network and provides the convenience of service. The rapid elasticity allows cloud computing to follow users' requirements more closely. The on-demand self-service allows users to decide how many services they want and change services by preference. It provides more flexibility when using cloud computing services than traditional IT services. The measured service characteristic of cloud computing requires user less financial cost than traditional IT. All the services provided by cloud providers can be measured and sometimes charged users by their usage. The flexible service makes cloud computing more reachable to the public and more affordable to use.

With resource pooling, users can access data simply through the internet at any place, but they do not know where they are stored [40]. Users submit data to the data center, and these data are stored distributed in worldwide data centers [23].

The five characteristics also bring convenience to service providers. Compared to traditional IT in which the data center is mandatory, cloud computing makes the management of data centers more standardized, centralized and professional. Service providers can decide how and where to set up their data center to reduce their cost on data center than traditional IT. Meanwhile, the reachable service cost can attract more users to choose cloud computing. The growth of customer groups can bring more profit to service providers.

Therefore, service providers can earn more while service users can have a flexible, convenient and affordable service with cloud computing.

• *Deployment models:* There are four deployment models of the cloud. The public cloud provides services to the public. The private cloud provides services to one company or organization only for internal usage. The community cloud provides services to multiple companies or organizations simultaneously. The hybrid cloud is a combination of two or more types of cloud.

The deployment model is defined based on the cloud user and the organization that owns or manages the cloud. It can be different even within a single deployment model.

• *Service models:* Service models on the cloud vary, and the three most famous models defined by NIST are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS), as shown in Fig. 1.4. In SaaS, the whole bundle of applications is provided as a service. Users can run or access applications on the cloud through web browsers or mobile applications [4]. Development platforms, including middleware and servers, are provided in the PaaS model. Users can develop their projects directly instead of managing the underlying server, net-

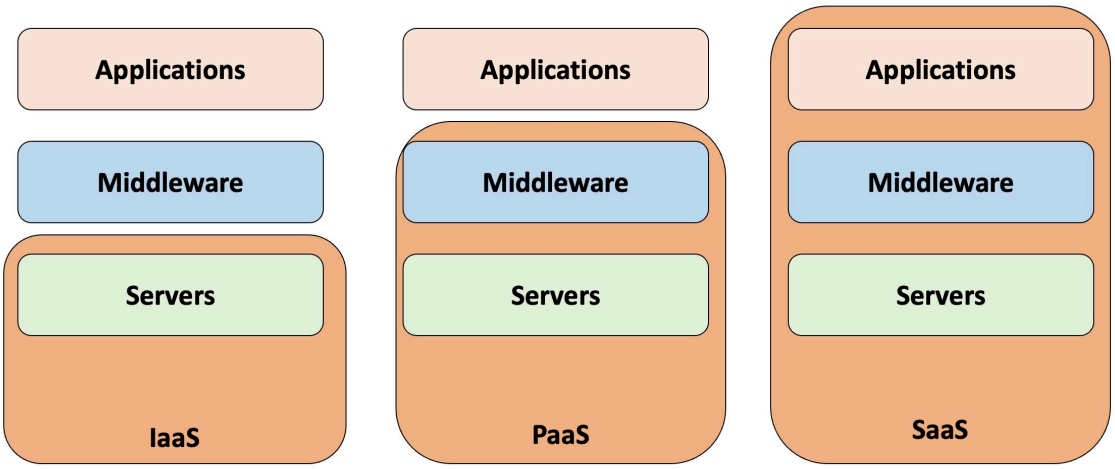

Figure 1.4: Service models in cloud computing

work, or other infrastructure by themselves. The IaaS model has only fundamental computing resources provided, such as computing, networking, or storage, to help users manage arbitrary software. This model needs less initial cost but has great flexibility.

1.3.2 Data Privacy in Cloud Computing

Although cloud computing can benefit companies with money, some unique characteristics may bring problems like data privacy. Compared to data in traditional IT, data on the cloud is more vulnerable [23]. More data is processed through the internet, and data are stored in a data center far away. Therefore, privacy risks transfer from users to the cloud provider. Unlike traditional data management solutions, cloud providers will be responsible for privacy issues when using the cloud framework [23, 26]. The shared infrastructure, internet usage and data stored by cloud providers made the cloud model based on the internet emphasize more attention on privacy [23, 13].

The definition of privacy is the combination of confidentiality, integrity and availability, ensuring completeness, leak prevention and usability of data. Data privacy

is a key implementation tool for information and data governance.

• *Data confidentiality:* Data confidentiality in the cloud refers to authorized users' access to data. In cloud computing models, users usually need to trust cloud servers for their services [40, 23]. However, cloud servers are always considered as honest-but-curious because cloud servers are provided by a third party. Therefore, avoiding data being accessed by unauthorized users and the cloud servers is a challenge for cloud computing.

• *Data integrity:* Data integrity in the cloud relates to data transfer and data storage. Due to the distributed structure of cloud computing, data travels more than expected. More transmission for data between cloud servers means more possibility of data being monitored or accessed by hackers. Data needs to keep not changed after transferring or stored between cloud servers. Therefore, achieving and verifying data integrity is complex and challenging in a cloud environment.

• *Data availability:* Data availability means that authorized users can access data at any time. It could be affected by many factors, like a network connection. Generally, data availability in the cloud is better than traditional IT because the downtime can be handled better with big suppliers than small companies [23].

1.4 Research Problem

Similarity query over genomic sequences is a hot topic, and there are a lot of state-of-art schemes designed based on either the secure multiparty computation model or the secure outsourcing model [44], which is explained more detailedly in Section 2.3. This thesis focuses on the secure outsourcing model-based similarity range query over genomic sequences. Due to secure outsourcing model-based scheme leveraging the computational ability and storge provided by cloud computing solutions, privacy-

preserving the data in the schemes becomes a priority. Since the cloud servers in the scheme are regarded as honest-but-curious, we trust cloud servers to handle data integrity and availability and focus on data confidentiality in our scheme.

Therefore, data in the schemes contains the following two parts considering data privacy:

• *Outsourced dataset:* The outsourced dataset is deposited at cloud servers, and the dataset might contain sensitive personal information like genomic information. Therefore, the outsourced dataset should be secure.

Cloud servers are usually considered honest-but-curious. They will follow the instructions and process the dataset. However, in most cases, the cloud servers may be curious about the information contained in the dataset. In this case, any dataset leakage needs attention. Therefore, performing similarity range queries on cloud servers may grab the concerns of data owners.

• *Query request:* The query request is the requesting query object and the threshold value sent to the cloud servers. With multiple times of requesting and processing, the cloud servers may have enough information to guess the pattern and information of the query request. As cloud servers are usually considered honest-but-curious, they may get some information during the query process. Therefore, not protecting query requests will raise more privacy concerns for query users.

Encryption techniques are a good solution to handle these two concerns. However, encryption techniques bring more calculation to the query processing step, leading to worse performance than not using encryption techniques. However, users prefer more efficient schemes. Efficiency, as a result, is also a part that needs attention.

In summary, this thesis studies how to perform an efficient and privacy-preserving similarity range query scheme over genomic sequences, which has the following two objectives in detail:

• *Privacy preservation:* The primary requirement of the proposed scheme is privacy-preservation. The scheme should preserve the privacy of both outsourced data and query requests.

• *Efficiency:* Another requirement of the proposed scheme is efficiency. We aim to minimize the computational cost of processing similarity range queries on cloud servers.

1.5 Main Contribution

As mentioned in Section 1.4, state-of-art schemes are designed based on either the secure multiparty computation model or the secure outsourcing model [44].

The secure multiparty computation model-based schemes require multiple rounds of communication between different parties and have a relatively large communication overhead.

The secure outsourcing model-based schemes outsource encrypted datasets to cloud servers and delegate the cloud servers to offer the similarity query, which is communication efficient. However, they are either inefficient or have limitations to support dynamic updates of the dataset. Specifically, some schemes [44, 37, 50] have a linear query efficiency because the queries are processed by traversing all sequences in the dataset. As a result, they are inefficient. Although some schemes [10, 30, 45] achieve sublinear query efficiency by building indexes for the dataset, they either cannot support dynamic data updates or require multiple rounds of communications between data owners and cloud servers to finish data updates. Therefore, achieving similarity queries with a sublinear search efficiency is still challenging while supporting the dynamic update of data.

Targeting the challenges, in this thesis, we propose an efficient and privacy-preserving similarity range query scheme over genomic data based on Zhang et al.'s [54] scheme.

We consider outsourcing the dataset and achieving similarity range queries using a two-server model to reduce the computation pressure and storage requirement. The key idea of our proposed scheme is to build a hash table to index the dataset and then use it to achieve efficient similarity range queries. After that, we design two privacy-preserving protocols to preserve the privacy of hash table-based similarity queries.

Specifically, the main contributions of this thesis can be summarized as follows:

1. Based on the Paillier encryption technique [34], a type of homomorphic encryption, we design two privacy-preserving protocols, minimum value computation protocol and equality test protocol, to support edit distance computation between two cloud servers, where edit distance is regarded as the similarity metric of our proposed scheme.

2. We present our scheme on similarity queries over genomic sequences under a two-server model by applying the two privacy-preserving protocols and a key-based hash function to preserve the privacy of hash table-based similarity queries. The proposed scheme can achieve efficient similarity queries and support dynamic updates of the dataset.

3. We analyze the security of our proposed scheme and prove that our proposed scheme secures the plaintext of the dataset and query sequences.

4. We conduct experiments to evaluate the performance of our proposed scheme. The results show that our proposed scheme is efficient.

1.6 Thesis Organization

The remainder of this thesis is organized as follows. We present common techniques and related works of similarity range queries over genomic sequences in Chapter 2.

In Chapter 3, we review some preliminaries. In Chapter 4, we introduce our system model, security model and design goals. We describe our scheme in Chapter 5. In Chapter 6, we analyze the security of our scheme, followed by a performance evaluation in Chapter 7. Finally, we conclude the thesis and discuss some future works in Chapter 8.

Chapter 2

Literature Review

In this chapter, we recall privacy-preserving techniques in state-of-art schemes. After that, we categorize state-of-art schemes and discuss open questions.

2.1 Cryptography Techniques

In this section, we review the cryptography techniques. Specifically, we first include cryptography's goal and basic concepts and then introduce symmetric encryption, asymmetric encryption, and homomorphic encryption.

2.1.1 Goals of Cryptography

Cryptography is the most commonly used technique to preserve the privacy of information with some specific algorithms during communication [39, 36, 17]. The goals of cryptography are as follows [1]:

• *Authentication:* Authentication ensures that every person involved in the system is from an authorized person.

• *Confidentiality:* Confidentiality implies that both the sender and receiver have to grant access to data before sending or receiving information from any system.

• *Data Integrity:* Only the authorized parties have access to modulate the database that belongs to a specific group or person.

• *Non-Repudiation:* Both the sender and the receiver of a sent message cannot deny the delivery of a message. Non-Repudiation ensures that both the sender and receiver must agree on the delivery of messages.

• *Access Control:* Access control implies that only those with correct authentication can access corresponding messages.

2.1.2 Basic Concepts of Cryptography

The basic concepts of cryptography include plaintext, ciphertext, encryption, decryption and key. Fig. 2.1 shows an example of cryptography techniques.

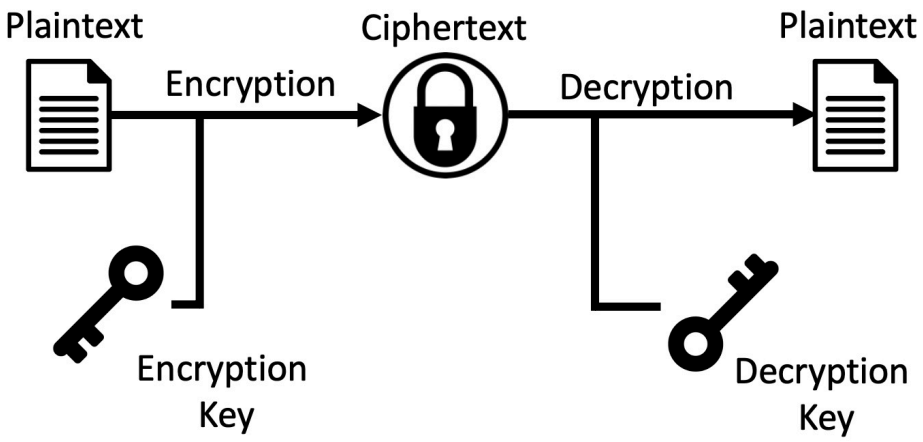

Figure 2.1: Example of cryptography technique

• *Plaintext:* The human-readable message (i.e., a string of characters) that an individual wants to send to another individual is called plaintext. For example, Alice, an individual, wants to send "Hello" to Bob. The message "Hello" in this example is a plaintext.

• *Ciphertext:* The ciphertext is a massage that does not make sense or is unreadable for human beings. In encryption techniques, plaintexts are converted to ciphertexts

before being sent to another individual. For example, Alice, an individual, wants to send a message to Bob. Alice first converts the message to a string of characters that others cannot comprehend before sending it to Bob. The message "@qe2$HK&EI" sent to Bob is a ciphertext.

• *Encryption:* The encryption process uses an algorithm to convert plaintexts to ciphertexts. The algorithm takes plaintext and a key as input, and the algorithm's output is a ciphertext. The goal of the encryption process is disguising plaintext by the message sender in the communication through insecure channels. The encryption process occurs at the sender's side before sending messages. For example, Alice, an individual, wants to send a message to Bob. Instead of sending the message, she transforms it into "@qe2$HK&EI" and sends it to Bob. Alice, in this example, is the sender and the process of transforming the message to "@qe2$HK&EI" is the encryption process. The algorithm and the key used in the encryption process are the encryption algorithm and the encryption key, respectively.

• *Decryption:* The decryption process uses an algorithm to transform plaintexts from ciphertexts. Decryption is the reverse process of encryption. The algorithm used in the decryption process takes a ciphertext and a key as input, and the algorithm's output is a plaintext. The decryption process occurs at the receiver's side after receiving messages. For example, Bob receives a message "@qe2$HK&EI" from Alice. Bob then transforms "@qe2$HK&EI" into a human-readable message. Bob, in this example, is the receiver and the process of transforming "@qe2$HK&EI" into a human-readable message is the decryption process. The algorithm and the key used in the decryption process are the decryption algorithm and the decryption key, respectively.

• *Key(s):* A key is a string of characters. Key is used as part of input in encryption or decryption algorithms. The key-choosing procedure decides the security of encryption techniques. Besides the encryption key and decryption key, there are other

categories of the key. Based on the access condition of keys, we have the secret key and the public key. If a key is open to everyone, it is a public key. Meanwhile, if only the owner only knows a key is a secret key.

Encryption and decryption algorithms in cryptography techniques can use the same or different keys during calculation. We can generally categorize the encryption technique into symmetric and asymmetric encryption based on the number of keys used in encryption and decryption algorithms. The difference between symmetric encryption and asymmetric encryption is summarized in TABLE 2.1.

2.1.3 Symmetric Encryption

Symmetric encryption, also known as single-key encryption, is a type of cryptography technique that only uses one key in the encryption algorithm and the decryption algorithm in symmetric encryption. The key used in symmetric is called secret key or shared secret and is represented by sk. The structure of symmetric encryption is shown in Fig. 2.2.

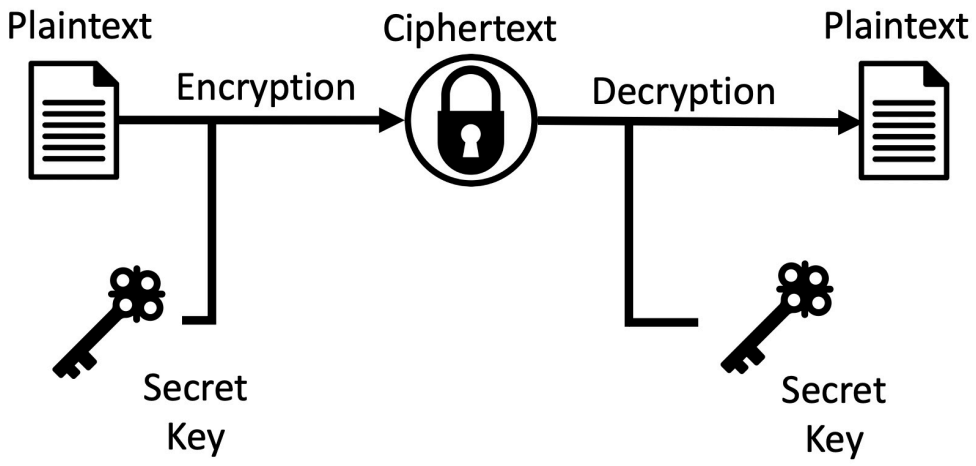

Figure 2.2: Structure of symmetric encryption

In symmetric encryption, if a sender wants to send a message to a receiver, they have

to share the same piece of information (i.e., secret key) before actually sending the message. When sending the message, the sender first transforms an intended text into ciphertext and sends the ciphertext to the receiver. After the receiver gets the ciphertext, the receiver uses the same key to retrieve the plaintext.

2.1.3.1 Requirements of Symmetric Encryption

We have two requirements to use symmetric encryption securely: requirement on key management and encryption algorithm [24].

• *Requirement on key management:* The sender and the receiver have to exchange a key securely and keep the key away from others. If a third person knows the key and the encryption algorithm, every message involved in the communication that uses the key is readable to the person.

• *Requirement on the underlying algorithm:* We need a robust algorithm to support symmetric encryption. Specifically, we want the encryption algorithm to ensure that if an adversary knows one or more ciphertexts and the algorithm, the adversary cannot retrieve plaintext or determine the key.

2.1.3.2 Advantages and Drawbacks of Symmetric Encryption

The advantage of the symmetric encryption technique is that it can complete the message transfer process in a short time and is used very often [17]. Using only one key brings advantages and attention to symmetric encryption. It also brings some issues simultaneously.

The same key used in the process requires the sender and receiver in the structure have to agree and exchange the key ahead of the process. To preserve the privacy of messages, we share with different people and prevent unauthorized individuals from retrieving messages, we need to share a different key with every single different individual we want to exchange messages. Therefore, symmetric encryption techniques

have high requirements for managing and preserving the privacy of a large amount different keys for each individual.

2.1.3.3 Classifications of Symmetric Encryption

The symmetric encryption technique contains block cipher and stream cipher, as shown in Fig. 2.3 and Fig. 2.4. There are many algorithms of symmetric encryption, and some examples are AES, DES, RC4, and Snow 2.0 [17, 38, 21, 1].

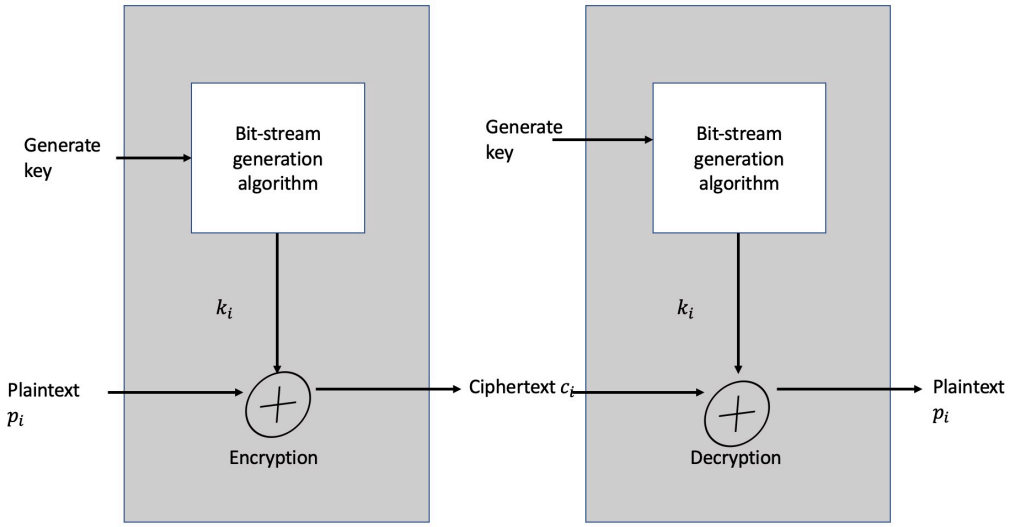

Figure 2.3: Example of stream cipher using algorithmic bit-stream generator

Stream cipher: A stream cipher is a cryptography technique that encrypts an intended message by one bit or one byte at a time. As a type of symmetric encryption, the corresponding secret key used in stream cipher must be exchanged between sender and receiver ahead of message exchange. If the key is large in size, problems will occur. Stream cipher may use a bit-stream generator to mitigate this problem if the key is a random long string. Fig. 2.3 shows an example of a stream cipher using an algorithmic bit-stream generator. The usage of the bit-stream generator can reduce the size of the secret key that requires to exchange ahead. In this case, the sender and receiver only need to exchange the generated key.

Block cipher: A block cipher is a cryptography technique that encrypts plaintext

blocks into ciphertext blocks of equal length. Usually, the blocks of plaintexts are 64 or 128 bits [24]. In block cipher, the sender and the receiver share the same key. Block cipher can achieve the same effect but is less efficient than stream cipher [17, 24]. However, block cipher seems applicable to more applications. For example, many cryptographic applications relates to network use block ciphers [24].

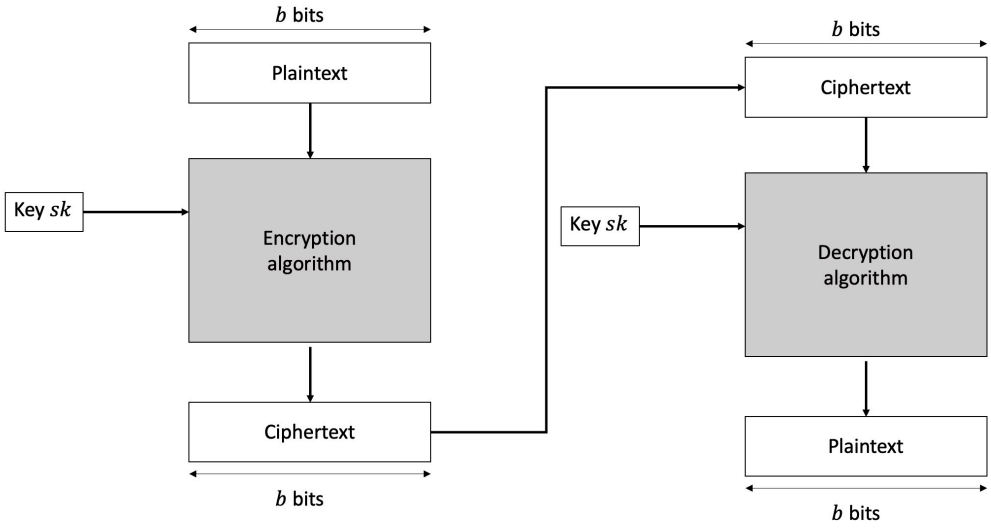

Figure 2.4: Example of block cipher

2.1.4 Asymmetric Encryption

Asymmetric encryption, also known as public-key encryption, is a type of cryptography technique that uses a pair of keys (i.e., the public key and secret key) throughout the process. There are a lot of algorithms of symmetric encryption, and some examples are RSA, Diffie-Hellman algorithm, ElGamal encryption, ECC, DSA [17, 1]. The structure of asymmetric encryption is shown in Fig. 2.5.

In asymmetric encryption, if a sender wants to send a message to a receiver, the receiver first defines a public key and a secret key and then announces the public key to everyone. After getting the public key, the sender then encrypts the plaintext into ciphertext. The sender then sends the ciphertext to the receiver. After the receiver

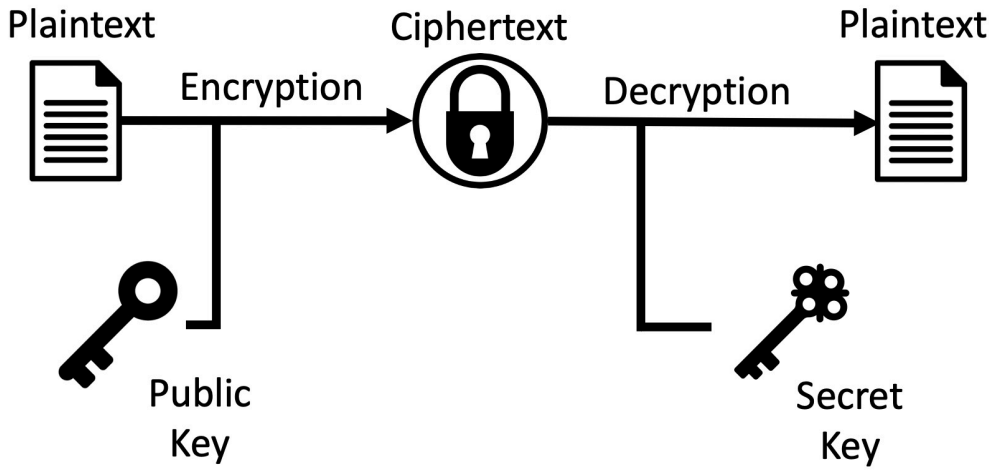

Figure 2.5: Structure of asymmetric encryption

gets the ciphertext, the receiver uses its secret key to retrieve the plaintext. Since only one public-secret key pair is used in this process, the receiver and the sender does not need to exchange key or reach some agreement ahead. Therefore, anyone can get the receiver's public key and use the key to send a message to the receiver.

2.1.4.1 Requirements of Asymmetric Encryption

We have two requirements to use asymmetric encryption securely: requirement on key management and the underlying algorithms [24].

• *Requirement on key management:* The sender and receiver must securely have either one in the key pair and keep the key away from others. Meanwhile, the sender and the receiver should not keep the same key in the key pair.

• *Requirement on underlying algorithms:* We need a robust algorithm to support asymmetric encryption. Specifically, we want the encryption algorithm to ensure that if an adversary knows some ciphertexts, one key in the key pair and the encryption algorithm, the adversary cannot determine the other key. Meanwhile, the algorithm should also ensure that decrypting a plaintext is impossible if one key in the key pair is unknown.

2.1.4.2 Advantages and Drawbacks of Asymmetric Encryption

Only one pair of keys are required to receive messages from others in asymmetric encryption. Compared to symmetric encryption, which needs multiple keys for each person, key management in asymmetric encryption is easier. However, asymmetric encryption most uses more powerful keys to provide similar protection to symmetric encryption. Meanwhile, asymmetric encryption is based on more complex mathematical computation. Therefore, symmetric encryption is more used in encrypting longer messages than asymmetric encryption [1]. All these leads to the results that asymmetric encryption is less efficient and more expensive than symmetric encryption [17, 1].

Table 2.1: Comparision of symmetric encryption and asymmetric encryption

Symmetric Encryption	**Asymmetric Encryption**
Same key for both encryption and decryption algorithms.	Different keys for encryption and decryption algorithms.
1. If an adversary knows one or more ciphertexts and the algorithm, the adversary cannot retrieve plaintext or determine the key.	1. If an adversary knows some ciphertexts, one key in the key pair and the encryption algorithm, the adversary cannot determine the other key. 2. Decrypting a plaintext is impossible if one key in the key pair is unknown.
It is more used in encrypting longer messages than asymmetric encryption.	It is slower as it needs more time processing with a more powerful key.
Example technique: AES, DES, RC4 and Snow 2.0, etc.	Example technique: RSA, Diffie-Hellman algorithm, ElGamal encryption, ECC, DSA, etc.

Besides symmetric encryption and asymmetric encryption, cryptography techniques also includes homomorphic encryption [34].

2.1.5 Homomorphic Encryption

Homomorphic encryption is a particular type of cryptography technique that has homomorphic properties. Homomorphic encryption ensures no data leakage when

operating on ciphertexts, whose corresponding plaintext is unknown. Therefore, homomorphic encryption is the most commonly used technique to preserve data privacy outsourced to a third party (e.g., cloud servers). Fig. 2.6 shows an example of homomorphic encryption.

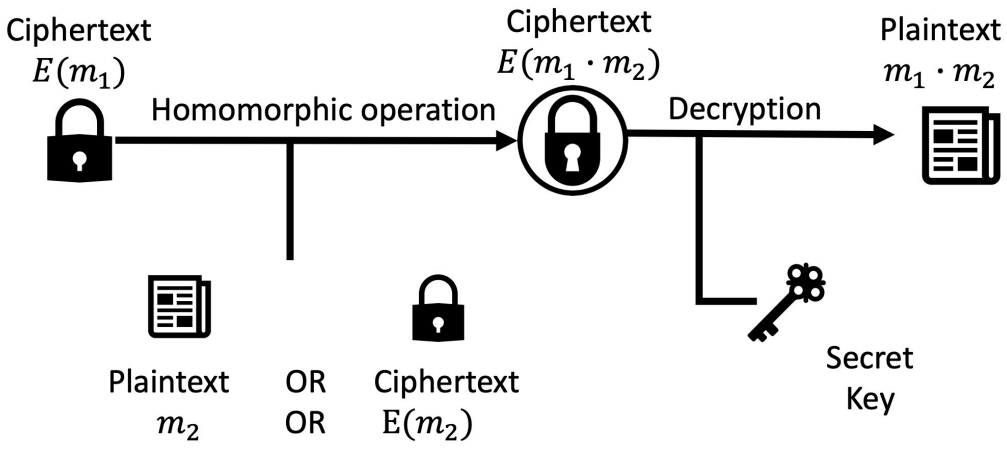

Figure 2.6: Example of homomorphic encryption

2.1.5.1 Homomorphic Properties

The term "homomorphic property" orgins from the notion of a group homomorphism which is shown in Definition 2 [39]. The homomorphic property supports operations on the encrypted text to achieve the same result as the corresponding operation on plantexts. For example, let m_1 and m_2 be two messages (plaintexts) and $E(m_1)$ and $E(m_2)$ are their ciphertexts. Homomorphic property means that the underlying algorithm of a cryptography technique satisfies the condition $E(m_1) \star E(m_2) = E(m_1 \cdot m_2)$.

Definition 2 (Homomorphism)**.** *Suppose G is anabelian group with operaton $\cdot$ and H is a group with group operation $\star$. A homomorphism from $(G, \cdot)$ to $(H, \star)$ is a mapping $f : G \rightarrow H$ that satisfies the condition $f(x_1) \star f(x_2) = f(x_1 \cdot x_2)$ for all $x_1, x_2 \in G$.*

2.1.5.2 Classifications of Homomorphic Encryption

Based on different operations for homomorphic operation and the number of times the technique can be used, homomorphic techniques can be categorized into partially homomorphic encryption, somewhat homomorphic encryption and fully homomorphic encryption, as summarized in TABLE 2.2.

Table 2.2: Comparision of homomorphic encryption schemes

Catogory	**Homomorphic Operation**	**Number of Homomorphic Operations**
Partially homomorphic encryption	Multiplication **OR** Addition	Arbitrary
Somewhat homomorphic encryption	Multiplication **AND** Addition	Limited
Fully homomorphic encryption	Multiplication **AND** Addition	Arbitrary

• Partially homomorphic encryption considers the homomorphic operation of only multiplication or addition and supports an arbitrary number of homomorphic operations. Paillier homomorphic encryption [34] is a commonly used partially homomorphic encryption technique that only supports the homomorphic operation of addition.

• Somewhat homomorphic encryption has more flexibility on computation and can support the homomorphic operation of both multiplication and addition. However, somewhat homomorphic encryption only supports a limited number of homomorphic operations before it cannot correctly decrypt messages. An example of somewhat homomorphic encryption is BGN [12].

• Fully homomorphic encryption supports an arbitrary number of homomorphic operations. It also supports homomorphic operations of both multiplication and addition.

2.2 Hash Function

In this section, we recall hash functions. Detailedly, we first discuss the general operation and properties of the hash function. Then we recall the requirements of the key-based hash function. We finally discuss applications of the hash function, i.e., message authentication and digital signature.

2.2.1 General Operation of Hash Function

The hash function is a function that maps a large string of characters into a smaller digest or hash value [9]. Fig. 2.7 gives an example of a hash function. Given a message m_1 and a hash function $H(\cdot)$, we can get a digest (hash value) of the message m_1 as $H(m_1)$.

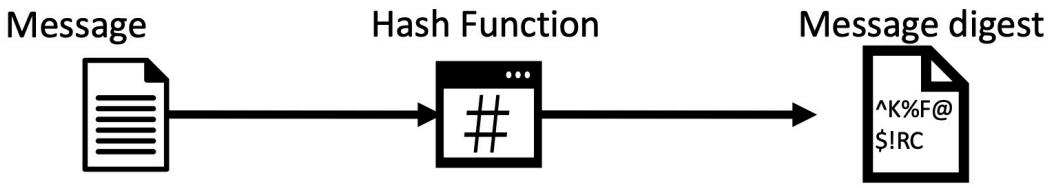

Figure 2.7: Example of a hash function

2.2.2 Properties of Hash Function

The following properties guarantee the security of the hash function [39, 24]:

• *One-wayness/Preimage resistance:* Given a message and a hash function, we can compute the hashed value (also known as digest). Meanwhile, it is impossible to compute the original message with the hashed message and the hash function.

• *Second preimage resistance:* Given a message m_1 and a hash function $H(\cdot)$, it is unlikely to find another message m_2 that $H(m_1) = H(m_2)$.

• *Collision resistance:* Given a hash function $H(\cdot)$, it is unlikely to find $H(m_1) = H(m_2)$ where m_1 and m_2 are two messages.

• *Variable input size:* A hash function can be applied to a message of any size.

• *Fixed output size:* A hash function produces a hash value of a fixed size.

• *Efficiency:* Given a message and a hash function, it is easy to compute the hashed value.

• *Pseudorandomness:* A hash function outputs a hash value that satisfies standard tests for pseudorandomness.

The first requirements are standard requirements of the hash function. The requirement of the size of input and output and efficiency are practical requirements for the hash function, and the last requirement is not traditionally listed as a requirement but has been implied [24].

The one-wayness property of the hash function ensures the computational impossibility of retrieving a message with a hash function and a hash value. Therefore, the hash function can resist cryptography analysis and has limitations on reverse engineering. As a result, the hash function is a commonly used technique for error detection, signature authentication and protecting data integrity and can be used in similarity queries.

2.2.3 Key-Based Hash Function

Key-based hash function, also known as message authentication code (MAC), is a typical technique that combines hash functions and symmetric encryption and is designed to reduce the computation pressure on handling data that does not have confidential requirements. The key-based hash function is used among two people that share a secret key. By comparing the hash value of messages, the two people can verify messages exchanged between them.

Definition 3 (Key-based Hash Function)**.** *A function $h_K(\cdot)$ that maps a message m_1 to a hash value $h_K(m_1)$ is a key-based hash function, if it satisfies the following properties:*

1. The description of $h_K(\cdot)$ is publicly known.

2. Given a key K and a message m_1, it is easy to compute the digest $h_K(m_1)$.

3. Without knowledge of K, it is hard to find m_1 when $h_K(m_1)$ is given, or find two distinct messages m_1 and m_2 such that $h_K(m_1) = h_K(m_2)$ when k is not publicly known.

4. Given message-digest pairs, it is hard to find the key K.

5. Without knowledge of K, it is hard to determine $h_K(m_1)$ for any message m_1, even when a large set of message-digest pairs, where $(m_1, h_K(m_1))$ pair is not given.

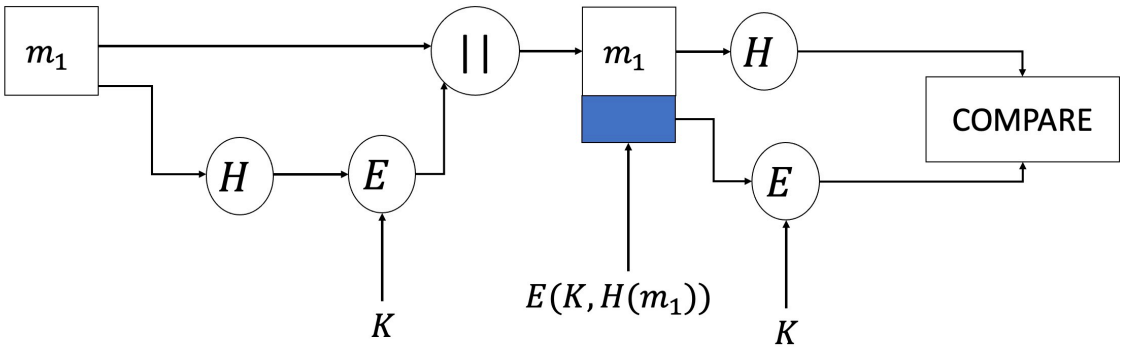

Figure 2.8: Example of key-based hash function

Based on the Definition 3 [9], the key-based hash function $h_K(\cdot)$ takes a message m_1 and a key K as the input. The key K used in the key-based hash function is a secret key.

Like symmetric encryption, the hash function is secure unless the key K is disclosed to a third person except for the receiver and the sender. Fig. 2.8 shows an example of a key-based hash function, and the process is as follow:

• ***Step 1:*** The sender first get a hash value of the message m_1 with hash function then use the secret key K to encrypt the hash value and gets $E(K, H(m_1))$.

• ***Step 2:*** The sender then sends $E(K, H(m_1))$ and the message m_1 to the receiver.

• ***Step 3:*** The receiver first decrypts the encrypted hash value with the shared secret key sk then verifies if the incoming hash value $H(m_1)$ equals to the calcuated

hash value $H(m_1')$.

i) If $H(m_1) = H(m_1')$, then the receiver can confirm that the incoming message is not being modified, inserted, deleted, or replayed (i.e., $m_1' = m_1$).

ii) If there is a mismatch, then either the message m_1 or hash value $H(m_1)$ is being changed.

As a type of hash function, the key-based hash function has three following requirements [39]:

• Given a message m_1, a key-based hash function $h_K(\cdot)$ and a hash value $h_K(m_1)$, it is computationally impossible to find a m_2 that $h_K(m_1) = h_K(m_2)$

• $h_K(m_1)$ is uniformly distributed. Given two randomly choosed message m_1 and m_2 and a key-based hash function $h_K(\cdot)$, the probablity of $h_K(m_1) = h_K(m_2)$ is 2^{-n}, where is the number of bit in the hash value.

• Given a mapping of m_1 to m_2 (i.e., $m_2 = f(m_1)$) and a key-based hash function $h_K(\cdot)$, $Pr[h_K(m_1) = h_K(m_2)] = 2^{-n}$.

2.2.4 Message Authentication

Message authentication is a mechanism for verifying the integrity of messages. The mechanism has a requirement that the sender must have a valid identity and can achieve with the key-based hash function, which is described in Section 2.2.3. The hash value in message authentication must be transmitted securely to ensure the hash value remains unchanged even if the message.

Fig. 2.9 shows an example of message authentication, and the process of message authentication is the following three steps:

• ***Step 1:*** The sender uses the bits in the message m_1 and a hash function $H(\cdot)$ to compute a hash value $H(m_1)$. Then the sender tansmits the hash value along with the message m_1.

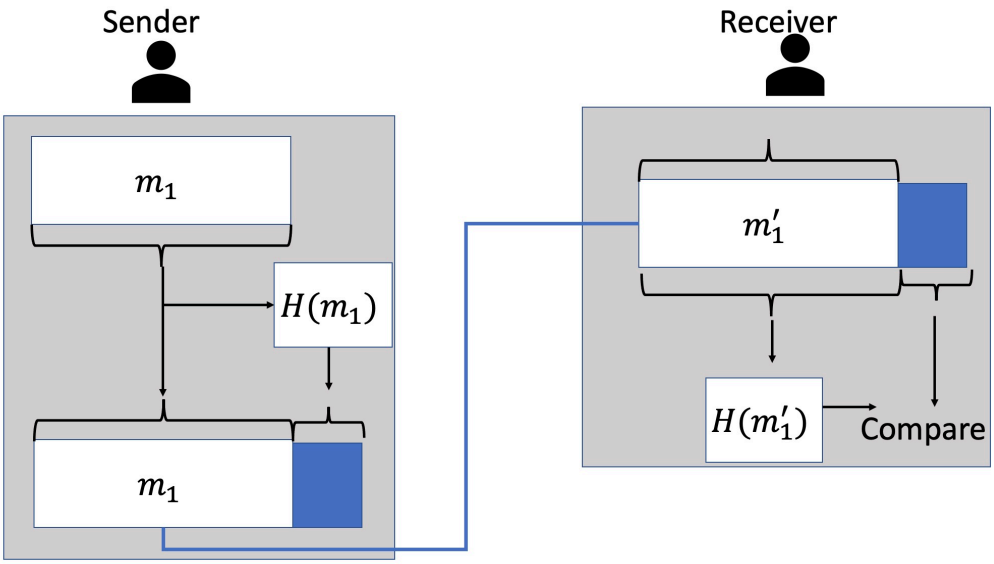

Figure 2.9: Example of message authentication

• ***Step 2:*** The receiver uses the same hash function $H(\cdot)$ and the received message m_1' to get a hash value $H(m_1')$.

• ***Step 3:*** The receiver verifies if the incoming hash value $H(m_1)$ equals to the calculated hash value $H(m_1')$.

i) If $H(m_1) = H(m_1')$, then the receiver can confirm that the incoming message is not being modified, inserted, deleted, or replayed (i.e., $m_1' = m_1$).

ii) If there is a mismatch, then either the message m_1 or hash value $H(m_1)$ is being changed.

2.2.5 Digital Signature

Similar to the key-based hash function, which is described in Section 2.2.3, the digital signature is another application of the hash function.

The only difference between a key-based hash function and a digital signature is that the digital signature is a combination of hash functions and asymmetric encryption. In the scenario of digital signature, whoever holds the secret key sk hashes a message and concatenates the hash value with the message into a signature. Anyone who

knows the corresponding public key pk can verify the signature. Fig. 2.10 shows an example of message authentication.

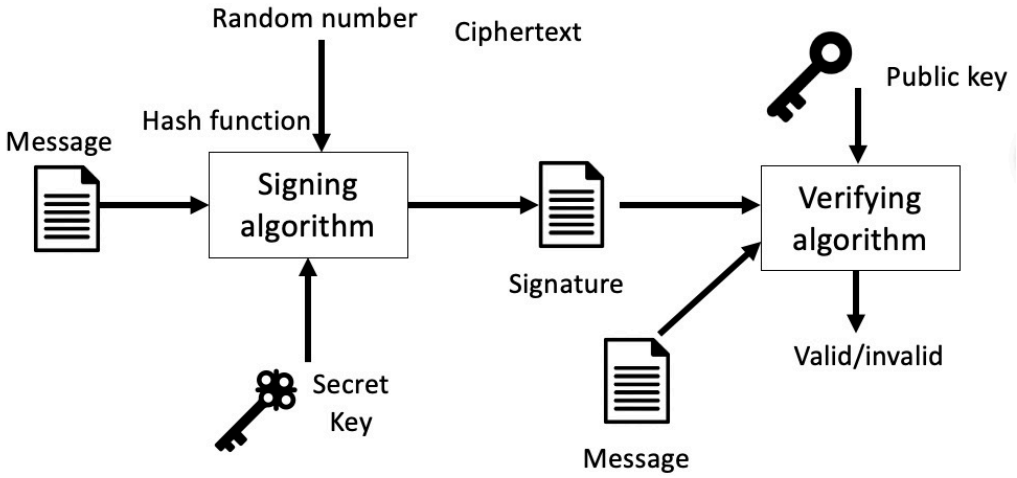

Figure 2.10: Example of digital signature

2.3 Similarity Queries over Genomic Sequences

As mentioned in Chapter 1.4, state-of-art schemes of similarity queries over genomic sequences are designed either based on a secure multiparty computation model or a secure outsourcing model. In this section, we discuss state-of-art models based on these two different models, respectively.

2.3.1 Secure Multiparty Computation Model-Based Schemes

The secure multiparty computation model contains multiple parties which work together to process different functions, and the particular cryptography model was first proposed by Yao [49] in 1986. The basic workflow of the secure multiparty computation model-based scheme is the query processing step. In this step, all parties in the system cooperate to get similar query results securely.

The model's primary focus is to ensure the computation process is secure, trusted,

and correct. The secure multiparty computation model accepts all kinds of input which makes classifying inputs essential, and it can handle both intentional and unintentional errors in the system. Therefore, the secure multiparty computation is widely used in secret sharing and is usually treated as a trusted third party or sequential composition in different systems.

Garbled Circuit [49] is the most common-used protocol in the secure multiparty computation model-based schemes. The Garbled Circuit is a particular type of multi-round protocol. It can be regarded as a Boolean circuit that uses oblivious transfer to exchange sensitive information (e.g., genomic instructions) between the two parties in the protocol.

2.3.1.1 Related Work

Atallah and Li [6] proposed one of the initial works in the similar sequence query. The scheme iteratively uses Garbled Circuit [49] to exchange information for computing edit distance between two sequences. Since the scheme requires an iteration time based on the length of two input strings, the communication and computation overhead is significant.

Jha et al. [22] then improved the efficiency of Atallah and Li's scheme. They proposed three protocols to preserve privacy when calculating edit distance over Garbled Circuit [49]. However, the new scheme still has a considerable communication overhead. For example, they need about 40 seconds to compute the edit distance between two sequences, the size of which is 25.

Troncoso-Pastoriza et al. [41] proposed a scheme using an encrypted finite state machine (FSM) to compute edit distance. Although the scheme uses oblivious transfer protocol to reduce communication payload, the scheme still has considerable communication overhead. Meanwhile, the scheme may need an increasing computation overhead on FSM generation when the input length grows.

Blanton et al. [11] improved Troncoso-Pastoriza et al.'s scheme [41] by outsourcing computation and communication to cloud servers, which act as external storage and do not take over computation tasks.

Wang et al. [46] proposed a scheme that uses interactive protocols to compute genetic data with data pre-processing. The scheme leverages a key characteristic of the human genome to categorize genome into public data, which shares across human beings and private data, which differs across human beings. Therefore, the dataset with the human genome can be pre-processed into a private dataset that contains the private data, and the distance computation step can be performed on the private dataset. The data pre-processing step improves the scheme's efficiency and significantly reduces the cost of the distance computation step.

Wang et al. [47] used the size of the intersection set among multiple parties to approximate the edit distance between two sequences. The scheme reduces computation overhead and can compute the edit distance efficiently and accurately. However, the scheme has limitations on computing dataset that has a wide variety and preserving the privacy of the dataset (i.e., the scheme discloses the underlying distribution of the dataset).

Medhi et al. [8] proposed two approximate calculation schemes which are accurate, efficient and can be used either individually or together. The first scheme uses the private set intersection method and shingling to effectively approximate the edit distance between genomic sequences. To improve the accuracy of the first scheme, they applied the banded alignment algorithm to the previously proposed scheme. However, their second has a lower efficiency than their first scheme.

Zhu et al. [59] later used the Garbled Circuit [49] to accelerate the accurate edit distance calculation at a lower cost. Ashgrove et al. [5] proposed a scheme that can handle datasets with high diversity. The scheme used a 2-dimensional matrix and the block-wise edit distance approximation to improve the performance. Furthermore,

the scheme used the look-up table (LUT), which pre-calculated edit distance between blocks of sequences. The usage of the look-up table reduced the repeated cost on the most computationally expensive part in the scheme. Ashgrove et al.'s scheme [5] has an efficiency of more than 100 times to Zhu et al.'s scheme [59].

Cheng et al. [15] used the Garbled Circuit [49], homomorphic encryption and two cloud servers to reduce communication overhead when calculating similarity queries. The proposed scheme supports queries over a joint dataset collected from multiple data owners.

2.3.1.2 Open Question

Based on the secure multiparty computation model, researchers design different workflows to achieve queries over genomic sequences with no data leakage to other parties in the system. Meanwhile, schemes based on this model inherent drawbacks of the secure multiparty computation model. Schemes based on the secure multiparty computation model need multiple rounds of computations between parties and, therefore, considerable communication overhead.

The secure multiparty computation model requires all parties to stay online all the time and therefore has a high requirement on all parties' status and computational capability in the system. Computing functions and sharing secrets between parties efficiently is a significant advantage of the secure multiparty computation model. The advantage leads to the high efficiency of the schemes based on the secure multiparty computation model.

The related work shows that the efficiency and privacy of secure multiparty computation model-based schemes rely greatly on secret sharing protocols. The secret sharing protocols' performance is key in the secure multiparty computation model-based schemes. Current schemes can process queries over a large genomic dataset within several seconds.

However, the privacy of the schemes and protocols relies on the status of all parties in the scheme. Take Garbled Circuits as an example; the Garbled Circuit protocols can only achieve the security requirement of a secure multiparty computation model if the two parties in the circuit follow the designed workflow and do no step further (i.e., honest but curious). If one of the parties is a malicious party, data leakage may happen if the malicious party analyzes, compares and launches functions when processing queries. Privacy concerns still occur.

Since no additional action was taken for all parties' performance and computational capability in the schemes, the pressure on the computational capability of all parties remains with the huge communication overhead taken by the secure multiparty computation model, creating more challenges to the capabilities of all parties. Computational capability remains a concern.

2.3.2 Secure Outsourcing Model-Based Schemes

Schemes based on the secure outsourcing model use honest-but-curious adversaries to take over most computation tasks, as mentioned earlier in Section 1.4. These schemes can mitigate the pressure on data storage and no longer require every party in the system to stay online at any time.

As shown in Fig. 2.11, the basic workflow for the secure outsourcing model-based schemes has the following steps.

- ***Step 0:*** Data owners authorize query users with authorized keys.
- ***Step 1:*** Data owners outsource a genomic dataset and the index to cloud servers and then authorize the cloud servers to offer similarity query services.
- ***Step 2:*** Authorized query users will send query requests to cloud servers.
- ***Step 3:*** Cloud servers will then cooperate to search the index for a candidate set of output and verify the candidates to get precise query results. After getting a bunch of output genomic sequences as query results, cloud servers then return the

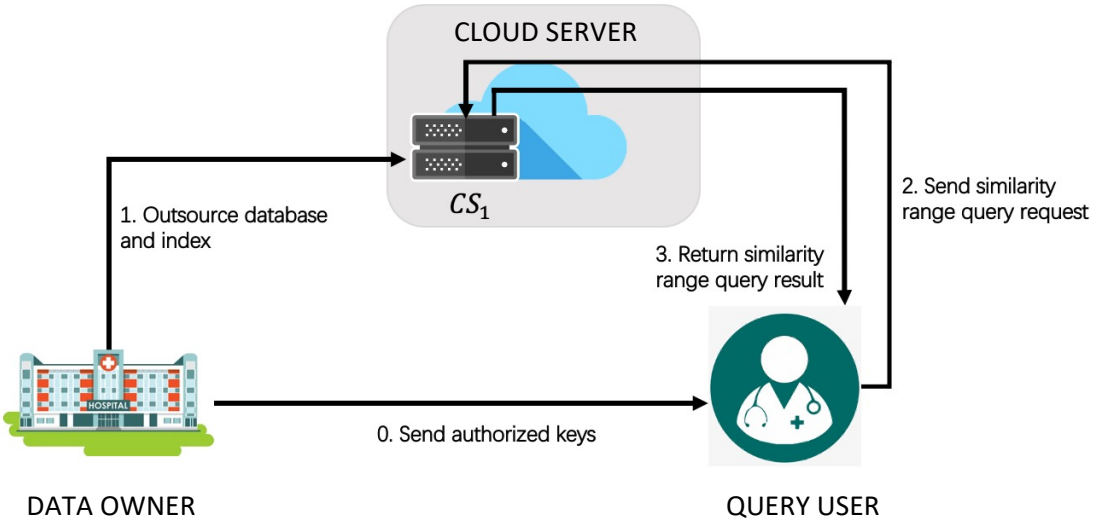

Figure 2.11: Secure outsourcing model-based scheme

result to the query users.

2.3.2.1 Related Work

In recent years, the secure outsourcing model-based scheme has been a new and trending research direction.

Wong et al. [48] proposed a SCONEDB (Secure Computation ON an Encrypted DataBase) model that can compute KNN queries over an outsourced database with an encrypted dataset in high efficiency.

Schneider et al. [37] improved Ashgrove et al.'s scheme [5] by outsourcing the dataset to a semi-honest cloud. Multiple non-collude cloud servers support the availability and correctness of this scheme.

Baldi et al. [10] used the private set intersection technique to achieve similarity computation of two sequences. The scheme builds a secure index based on nucleotide patterns, and then the cloud server calculates the number of occurrences of a specific nucleotide pattern in genomic sequences.

Chen et al. [14] proposed a scheme based on two types of cloud servers. The scheme first uses a public cloud to process sequences over encrypted data then perform

accurate calculations between plaintexts in a private cloud.

Yasuda et al. [50] used somewhat homomorphic encryption [31] to compute similarity. The scheme gave a practical solution of computing multiple distances over encrypted data efficiently. It can compute the distance between two sequences with a maximum length of 2048 within 12.43 ms.

Cheon et al. [16] used lattice encryption [2], a type of homomorphic encryption, to compute the edit distance of two sequences. The scheme used a recursive algorithm to compute edit distance over two sequences dynamically and requires a large computational overhead. It requires 27.5 seconds to compute the edit distance between two sequences at length 8.

Wang et al. [45] proposed a scheme for similarity queries on an untrusted cloud. The scheme used a tree structure and order-preserving encryption [35] to achieve the similarity query in sub-linear query efficiency. Specifically, the scheme needs an average of 4 seconds to search over a dataset of 1 million encrypted data on the Amazon EC2 platform.

Wang et al. [44] proposed a one-round interaction communication scheme. It achieves a wildcard-based sequence pattern match with a predicate encryption scheme [25] and suffix tree. The scheme preserves the privacy of both dataset and search pattern at a reasonable cost.

2.3.2.2 Open Question

The main question for secure outsourcing model-based schemes for similarity queries over genomic sequences are privacy and efficiency.

The related work shows that the efficiency and privacy of outsourcing model-based schemes rely significantly on the design of schemes and the privacy of the cryptosystem used in the scheme. Security concerns have become more critical. Some of the existing solutions cannot guarantee their security as the cryptosystem used in their

schemes leads to leakage of query patterns to the cloud servers. Since most of the computation tasks are taken over by cloud servers, we need to define the computation process for cloud servers securely.

Another concern of the secure outsourcing model-based schemes is efficiency. In this type of scheme, encrypting the dataset and index before outsourcing is necessary, and most of the computations on the cloud servers are processed over encrypted values. Therefore, using homomorphic cryptosystems in the schemes is becoming more popular. Compared to other cryptosystems, homomorphic cryptosystems are still in an early phase and have worse performance in applications, especially those with complex computations and heavy works. The severe work to compute similarity queries over genomic sequences makes secure outsourcing model-based schemes much less efficient than secure multiparty computation model-based schemes. Considering how to improve the performance of secure outsourcing model-based schemes is a big challenge.

With the development of technology, there are more data transmitting, sharing, and changing every day. Meanwhile, current solutions achieve sublinear query efficiency by building indexes for the dataset and do not support dataset updates. Unable to support dynamic updates of dataset makes the schemes costly in maintenance when facing data bloom.

Chapter 3

Preliminaries

In this chapter, we briefly review the edit distance algorithm over genomic sequences and a similarity range query algorithm over a hash table.

3.1 Edit Distance Over Genomic Sequences

Edit distance (also known as Levenstein distance) is first presented by Levenstein in 1965 [29]. It is a distance metric used to measure the similarity between two sequences, e.g., genomic sequences, a string of nucleotides. Edit distance over genomic sequences computes the minimum number of operations required to transform one genomic sequence into the other. There are three types of edit distance operations for genomic sequences, i.e., insertion, deletion, and substitution of one nucleotide. The edit operations over genomic sequences are defined formally as follows [52].

• *Insertion:* When inserting a nucleotide c into a genomic sequence $s = s_1, s_2, ..., s_{l_i}$, where l_i is the length of the sequence, at position i, we can transfrom the genomic sequence s into $s_1, s_2, ..., s_i, c, s_{i+1}, s_{l_i}$.

• *Deletion:* When deleting a nucleotide c into a genomic sequence $s = s_1, s_2, ..., s_{l_i}$, where l_i is the length of the sequence, at position i, we can transfrom the genomic sequence s into $s_1, s_2, ..., s_{i-1}, s_{i+1}, s_{l_i}$.

• *Substitution:* When replacing a nucleotide at position i in genomic sequence $s = s_1, s_2, ..., s_{l_i}$, where l_i is the length of the sequence, with a character c, we can transfrom the genomic sequence s into $s_1, s_2, ..., s_{i-1}, c, s_{i+1}, s_{l_i}$.

3.1.1 Edit distance computation algorithm

The edit distance [33] between two genomic sequences can be computed by Wagner-Fischer algorithm [43]. The algorithm processes by recursively computing the edit distance of subsequences. Suppose that $s_i = s_{i,1}, s_{i,2}, ..., s_{i,l_i}$ and $s_j = s_{j,1}, s_{j,2}, ..., s_{j,l_j}$ are two genomic sequences, where l_i and l_j are the lengths of the two sequences, respectively. The algorithm first prepares an $(l_i + 1) \times (l_j + 1)$ distance matrix D. Each $\mathrm{D}[i', j']$ denotes the edit distance between the subsequence $s_{i,1}, s_{i,2}, ..., s_{i,i'}$ and $s_{j,1}, s_{j,2}, ..., s_{j,j'}$ for $i' = 1, 2, ..., l_i$ and $j' = 1, 2, ..., l_j$. The first row and column of the matrix are initialized by $\mathrm{D}[i', 0] = i'$ and $\mathrm{D}[0, j'] = j'$ for all $0 \leq i' \leq l_i$ and $0 \leq j' \leq l_j$. The algorithm then iteratively sets the rest of the matrix D, $\mathrm{D}[i', j']$ for all $1 \leq i' \leq l_i$ and $0 \leq j' \leq l_j$, as

$$D(i,j) = \min \begin{cases} \mathrm{D}[i'-1, j'-1] + cost, \\ \mathrm{D}[i'-1, j'] + 1, \\ \mathrm{D}[i', j'-1] + 1. \end{cases}$$

where $cost = 0$ if $s_{i,i'} = s_{j,j'}$, and $cost = 1$ otherwise.

Therefore, the Wagner-Fischer algorithm can compute all values in the distance matrix D, and we finally can get $d(s_i, s_j) = \mathrm{D}[l_i, l_j]$.

3.2 Paillier Encryption

The Paillier encryption technique [34] is a homomorphic encryption scheme that only supports the homomorphic operation of addition. Based on definition 2, the Paillier

encryption operation has a homomorphism from $(\mathbb{Z}_{\mathtt{n}}, +)$ to $(\mathbb{Z}_{\mathtt{n}^2}, \cdot)$.

The Paillier encryption scheme consists of three algorithms, i.e., key generation, encryption and decryption.

• *Key Generation:* Given a security parameter κ, the key generation algorithm first randomly chooses two large prime numbers $\mathtt{p}$, $\mathtt{q}$ of length κ. Let $\mathtt{n} = \mathtt{pq}$ and $\lambda = lcm(\mathtt{p}-1, \mathtt{q}-1)$. Then it selects a generator $\mathtt{g} \in \mathbb{Z}^*_{\mathtt{n}^2}$ and calculates $\mathtt{u} = (L(\mathtt{g}^\lambda \mod \mathtt{n}^2))^{-1} \mod n$ where $L(\mathtt{x}) = \frac{\mathtt{x}-1}{\mathtt{n}}$. Finally, the algorithm outputs the public key $pk = (\mathtt{n}, \mathtt{g})$, and the corresponding secret key $sk = (\lambda, \mathtt{u})$.

• *Encryption:* With the public key pk, the encryption algorithm first chooses a random number $r \in \mathbb{Z}^*_{\mathtt{n}}$ then encrypts a message $m \in \mathbb{Z}_{\mathtt{n}}$ as $c = E(m) = \mathtt{g}^m r^{\mathtt{n}} \mod \mathtt{n}^2$.

• *Decryption:* Given the ciphertext $c \in \mathbb{Z}^*_{\mathtt{n}^2}$ and the secret key sk, the decryption algorithm recovers a message m' as $m' = D(c) = L(c^\lambda \mod \mathtt{n}^2) \times u \mod \mathtt{n}$.

3.2.1 Correctness of Paillier Encryption

The correctness of Paillier encryption can be proved by showing out that the key pair used in the algorithm can cancel each other out [27].

$$
\begin{aligned}
m &= L(c^\lambda \mod \mathtt{n}^2) \times u \mod \mathtt{n} \\
&= L(c^\lambda \mod \mathtt{n}^2) \times (L(\mathtt{g}^\lambda \mod \mathtt{n}^2))^{-1} \mod \mathtt{n} \\
&= \frac{L(c^\lambda \mod \mathtt{n}^2)}{L(\mathtt{g}^\lambda \mod \mathtt{n}^2)} \mod \mathtt{n} \\
&= \frac{L((\mathtt{g}^m \times r^{\mathtt{n}})^\lambda \mod \mathtt{n}^2)}{L(\mathtt{g}^\lambda \mod \mathtt{n}^2)} \mod \mathtt{n} \\
&= \frac{L(\mathtt{g}^{m\lambda} \times r^{\mathtt{n}\lambda} \mod \mathtt{n}^2)}{L(\mathtt{g}^\lambda \mod \mathtt{n}^2)} \mod \mathtt{n} \\
&= \frac{L(\mathtt{g}^{m\lambda} \mod \mathtt{n}^2)}{L(\mathtt{g}^\lambda \mod \mathtt{n}^2)} \mod \mathtt{n} \\
&= m \mod \mathtt{n}
\end{aligned}
\tag{3.1}
$$

3.2.2 Properties of Paillier Encryption

The Paillier encryption technique has the following two properties.

• *Homomorphic Addition of Plaintexts:* The encryption of the sum of two plaintext is equivalent to the product of the two corresponding ciphertext. Specifically, given two ciphertexts $E(m_1)$ and $E(m_2)$, we have $E(m_1) \times E(m_2) = E(m_1 + m_2)$. The addictive homomorphic property can be proved as equation 3.2.

$$\begin{aligned} E(m_1) \times E(m_2) &= \mathtt{g}^{m_1} {r_1}^{\mathtt{n}} \mathtt{g}^{m_2} {r_2}^{\mathtt{n}} \mod \mathtt{n}^2 \\ &= \mathtt{g}^{m_1+m_2} (r_1 \times r_2)^{\mathtt{n}} \mod \mathtt{n}^2 \\ &= E(m_1 + m_2) \end{aligned} \tag{3.2}$$

• *Homomorphic Multiplication of Plaintexts:* The encryption of the product of two plaintext is equlivalent to the value of a ciphertext of one plaintext to the value of the other plaintext. Specifically, given a ciphertext $E(m_1)$ and a plaintext $m_2 \in \mathbb{Z}_{\mathtt{n}}$, we have $E(m_1)^{m_2} = E(m_1 \times m_2)$. The multiplicative homomorphic property can be proved as equation 3.3.

$$\begin{aligned} E(m_1)^{m_2} &= (\mathtt{g}^{m_1} r^{\mathtt{n}})^{m_2} \mod \mathtt{n}^2 \\ &= \mathtt{g}^{m_1 \times m_2} (r^{\mathtt{n}})^{m_2} \mod \mathtt{n}^2 \\ &= E(m_1 \times m_2) \end{aligned} \tag{3.3}$$

3.2.3 Security of Paillier Encryption

The original Paillier encryption [34] is semantic secure against chosen-plaintext attacks (i.e., IND-CPA). Detailed analysis can be found in [34].

3.3 Hash Table-based Similarity Range Query

In this section, we first formally define the similarity range query problem. Then we introduce an efficient similarity range query algorithm over a hash table. As mentioned in Section 1.2, the similarity range query problem is defined as follows.

Definition 4 (Similarity range query)**.** *Suppose that* $\mathcal{S} = \{s_i = s_{i,1}, s_{i,2}, ..., s_{i,l_i} | i = 1, 2, ..., N\}$ *is a dataset. Each* $s_i \in \mathcal{S}$ *has a unique identity* id_i*. Then suppose* (q, k) *is a query request where* q *is a query sequence, and* k *is a similarity range threshold. The similarity range query is to search* $\mathcal{S}$ *for a collection of sequences whose distance to* q *is equal to or less than* k*, i.e.,* $\{id_i | d(s_i, q) \leq k\}$*.*

To achieve efficient similarity range queries over genomic sequences, Zhang et al. [54] proposed a similarity range query scheme based on a hash table. The scheme first uses a hash table to index the dataset. Then the similarity range query algorithm is built over the hash table, which stores subsequences systematically.

3.3.1 Hash Table Generation

The hash table can efficiently support the similarity range query over sequences of different lengths in the dataset. Suppose that $\mathcal{S} = \{s_i = s_{i,1}, s_{i,2}, ..., s_{i,l_i} | i = 1, 2, ..., N\}$ is a genomic dataset. We can build a hash table based on $\mathcal{S}$ by recursively partitioning s_i in the dataset. Specifically, each s_i is partitioned as follows.

• ***Step 1:*** We give nucleotides in s_i different values and store them in an array P. An n-grams of nucleotide $s_{i,i'}$ in s_i where $1 \leq i' \leq l_i$ is a sequence of n nucleotides starting from $s_{i,i'}$, i.e., $s_{i,i'}, s_{i,i'+1}, ..., s_{i,i'+n-1}$. The first $l_i - n + 1$ nucleotides in s_i have n-grams and will be given the hash value of their n-grams. For each nucleotide that has n-grams, both its n-gram and its value will be stored in P.

Example 1. *Suppose we have a sequence* $s_1 =$ CACGGGAGCTCTGCAT *with* id_1*. The length of* s_1*, which denotes* l_1*, is 16. We use* $n = 3$ *in the algorithm. We*

first find 3-grams for each of the first 16-3+1=14 nucleotides in s_1. Then, give each nucleotide a hash value of their 3-grams. We finally store the 14 nucleotides with corresponding n-grams and values in the array P. Therefore, we can get TABLE 3.1.

Table 3.1: Array P for example 1

$s_{i,i'}$	n-grams	value
C	CAC	0.43
A	ACG	0.35
C	CGG	0.11
G	GGG	0.38
G	GGA	0.40
G	GAG	0.03
A	AGC	0.29
G	GCT	0.05
C	CTC	0.01
T	TCT	0.26
C	CTG	0.22
T	TGC	0.12
G	GCA	0.19
C	CAT	0.21

Table 3.2: Array R for example 2

i'	$R_{i'}$
1	∞
3	2
6	2
9	5
12	2
17	∞

• ***Step 2:*** We give each $s_{i,i'} \in P$ a rank value $R_{i'}$ and store some rank values in an array R. Specifically, $R_{i'}$ of nucleotide $s_{i,i'}$ is the radius of the neighbourhood in which $R_{i'}$ is a local minimum. All $R_{i'}$'s have a default value one. Rank values store in R has a lower bound $rank_{min} = 1$ to support query sequences of different lengths and are represented as $(i', R_{i'})$ pairs. Meanwhile, all $(i', R_{i'})$ pairs in R are stored with increasing order of i'. We finally insert two $(i', R_{i'})$ pairs with values of $(1, \infty)$ and $(l_i + 1, \infty)$ into the front and the end of the array R to help to partition sequences in step 3.

Example 2. *Based on hash values in TABLE 3.1, we can give each nucleotide in P a rank value. Each $R_{i'}$ has a default value one. When $i' = 2$, $s_{1,i'}$ is less than $s_{1,3}$ and $s_{1,1}$. Therefore, $s_{1,2}$ is a local minimum of a neighbourhood with a radius of one. However, since $i' - 2 < 1$, we cannot enlarge the size of the neighbourhood for R_2. Finally, we cannot add $(i', R_{i'})$ into the array R since $R_{i'} = rank_{min} = 1$.*

For $i' = 3$, we can see that $s_{1,3}$ is less than $s_{1,4}$ and $s_{1,2}$, and $i' - 2 \geq 1$. Therefore, we can now increase R_3 by one. We can then find that $s_{1,3}$ is less than both $s_{1,5}$ and $s_{1,1}$. However, since $i' - 3 < 1$, we cannot further enlarge the size of the neighbourhood or increase the value of R_3. Finally, since $R_{i'} > rank_{min} = 1$, we can now add $(i', R_{i'})$ into R.

We can recursively get rank values for all $1 \leq i' \leq l_i - n + 1$. Since $l_i + 1 = 17$, we can finally insert two $(i', R_{i'})$ pairs with values $(1, \infty)$ and $(17, \infty)$ into the array R and get TABLE 3.2.

• ***Step 3:*** We partition s_i into l_b subsequences. Specifically, s_i will be partitioned by decreasing values of $R_{i'}$ in R. We first find the greatest $R_{i'}$ where $1 < i' \leq l_i$. Then partition s_i into two subsequences $s_{i:sub_1} = s_{i,1}, s_{i,2}, ..., s_{i,i'-1}$ and $s_{i:sub_2} = s_{i,i'}, s_{i,i'+1}, ..., s_{i,l_i}$. For $s_{i:sub_1}$ (resp. $s_{i:sub_2}$), we attach a level value $l_{i:sub_1}$ (resp. $l_{i:sub_2}$) to it. The value of $l_{i:sub_b}$ equals a smaller rank value between the first nucleotide in $s_{i:sub_b}$ and after $s_{i:sub_b}$ in s_i where $b = 1$ or 2. If s_i does not contain any nucleotide after $s_{i:sub}$, as mentioned earlier, the rank value of s_{i,l_i+1} is ∞. Each subsequence can be represented as an $(s_{i:sub_b}, l_{i:sub_b})$ pair where $1 \leq b \leq l_b$.

Example 3. *Based on TABLE 3.2, we can get when $i' = 9$, $R_{i'}$ reaches its greatest value where $1 < i' < l_i$. Therefore, we first partition the sequence with $i' - 1 = 8$ then get $s_{1:sub_1} =$ CACGGGAG and $s_{1:sub_1} =$ CTCTGCAT. Since $l_i + 1 = 17$ and $R_1 = R_{17} > R_9$, we can get $l_{i:sub_1} = l_{i:sub_2} = R_9 = 5$. Therefore, s_1 can be first parititioned into two subsequences* $(\text{CACGGGAG}, 5)$ *and* $(\text{CTCTGCAT}, 5)$. *Finally, we can get eight subsequences after partitioning s_1 by $R_{i'} \in R$ where $1 < i' < l_i$ with decreasing order, as shown in Fig. 3.1.*

• ***Step 4:*** We generate a hash table T, which consists of multiple level buckets $T_{l_{i:sub_b}}$ which indexed by different level values. Each level bucket stores multiple $s_{i:sub_b}$ that attached the same $l_{i:sub_b}$ and consists of several subsequence buckets. Each

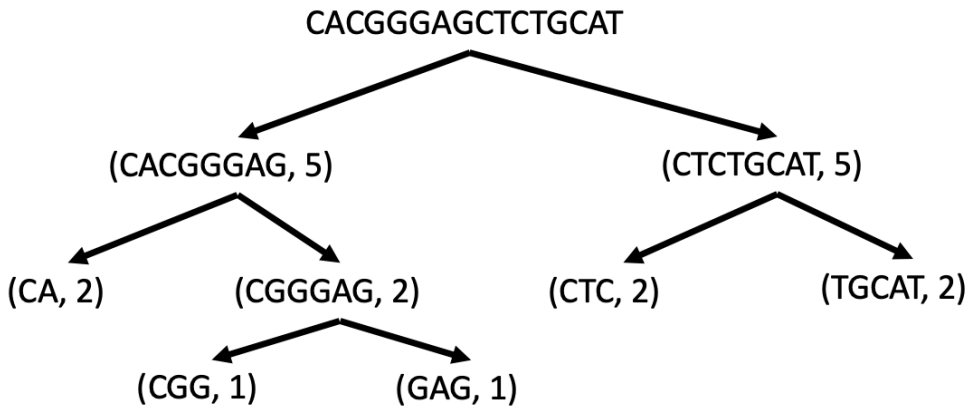

Figure 3.1: Illustration of Step 3 for s_1

subsequence bucket (denoted by $T_{l_{i:sub_b}}(s_{i:sub_b})$) is indexed by $s_{i:sub_b}$'s and contains the identities of sequence that can be partitioned into $(s_{i:sub_b}, l_{i:sub_b})$. Each $(s_{i:sub_b}, l_{i:sub_b})$ will be added into T once it is partitioned in step 3. After adding all subsequences, we can finally get T.

Suppose a $(s_{i:sub_b}, l_{i:sub_b})$ pair where $1 \leq b \leq l_b$ is a subsequence partitioned from sequence s_i. If the level bucket $T_{l_{i:sub_b}}$ already exists, we search the bucket and add the identity id_i into the corresponding subsequence bucket. If the subsequence bucket $T_{l_{i:sub_b}}(s_{i:sub_b})$ exists, we directly add id_i into the bucket. Otherwise, we generate an empty subsequence bucket before adding identity. All subsequences will be stored in the hash table according to $s_{i:sub_b}$ and $l_{i:sub_b}$.

Example 4. *We can generate the hash table by adding all eight subsequences into a hash table. For the first subsequence* $(\text{CACGGGAG}, 5)$*, we first search for a level bucket* T_5 *in the hash table* T*.* T *does not contain the level bucket* T_5*, so we create a new level bucket* T_5*. Since* T_5 *is newly created and empty, we create a subsequence bucket* $T_5(\text{CACGGGAG})$ *within* T_5 *and add the identity* id_1 *into the subsequence bucket.*

For the second subsequence $(\text{CTCTGCAT}, 5)$*, we first search for* T_5*, which already exists in* T*. We then search for* $T_5(\text{CTCTGCAT})$ *and find that* T_5 *does not contain the subsequence bucket. We create the subsequence bucket within* T_5 *and add* id_1

into the subsequence bucket. Therefore, we update T_5 *and get* $\{\text{CACGGGAG} \rightarrow \{id_1\}, \text{CTCTGCAT} \rightarrow \{id_1\}\}$. *All eight subsequences can follow the same pattern and be filled into Table 3.3.*

Meanwhile, if another sequence s_2 *with identity* id_2 *is partitioned into an existing subsequence, e.g.,*$(\text{GAG}, 1)$*, we add* id_2 *into the subsequence bucket* $T_1(\text{GAG})$ *and update it into* $\{id_1, id_2\}$. *The updated* T_1 *bucket is* $\{\text{CGG} \rightarrow \{id_1\}, \text{GAG} \rightarrow \{id_1, id_2\}\}$.

Table 3.3: Hash table based on Fig. 3.1

Level	T_l bucket
5	$\{\text{CACGGGAG} \rightarrow \{id_1\}, \text{CTCTGCAT} \rightarrow \{id_1\}\}$
2	$\{\text{CA} \rightarrow \{id_1\}, \text{CGGGAG} \rightarrow \{id_1\}, \text{CTC} \rightarrow \{id_1\},$ $\text{TGCAT} \rightarrow \{id_1\}\}$
1	$\{\text{CGG} \rightarrow \{id_1\}, \text{GAG} \rightarrow \{id_1\}\}$

3.3.2 Hash Table-Based Similarity Range Query Algorithm

We introduce a hash table-based similarity range query algorithm proposed by Zhang et al. [54]. The algorithm is based on Theorem 1, which has been proved in [53]. An idea of the algorithm is to find a set of sequences in the dataset that share at least one subsequence with the query sequence.

Theorem 1. *Let* (s_i, q) *be two sequences, and they are partitioned with rank values larger than* $max\{1, \frac{l_q - n + 1 - \tau k}{2\tau k + 2}\}$*, where* $\tau = 120$ *and* n *is the length of* n*-grams for partitioning sequences. Then,* s_i *and* q *will share at least one common subsequence with a probability of 0.99, if* $d(s_i, q) \leq k$.

Suppose that we have a dataset $\mathcal{S} = \{s_i = s_{i,1}, s_{i,2}, ..., s_{i,l_i} | i = 1, 2, ..., N\}$, a query sequence $q = q_1, q_2, ..., q_{l_q}$ and a threshold value k. We first generate a hash table T to index the dataset $\mathcal{S}$. To get all sequences satisfying $d(s_i, q) \leq k$, we can search over the hash table with three stages, i.e., hash token genration, filtration and verification.

3.3.2.1 Hash Token Generation Stage

In the hash token generation stage, we generate a hash token t, a set of all subsequences partitioned from a query sequence q, for similarity range query requests. A similarity range query request consists of a threshold value k, the query sequence q and the hash token t. We generate the hash token t from the query sequence q by following steps 1, 2 and 3 in the hash table generation process, where $rank_{min}$ is set to be $max\{1, \frac{l_q-n+1-\tau k}{2\tau k+2}\}$ and τ is set to be 120.

3.3.2.2 Filtration Stage

According to Theorem 1, in the filtration stage, we filter genomic sequences in dataset $\mathcal{S}$ by finding sequences that share at least one common subsequence with the query sequence q. We recursively search the hash table T by subsequences in the hash token.

Suppose a hash token contains $l_{b'}$ subsequences $(q_{sub_{b'}}, l_{sub_{b'}})$, where $1 \leq b' \leq l_{b'}$. We recursively collect all id_i in $T_{l_{sub_{b'}}}(q_{sub_{b'}})$ buckets for each subsequence and form a candidate set $\mathcal{C}$. Since a sequence s_i may share multiple common subsequences with q, the set $\mathcal{C}$ may contain duplicate identites. Therefore, we need to remove duplicate identites in $\mathcal{C}$. We finally get a candidate set $\mathcal{C}$ after removing sequences with l_i either greater than $l_q + k$ or less than $l_q - k$, which has a distance to q greater than k.

3.3.2.3 Verification Stage

The purpose of the verification stage is to verify if the candidate sequences are qualified outputs of the algorithm. For each $id_i \in \mathcal{C}$, we calculate its distance to q. If a sequence has a distance to q less than or equal to k (i.e., $d(s_i, q) \leq k$), add the identity id_i into a set $\mathcal{O}$. Finally, we can get the output set $\mathcal{O}$.

Chapter 4

Models and Design Goals

In this chapter, we formalize our system model and security model, and identify our design goals.

4.1 System Model

In our system, we consider a cloud-assisted similarity range query model over genomic sequences. The model involves a healthcare center HC, two cloud servers $\{CS_1, CS_2\}$ and multiple query doctors, as shown in Fig. 4.1.

4.1.1 Healthcare Center

In our system model, the healthcare center HC has a genomic dataset $\mathcal{S}$ containing N genomic sequences and act as the data owner. Each sequence $s_i = s_{i,1}, s_{i,2}, ..., s_{i,l_i} \in \mathcal{S}$ is a genomic sequence of length l_i and has a unique identity id_i. HC is willing to offer similarity range query services over genomic sequences to authorized query doctors. However, since it has limited computing capability and storage space, it outsources the dataset to the cloud before employing cloud servers to offer the similarity range query services to authorized query doctors. Since genomic sequences are sensitive data, HC will encrypt the genomic dataset before outsourcing them to the cloud.

4.1.2 Cloud Servers

We consider two cloud servers, namely CS_1 and CS_2, in our system model. Both of them are powerful in computing capability and have large storage space. They store the encrypted genomic dataset outsourced by HC and work together to offer the similarity range query services to authorized query doctors.

4.1.3 Query Doctors

The system has multiple query doctors who launch queries. When a query doctor registers in the system, the query doctor will be authorized by HC with authorized keys (as shown in Fig. 4.1). After the authorization, the query doctor can launch similarity range queries by sending similarity range query requests to cloud servers. Specifically, a query doctor first generates an encrypted similarity range query request from a query sequence q. The query doctor then sends the request with a threshold value k to the cloud. The cloud servers will return the identity of sequences that has a distance less than or equal to k, i.e., $\{id_i | d(s_i, q) \leq k\}$ to the query doctor, where $d(\cdot, \cdot)$ denotes the edit distance between s_i and q.

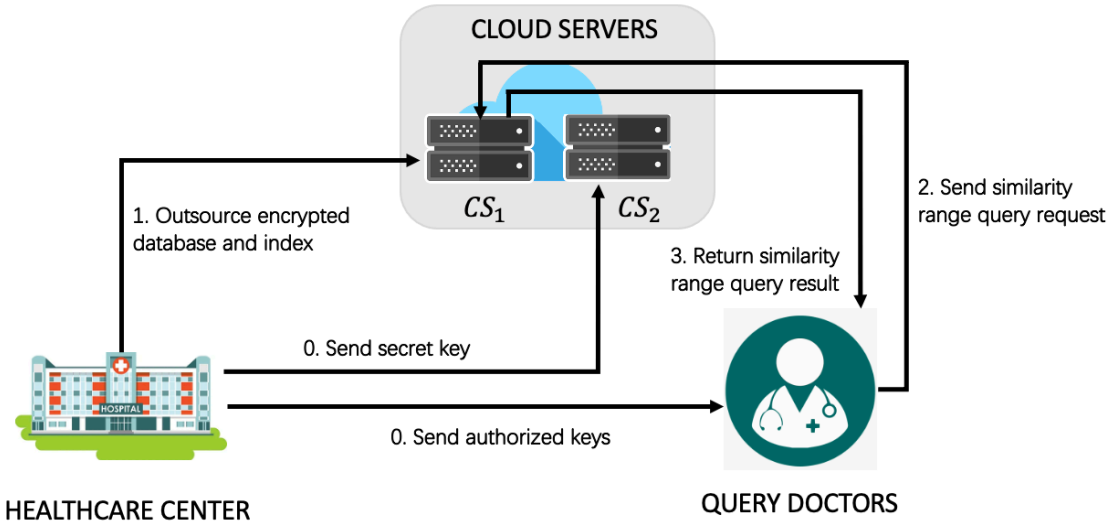

Figure 4.1: System model

4.2 Security Model

In our security model, the healthcare center HC is trusted as it owns the dataset and has no motivation to deviate from the service. Meanwhile, the query doctors are assumed to be honest. Specifically, they will send similarity range query requests to the cloud servers honestly. For the cloud servers CS_1 and CS_2, both of them are honest-but-curious. Specifically, they will honestly store the outsourced data from HC, perform the similarity range query over encrypted data and return the query results to the query doctors. However, they may be curious about the plaintext of genomic sequences in the dataset and query sequences. We also assume no collusion between CS_1 and CS_2 as they have conflicts of interest [57]. Note that there may be active attacks, e.g., Denial of Service (DoS) attacks, in the system. Those attacks are beyond the scope of this thesis and will be discussed in our future work.

4.3 Design Goal

Our design goal is to present an efficient and privacy-preserving similarity range query scheme over genomic sequences. Specifically, two requirements should be satisfied.

• *Privacy preservation:* The primary requirement of our scheme is privacy-preservation. We focus on preserving the privacy of both the outsourced data and query requests.

• *Efficiency:* We aim to minimize the computational cost of similarity range queries and optimize the performance of the proposed scheme.

Chapter 5

Our Proposed Scheme

In this chapter, we first introduce two privacy-preserving protocols. Then, we present an efficient and privacy-preserving similarity range query scheme over genomic sequences based on the two privacy-preserving protocols and the similarity range query algorithm introduced in Section 3.3.

5.1 Privacy-Preserving Protocols

We introduce two privacy-preserving protocols based on the Paillier encryption technique mentioned in Section 3.2 and the model mentioned in Chapter 4. The two privacy-preserving protocols include a minimum value computation protocol and an equality test protocol. All parameters and algorithms used in the protocols are based on the Paillier encryption technique.

5.1.1 Minimum Value Computation Protocol

The privacy-preserving minimum value computation protocol obtains the encrypted minimum value among the two ciphertexts. Specifically, given $E(m_1)$ and $E(m_2)$, the protocol computes $E(\min\{m_1, m_2\})$. It involves two cloud servers, namely CS_1 and CS_2. CS_1 has ciphertexts $E(m_1)$ and $E(m_2)$, where $m_1, m_2 \in \{0,1\}^a$ and

$a \ll \mathtt{n}$, and CS_2 has the corresponding secret key sk. CS_1 and CS_2 can then obtain $E(\min\{m_1, m_2\})$ while preserving the privacy of m_1 and m_2 as follows.

• ***Step 1:*** CS_1 uses Paillier encryption technique's homomorphic properties to compute $E(\mathtt{m}) = E(m_1 - m_2)$.

• ***Step 2:*** CS_1 first chooses two random numbers $\gamma_1, \gamma_2 \in \{0,1\}^b$ where $a + b \ll \mathtt{n}$ and $\gamma_1 \geq \gamma_2$ and computes $E(\mathtt{m}') = E(\mathtt{m} \times \gamma_1 + \gamma_2)$ with Paillier encryption technique's homomorphic properties. It then computes $E(M) = E(\mathtt{m} + \gamma)$ with a random number γ satisfying $\gamma \in \mathbb{Z}_{\mathtt{n}} \setminus \{0\}$ and $\gamma > \frac{\mathtt{n}}{2}$. Finally, CS_1 sends $\{E(\mathtt{m}'), E(M)\}$ to CS_2.

• ***Step 3:*** On receiving $E(\mathtt{m}')$ and $E(M)$, CS_2 uses sk to recover $\mathtt{m}'$ and M. It then sets $E(res) = E(1)$ if $\mathtt{m}'$ is larger than $\frac{\mathtt{n}}{2}$ and $E(res) = E(0)$ otherwise. Then CS_2 returns $E(M \times res)$ and $E(res)$ to CS_1.

• ***Step 4:*** With the return results from CS_2, CS_1 can obtain $E(\min\{m_1, m_2\}) = E(M \times res) \times E(res)^{-\gamma} \times E(m_2) = E(\mathtt{m} \times res + m_2) = E((m_1 - m_2) \times res + m_2)$.

Correctness: The minimum value computation protocol is correct if and only if $E(\min(m_1, m_2)) = E(M \times res) \times E(res)^{-\gamma} \times E(m_2)$. According to the homomorphic properties of the Paillier encryption technique, it is equivalent to prove that $E(\min(m_1, m_2) = E((m_1 - m_2) \times res + m_2)$ equals to $E(m_1)$ when $m_1 < m_2$ and $E(m_2)$ otherwise.

Proof. When $m_1 < m_2$, we have $\mathtt{m} = m_1 - m_2 < 0$. Then, $\mathtt{m}' = \mathtt{m} \times \gamma_1 + \gamma_2$ is larger than $\frac{\mathtt{n}}{2}$. We have $E(res) = E(1)$ and $E(\min\{m_1, m_2\}) = E(m_1 - m_2 + m_2) = E(m_1)$. Similarly, when $m_1 \geq m_2$, we have $\mathtt{m} = m_1 - m_2 \geq 0$. Then, $\mathtt{m}' = \mathtt{m} \times \gamma_1 + \gamma_2 < \frac{\mathtt{n}}{2}$. We have $E(res) = E(0)$ and $E(\min\{m_1, m_2\}) = E((m_1 - m_2) \times res + m_2) = E((m_1 - m_2) \times 0 + m_2) = E(m_2)$. ■

5.1.2 Equality Test Protocol

The privacy-preserving equality test protocol is to check if two ciphertexts share the same plaintext. Specifically, given two ciphertexts, $E(m_1)$ and $E(m_2)$, it checks if

m_1 equals to m_2. The protocol involves two cloud servers, namely CS_1 and CS_2. CS_1 holds $E(m_1)$ and $E(m_2)$, while CS_2 holds the corresponding secret key sk. CS_1 and CS_2 can obtain if m_1 equals to m_2 while preserving the privacy of m_1 and m_2 as follows.

• ***Step 1:*** CS_1 chooses a random number $\gamma \in \mathbb{Z}_{\mathtt{n}} \setminus \{0\}$.

• ***Step 2:*** CS_1 computes $E(\mathtt{m}) = E((m_1 - m_2) \times \gamma)$ with homomorphic properties and sends $E(\mathtt{m})$ to CS_2.

• ***Step 3:*** On receiving $E(\mathtt{m})$, CS_2 uses sk to recover $\mathtt{m}$. CS_2 then generates $E(res) = E(0)$ if $\mathtt{m} = 0$ and $E(res) = E(1)$ otherwise. Finally, CS_2 returns $E(res)$ to CS_1.

Correctness: The equality test protocol is correct if and only if $m_1 = m_2 \Leftrightarrow res = 0$ and $m_1 \neq m_2 \Leftrightarrow res = 1$. Since the two statements are equivalent to each other, we only prove the statement $m_1 = m_2 \Leftrightarrow res = 0$.

Proof. We prove the statement $m_1 = m_2 \Leftrightarrow res = 0$ by proving its sufficiency and necessity, respectively.

Sufficiency: In the equality test protocol, CS_1 computes $E(\mathtt{m}) = E((m_1 - m_2) \times \gamma)$. When $m_1 = m_2$, $\mathtt{m} = 0$ holds since $m_1 - m_2 = 0$. Therefore, CS_2 will generate $E(res) = E(0)$ after recovering $\mathtt{m}$. We now proved that $m_1 = m_2 \Rightarrow res = 0$.

Necessity: $res = 0$ is equivalent to $(m_1 - m_2) \times \gamma = 0$ in the equality test protocol. Since $\gamma \in \mathbb{Z}_{\mathtt{n}} \setminus \{0\}$, we can get that $m_1 - m_2 = 0$, which is equivalent to $m_1 = m_2$. We now proved that $res = 0 \Rightarrow m_1 = m_2$. ■

5.2 Description of Our Scheme

In this section, we present our similarity range query scheme based on the hash table-based similarity range query algorithm and the above privacy-preserving protocols. Specifically, the proposed scheme consists of four phases, including system initialization, local data outsourcing, similarity range query processing, and dynamic

update of dataset.

5.2.1 System Initialization

The healthcare center HC initializes the whole system. Firstly, given security parameter κ, HC calls the key generation algorithm of Paillier encryption technique to generate a key pair (i.e., a secret key sk and a public key pk). Then, HC publishes pk and sends sk to CS_2. Meanwhile, HC picks a random hash function $H(\cdot)$ and a random n. It builds a hash table to index the dataset. Furthermore, to secure the hash table, HC chooses a key K and generates a key-based hash function $h_K(\cdot)$. Finally, when a query doctor registers in the system, HC will authorize it by sending $H(\cdot)$, $h_K(\cdot)$ and n to the query doctor.

5.2.2 Local Data Outsourcing

HC builds a hash table to index the dataset. Then it encrypts the hash table and the dataset before outsourcing them to a cloud server.

Suppose $\mathcal{S} = \{s_i = s_{i,1}, s_{i,2}, ..., s_{i,l_i} | i = 1, 2, ..., N\}$ is a dataset. Each $s_i \in \mathcal{S}$ has a unique identity id_i. HC outsources the dataset to cloud server CS_1 as follows:

• ***Step 1:*** Based on $H(\cdot)$ and n, HC builds a hash table T for the dataset $\mathcal{S}$ with $rank_{min} = 1$ (as described in Section 3.3).

• ***Step 2:*** HC encrypts the hash table T as $E(T)$. The hash table T has multiple level buckets, and each level bucket consists of several subsequence buckets which store identities. The hash table is encrypted as follows:

(1) HC encrypts every identity id_i in subsequence buckets as $E(id_i)$ using public key pk.

(2) HC replaces the indexes of subsequence buckets $s_{i:sub_b}$ with its hash value $h_K(s_{i:sub_b})$.

• ***Step 3:*** HC encrypts the dataset $\mathcal{S}$. For a genomic sequence $s_i = s_{i,1}, s_{i,2}, ..., s_{i,l_i}$ with identity id_i in the dataset, HC encrypts s_i as $E(s_i) = E(s_{i,1}), E(s_{i,2}), ..., E(s_{i,l_i})$,

and id_i remains in plaintext.

• ***Step 4:*** HC outsources the encrypted hash table $E(T)$ and the encrypted dataset, denoted by $E(\mathcal{S}) = \{E(s_i) = E(s_{i,1}),\ E(s_{i,2}), ..., E(s_{i,l_i}) | i = 1, 2, ..., N\}$ with identities in plaintext to the cloud server CS_1.

5.2.3 Similarity Range Query Processing

On receiving $E(\mathcal{S})$ and $E(T)$, the cloud servers can offer similarity range query services to authorized query doctors. An authorized query doctor can launch a query request (q, k) as follows.

• ***Step 1:*** The query doctor generates a hash token t, $l_{b'}$ subsequences partitioned from the query q and complete the hash token genertaion stage in the similarity range query (as described in Section 3.3.2).

• ***Step 2:*** The query doctor encrypts the query token, which contains the hash token t, the query sequence $q = q_1, q_2, ..., q_{l_q}$, and the threshold value k. First, the query doctor encrypts the hash token with $h_K(\cdot)$ as $E(t) = (h_K(q_{sub_{b'}}), l_{sub_{b'}})$ where $1 \leq b' \leq l_{b'}$. Then the query doctor uses pk to encrypt the query sequence q as $E(q) = E(q_1), E(q_2), ..., E(q_{l_q})$. Finally, the query doctor encrypts the threshold value k as $E(k)$.

• ***Step 3:*** The query doctor sends $\{E(q), E(t), E(k)\}$ to CS_1 through a secure channel.

• ***Step 4:*** On receiving $\{E(q), E(t), E(k)\}$, CS_1 and CS_2 corporate to search $E(T)$ and $E(\mathcal{S})$ for the query result. Specifically, the searching process on cloud servers contains filtration stage and verification stage.

Filtration stage: In the filtration stage, CS_1 searches $E(T)$ for a candidate set $\mathcal{C}$ with encrypted identities. The filtration stage over an encrypted hash table is similar to that over a hash table in plaintext. Instead of searching $q_{sub_{b'}}$, CS_1 searches $h_K(q_{sub_{b'}})$ to lookup subsequence buckets in $E(T)$. Since two genomic sequences

may share more than one common subsequence, CS_1 then sends $\mathcal{C}$ to CS_2 to remove duplicate identities in the candidate set.

On receiving $\mathcal{C}$, CS_2 decrypts encrypted identities in the set by the secret key sk and gets a set of identities in plaintext. CS_2 forms a set $\mathcal{C}'$ in which duplicate identities are removed. Finally, CS_2 sends $\mathcal{C}'$ back to CS_1.

On receiving $\mathcal{C}'$, CS_1 checks the length of s_i to filter the sequences in the dataset again. For each identity id_i in the set $\mathcal{C}'$, if $||E(q)| - |E(s_i)|| > k$, CS_1 removes id_i from $\mathcal{C}'$.

Verification stage: In this process, CS_1 verifies the distance between s_i and q as follows:

(1) For all identities id_i in set $\mathcal{C}'$, CS_1 and CS_2 corporate to compute the encrypted distance between s_i and q, i.e., $E(d(s_i, q))$. The edit distance computation algorithm over encrypted sequences is similar to that over plaintext (described in Section 3.1). Differently, since both q and s_i are encrypted, the algorithm will compute an encrypted distance matrix. Based on the edit distance calculation algorithm described in Section 3.1, computing each element of the encrypted distance matrix requires three types of basic operations over encrypted data, i.e., addition, minimum value computation and equality test. Addition computation between two ciphertexts can be completed based on homomorphic properties of the Paillier encryption technique, while equality test computation and a minimum value computation can be handled by the minimum value computation protocol and the equality test protocol (described in 5.1), respectively. Finally, CS_1 can obtain $E(d(s_i, q))$.

(2) CS_1 and CS_2 corporate together to check if $d(s_i, q) \leq k$ or not with $E(d(s_i, q))$ and $E(k)$. They do the work similar to the minimum value computation protocol, except CS_2 returns 0 if $d(s_i, q) \leq k$ and returns 1 otherwise. Then, CS_1 puts valid sequences' plaintext identities in a set $\mathcal{O}$.

• ***Step 5:*** Finally, CS_1 sends the set $\mathcal{O}$ to the query doctor.

5.2.4 Dynamic Update of Dataset

We present how the hash table can support add, delete, and renew genomic sequences s_a with length l_a in the dataset.

5.2.4.1 Adding a Sequence into The Dataset

To add a new sequence s_a with identity id_a into $E(\mathcal{S})$, HC first encrypts s_a as $E(s_a)$ with Paillier encryption technique. Then it follows step 1, 2 and 3 in the hash table generation process in Section 3.3 to partition s_a into l_b subsequences $(s_{a:sub_b}, l_{a:sub_b})$ where $1 \leq b \leq l_b$ and $rank_{min} = 1$. Then, HC hashes $s_{a:sub_b}$ as $h_K(s_{a:sub_b})$ and encrypts id_a as $E(id_a)$. Then, HC sends $\{(h_K(s_{a:sub_b}), l_{a:sub_b}), E(id_a)\}$ and $\{id_a, E(s_a)\}$ to CS_1. After receiving them, CS_1 adds $E(id_a)$ into subsequence buckets $T_{l_{a:sub_b}}(s_{a:sub_b})$ and $\{id_a, E(s_a)\}$ into $E(\mathcal{S})$.

5.2.4.2 Deleting a Sequence from The Dataset

To delete an old sequence s_a with identity id_a from $E(\mathcal{S})$, HC first follows step 1, 2 and 3 in the hash table generation process in Section 3.3 to partition s_a into l_b subsequences $(s_{a:sub_b}, l_{a:sub_b})$ where $1 \leq b \leq l_b$ with $rank_{min} = 1$. Then, HC hashes $s_{a:sub_b}$ as $h_K(s_{a:sub_b})$. After that, HC sends $(h_K(s_{a:sub_b}), l_{a:sub_b})$ and id_a to CS_1. After receiving them, CS_1 first removes $\{id_a, E(s_a)\}$ from $E(\mathcal{S})$. Then, it searches subsequence buckets $T_{l_{a:sub_b}}(s_{a:sub_b})$ and sends all encrypted identities in the buckects back to HC. HC decrypts the elements in the buckets and removes all id_a. Finally, HC encrypts the updated buckets and sends them to CS_1.

5.2.4.3 Renewing a Sequence in The Dataset

To renew a sequence in the dataset, HC follows the steps of deleting the original sequence from the dataset and then adds the new sequence.

Remark: To prevent CS_1 from linking the added and deleted sequences to their subsequences in the hash table, we suggest the data owner add/delete multiple sequences at once.

Chapter 6

Security Analysis

In this chapter, we analyze the security of our proposed scheme. Specifically, we first analyze the security of the proposed two privacy-preserving protocols. Then, following the security requirements discussed in Chapter 4, we will show that the dataset and query sequences are privacy-preserving.

6.1 Security of Privacy-Preserving Protocols

As described in Section 5.1, two privacy-preserving protocols are designed between two honest-but-curious and non-collusive servers, i.e., CS_1 and CS_2. Since the Paillier encryption technique secures the communication between the two servers, we focus on the security issues related to CS_1 and CS_2.

6.1.1 Security of Minimum Value Computation Protocol

As detailed in Section 5.1, given two ciphertexts $E(m_1)$ and $E(m_2)$, the minimum value computation protocol can obtain $E(\min\{m_1, m_2\})$. The two cloud servers, CS_1 and CS_2, may be curious about the plaintexts (i.e., m_1, m_2 and $\min\{m_1, m_2\}$). We show that CS_1 and CS_2 cannot obtain the plaintexts as follows.

i) Before running the protocol, CS_1 can obtain $E(m_1)$ and $E(m_2)$. After running the protocol, it can further obtain ciphertexts returned from CS_2 and get $E(\min\{m_1, m_2\})$ based on homomorphic properties. Since it does not have the secret key, it cannot obtain the plaintexts of the ciphertexts.

ii) After running the protocol, CS_2 can only obtain $E(\mathtt{m}')$ and $E(M)$. CS_2 can also decrypt the ciphertexts, and get $\mathtt{m}'$ and M. Since only the values of random numbers and $m_1 - m_2$ are used in the protocol, CS_2 has no idea on how the CS_1 calculates $\mathtt{m}'$ or M, CS_2 will not be able to obtain m_1 or m_2.
Therefore, the minimum value computation protocol is privacy-preserving.

6.1.2 Security of Equality Test Protocol

As detailed in Section 5.1, given two ciphertexts $E(m_1)$ and $E(m_2)$, the equality test protocol can check if $m_1 = m_2$ or not. The two servers, CS_1 and CS_2, may be curious about the plaintexts (i.e., m_1 or m_2). We show that CS_1 and CS_2 cannot obtain the plaintexts as follows.

i) Before running the protocol, CS_1 can obtain $E(m_1)$ and $E(m_2)$. After running the protocol, it can further obtain the ciphertext returned from CS_2 . Since it does not have a secret key, it cannot obtain any plaintext.

ii) After running the protocol, CS_2 can only obtain $E(\mathtt{m})$ and $\mathtt{m}$. Since only a random number and $m_1 - m_2$ are used in the protocol, and CS_2 has no idea how the CS_1 calculates $\mathtt{m}$, CS_2 will not obtain the plaintexts m_1 or m_2.
Therefore, the equality test protocol is privacy-preserving.

6.2 Security of Our Scheme

As described in Section 4.2, our scheme is designed based on two honest-but-curious and non-colluding cloud servers, namely CS_1 and CS_2, responsible for storing outsourced data and offering similarity range query services. Both cloud servers will honestly do their work, but they may be curious about the plaintext of genomic sequences in dataset and query sequences. Based on the privacy-preservation of the two protocols, we show that our scheme can preserve the privacy of the dataset and query sequences.

6.2.1 Security of Plaintexts in Dataset

In our scheme, CS_1 stores an encrypted dataset $E(\mathcal{S})$ and an encrypted hash table $E(T)$. We show that CS_1 and CS_2 will not be able to obtain the plaintexts of original sequences in the dataset as follows.

i) Before processing any query, CS_1 only has an encrypted dataset and an encrypted hash table. Since CS_1 has no access to secret key sk, the cloud server cannot decrypt any ciphertext. Therefore, CS_1 does not have enough information to obtain the plaintexts of sequences in the dataset before receiving queries. After conducting queries, CS_1 further gets outputs of the privacy-preserving protocols. Since CS_1 does not have sk and the edit distance is calculated over encrypted sequences, CS_1 cannot obtain further information. Meanwhile, the indexes of subsequence buckets in the hash table are a hash value generated from the function $h_k(\cdot)$. CS_1 has no idea on the secret key K, it cannot obtain the plaintexts of the sequences in the dataset.

ii) CS_2 has access to sk but can only access messages when running privacy-preserving

protocols. Then, the security of two privacy-preserving protocols can guarantee that CS_2 cannot obtain the plaintexts of sequences in the dataset.
Thus, the plaintexts of sequences in the dataset are privacy-preserving.

6.2.2 Security of Plaintexts of Query Sequences

In our scheme, the two cloud servers, namely CS_1 and CS_2, may be curious about the plaintexts of the query sequence q. As described in Section 5.2, query doctor launches a query request by sending $\{E(q), E(t), E(k)\}$ to CS_1. We show that CS_1 and CS_2 will not get the plaintext of q as follows.

i) After conducting queries, CS_1 can access the encrypted query sequence $E(q)$. Then, the security of the Paillier encryption can guarantee that CS_1 cannot obtain the plaintext of q. Meanwhile, the hash token sent to CS_1 are hash values of q. Since CS_1 does not have the secret key K, it cannot obtain the plaintexts of sequences related to q.

ii) CS_2 only has access to ciphertexts in privacy-preserving protocols. Since both protocols are privacy-preserving, CS_2 cannot obtain the plaintexts of query sequences. Thus, plaintexts of query sequences are privacy-preserving.

Chapter 7

Performance Evaluation

We evaluate the performance of our scheme from data outsourcing, query token generation, similarity range query processing, adding a sequence into the dataset and deleting a sequence from the dataset. We do not show the overall performance of updating a sequence in this chapter, as updating a sequence consists of adding and deleting a sequence.

7.1 Experimental Setting

We implemented our scheme in Java and tested with an experimental environment shown in TABLE 7.1. The machine used for performance test has Intern(R) Core(TM) i5 CPU @2.3GHz with 8G RAM. The security parameter κ of the Paillier homomorphic encryption technique is set to be $\kappa = 512$, and another parameter in the model n is set to be $n = 3$, which is the same n value Zhang et al. used in their evaluation [54]. The secret key for $h_K(\cdot)$ is set as $K =$"HashTableHmacKey" and the function $h_K(\cdot)$ used for evaluation is an SHA-256 function. We evaluate the performance over a dataset chosen from Bio project PRJNA297868 on GenBank [32]. The evaluated dataset contains 3700 sequences with an average length of 830. Since all sequences in the dataset are represented in nucleotides, we replace nucleotides

with integers. In addition, for each experiment, we conduct multiple times, and the average result is reported.

Table 7.1: Experimental environment

Parameter	Setting
Evaluating CPU	Intern(R) Core(TM) i5 CPU @2.3GHz
Evaluating RAM	8GB
Programming Language	Java
Paillier's parameter	$\kappa = 512$
Parameter for Hash Table Generation	$n = 3$
Parameter for Key-based Hash function	$K =$"HashTableHmacKey" Key-based SHA-256 function
Dataset	3700 sequences average length of 830

7.2 Experimental Results

We show the experimental results of data outsourcing, query token generation, similarity range query processing, adding a sequence into and deleting a sequence from the dataset.

7.2.1 Data Outsourcing

In Fig. 7.1, we plot the computational cost of data outsourcing with N, which is the number of sequences in the dataset. Since the dataset we used to perform the evaluation has 7300 sequences with an average length of 830, in this experiment, the parameters are set as $N = \{1000, 1500, 2000, 2500, 3000\}$ and $l_i = \{100, 200, 300\}$ respectively. We can see that with the same length of sequences in the dataset l_i, the computational cost of data outsourcing will increase as N increases in Fig. 7.1. As N increases, the size of the hash table and dataset increases, therefore the more time is needed to encrypt and index the dataset. Meanwhile, regardless of the increase

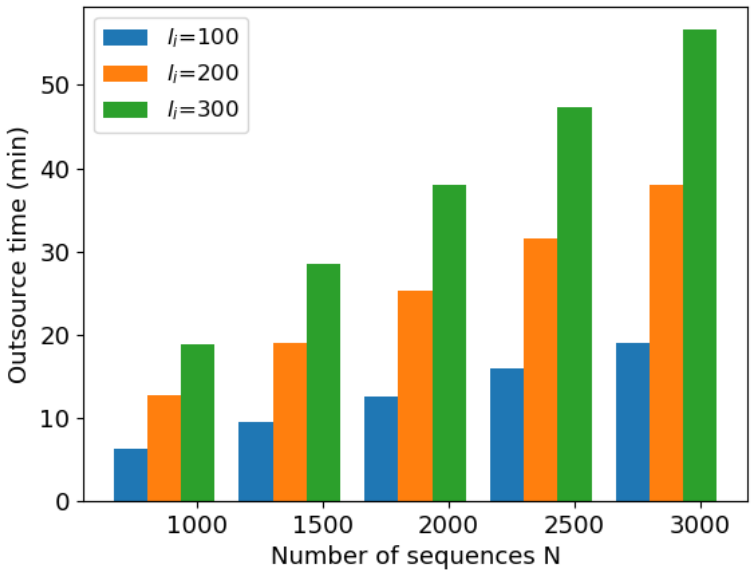

Figure 7.1: Outsourcing time

of N, the time to process one sequence of length 300, 200 and 100 needs 0.018 min, 0.012 min and 0.006 min, respectively.

Meanwhile, this figure shows that the computational cost increases as l_i increases with the same N. This is because it needs to go through the whole sequence several times when indexing the dataset. The longer the sequences are, the more time it will cost to index the dataset.

7.2.2 Query Token Generation

As described in Section 5.2, the computational cost of query token generation is mainly related to the length of the query sequence l_q. We randomly pick a sequence from the dataset as the query sequence in this experiment. We evaluate the computational cost of our scheme by setting $l_q = \{200, 300, 400, 500, 600, 700\}$. We set $k = 100$ for the experiment because we can have at least one identity in the query result for the dataset and the query sequence we pick from the dataset. TABLE 7.2 shows that when l_q becomes larger, it takes longer to generate a query token when

the threshold value $k = 100$. This is because it needs to go through the query sequence several times when generating a query token. The longer the query sequences are, the more time it will cost to generate a query token.

Table 7.2: Query token generation time with $k = 100$

l_q	200	300	400	500	600	700
Time (ms)	853	1206	1661	2059	2515	2814

7.2.3 Similarity Range Query Processing

We evaluate the computational cost of the similarity range query processing. As described in Section 5.2, the similarity range query processing consists of a filtration stage and a verification stage. The computational cost of similarity range query processing is affected by parameters $\{N, l_i, k\}$. In our evaluation, to improve the query efficiency in the verification stage, we check if the distance exceeds the threshold for every five nucleotides instead of calculating the accurate edit distance between the whole candidate sequence and the query sequence before comparing the distance and threshold value. Detailed experimental results are shown as follows.

Table 7.3: Number of identities in candidate set $\mathcal{C}'$

	$k = 100$	$k = 150$	$k = 200$	$k = 250$	$k = 300$
N=1000	246	366	371	493	621
N=1500	381	564	570	758	919
N=2000	510	553	765	1017	1179
N=2500	637	946	960	1275	1481
N=3000	764	1134	1153	1531	1799

• *Number of sequences in the dataset N:* In Fig. 7.2, we plot the computational cost of the similarity range query with different numbers of sequences in the dataset. Since the dataset we used to perform the evaluation has 7300 sequences, in this experiment, the parameters are set as $N = \{1000, 1500, 2000, 2500, 3000\}$. We also set the parameter $k = 100$, the same value we use to evaluate the query token

generation. Meanwhile, we can get at least one identity in the query results with different N in the experiment. From this figure, we can see that the computational cost will increase if N increases. This is because the number of candidate sequences to be verified in the verification stage will increase when N increases, as shown in TABLE 7.3.

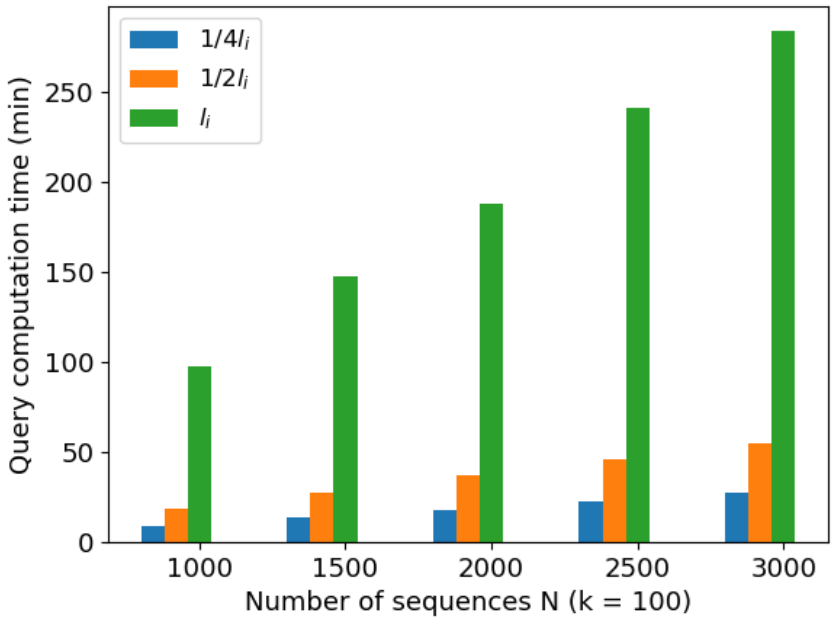

Figure 7.2: Query processing time with N

• *Length of sequences in the dataset l_i:* In Fig. 7.2, we plot the computational cost of the similarity range query with different lengths of genomic sequences s_i in the dataset. In this experiment, l_i is set as a quarter, half or equal to the length of sequences in the dataset, respectively. The threshold value is set as $k = 100$, the same as the last experiment. From this figure, we can see that the computational cost will increase with l_i. This is because a sequence s_i with a larger l_i may be partitioned into more subsequences, leading to the increase of computational cost in the filtration stage. Meanwhile, by comparing the computational costs with and without filtration, Fig. 7.3 demonstrates the effectiveness of our filtration strategy.

• *Threshold value k:* In Fig. 7.3, we plot the computational cost of the similarity

range query varying with the threshold value k. In this experiment, to validate the efficiency of the filtration stage, we also plot the computational cost of the similarity range query without filtration. In this experiment, the parameters are set as $N = 1000$ with $k = \{100, 150, 200, 250, 300\}$. From this figure, we can see that the computational cost will increase if k increases. This is because the number of candidate sequences to be verified in the verification stage increases when k increases (as shown in TABLE 7.3).

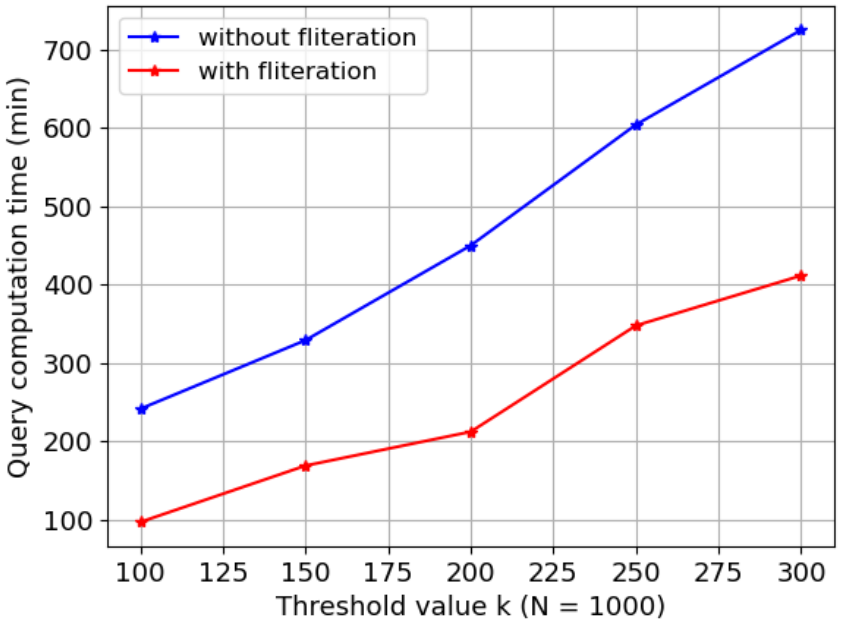

Figure 7.3: Query processing with and without filtration

7.2.4 Adding a Sequence into The Dataset

As described in Section 5.2, the computational cost of adding a sequence s_a with length l_a into the dataset relates to the length of the sequence. We evaluate the computational cost by setting $l_a = \{200, 300, 400, 500, 600, 700\}$, the same set of we used in the experiment of the query token generation. TABLE 7.4 shows that when l_a becomes larger, it takes longer to add a sequence into the dataset (when $N = 1000$). There are three reasons that l_a affects the computational cost. Firstly, the longer

a sequence is, the more time is needed to encrypt the sequence. Secondly, when a sequence is longer, we need a longer time to partition the sequence. Meanwhile, a longer sequence may have more subsequences, and therefore more time is needed to encrypt and insert the identities.

Table 7.4: Time to add a sequence with $N = 1000$

l_a	200	300	400	500	600	700
Time (ms)	1038	1489	2046	2526	3113	3498

Table 7.5: Time to delete a sequence with $N = 1000$

l_a	200	300	400	500	600	700
Time (min)	11.05	15.32	22.44	25.17	32.87	37.36

7.2.5 Deleting a Sequence from The Dataset

As described in Section 5.2, the computational cost of deleting a sequence s_a is affected by the length of the sequence l_a. We evaluate the computational cost by setting $l_a = \{200, 300, 400, 500, 600, 700\}$, the same parameter we used in the last experiment. TABLE 7.5 shows that when l_a becomes larger, it takes longer to delete a sequence from the dataset (when $N = 1000$). This is because when l_a is longer, partitioning the sequence s_a will take more time, and more elements in subsequence buckets that are related to s_a need to be decrypted and re-encrypted.

Chapter 8

Conclusion And Future Work

This chapter summarizes our contribution in similarity queries over genomic sequences and discusses extensions and directions towards future work.

8.1 Conclusion

This thesis proposed an efficient and privacy-preserving similarity range query scheme over encrypted genomic sequences. The scheme supports dynamic updates of the dataset with the hash table as an index and provides flexibility in dataset maintenance. Furthermore, the usage of homomorphic encryption preserves the privacy of the outsourced dataset and the query sequences. Specifically, based on the Paillier encryption technique, we first introduced two privacy-preserving protocols to preserve the privacy of edit distance computation and similarity range query processing. Then, we applied these privacy-preserving protocols to preserve the privacy of the hash table-based similarity range query algorithm and proposed our scheme. We analyzed the security of our scheme, and the results indicate that our scheme can preserve the privacy of the dataset and query sequences. In addition, we conducted experiments to evaluate the performance of our scheme. The experiments validate that our scheme is computationally efficient.

8.2 Future Work

In Section 2.3.2, we summarize state-of-art solutions towards secure outsourcing model-based schemes and discuss open questions. Based on these discussions and our model, we discuss some extensions and directions towards future work in this section.

• Since we are always facing the problem of trading off between privacy and efficiency, improving efficiency leads to less protection of the scheme. However, genomic sequences are highly sensitive. In our future work, we will focus on the index structure and design protocols to improve the efficiency of schemes.

• In our scheme, we handle genomic sequences as a string and process them based on their property as a string of nucleotides. In biology, genomic sequences have other special characteristics like SNP. These biological characteristics can help improve the performance or help preserve the scheme's privacy. For example, SNP can help to reduce the amount of dataset. In our future work, we want to design efficient and privacy-preserving similarity query schemes over genomic sequences based on the characteristics of genomic sequences.

• In our scheme, we use edit distance as a similarity measure. The similarity measure is critical in similarity queries and affects the design of the index and privacy-preserving protocols. With the development of personalized medicine, providing a variety of similarity measures is a trend. Based on similarity measures, the use of encryption techniques might be different. In our future work, we want to apply different similarity measures on similarity query schemes over genomic sequences.

Bibliography

[1] Omar Abood and Shawkat Guirguis, *A survey on cryptography algorithms*, International Journal of Scientific and Research Publications **8** (2018), 495–516.

[2] Miklós Ajtai, *Generating hard instances of lattice problems (extended abstract)*, Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996 (Gary L. Miller, ed.), ACM, 1996, pp. 99–108.

[3] Mete Akgün, Ali Osman Bayrak, Bugra Ozer, and Mahmut Samil Sagiroglu, *Privacy preserving processing of genomic data: A survey*, J. Biomed. Informatics **56** (2015), 103–111.

[4] C Alliance, *Security guidance for critical areas of focus in cloud computing v4. 0*, Cloud Security Alliance (2017), 1–152.

[5] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin, *Privacy-preserving search of similar patients in genomic data*, Proc. Priv. Enhancing Technol. **2018** (2018), no. 4, 104–124.

[6] Mikhail J. Atallah and Jiangtao Li, *Secure outsourcing of sequence comparisons*, Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004, Revised Selected Papers, Lecture Notes in Computer Science, vol. 3424, 2004, pp. 63–78.

[7] Erman Ayday, Jean Louis Raisaro, Urs Hengartner, Adam Molyneaux, and Jean-Pierre Hubaux, *Privacy-preserving processing of raw genomic data*, Data Privacy Management and Autonomous Spontaneous Security - 8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers, vol. 8247, 2013, pp. 133–147.

[8] Md. Momin Al Aziz, Dima Alhadidi, and Noman Mohammed, *Secure approximation of edit distance on genomic data*, BMC Medical Genomics **10** (2017).

[9] Shahram Bakhtiari, Reihaneh Safavi-Naini, Josef Pieprzyk, et al., *Cryptographic hash functions: A survey*, Tech. report, Citeseer, 1995.

[10] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik, *Countering GATTACA: efficient and secure testing of fully-sequenced human genomes*, Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011, ACM, 2011, pp. 691–702.

[11] Marina Blanton and Mehrdad Aliasgari, *Secure outsourcing of DNA searching via finite automata*, Data and Applications Security and Privacy XXIV, 24th Annual IFIP WG 11.3 Working Conference, Rome, Italy, June 21-23, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6166, 2010, pp. 49–64.

[12] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim, *Evaluating 2-dnf formulas on ciphertexts*, Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings, Lecture Notes in Computer Science, vol. 3378, 2005, pp. 325–341.

[13] Deyan Chen and Hong Zhao, *Data security and privacy protection issues in cloud computing*, 2012 International Conference on Computer Science and Electronics Engineering, vol. 1, IEEE, 2012, pp. 647–651.

[14] Yangyi Chen, Bo Peng, XiaoFeng Wang, and Haixu Tang, *Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds*, 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012, The Internet Society, 2012.

[15] Ke Cheng, Yantian Hou, and Liangmin Wang, *Secure similar sequence query on outsourced genomic data*, Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018, ACM, 2018, pp. 237–251.

[16] Jung Hee Cheon, Miran Kim, and Kristin E. Lauter, *Homomorphic computation of edit distance*, Financial Cryptography and Data Security - FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers, Lecture Notes in Computer Science, vol. 8976, 2015, pp. 194–212.

[17] Caroline Fontaine and Fabien Galand, *A survey of homomorphic encryption for nonspecialists*, EURASIP J. Inf. Secur. **2007** (2007).

[18] Yunguo Guan, Rongxing Lu, Yandong Zheng, Songnian Zhang, Jun Shao, and Guiyi Wei, *Toward privacy-preserving cybertwin-based spatio-temporal keyword query for its in 6g era*, IEEE Internet of Things Journal (2021), 1–1, doi=10.1109/JIOT.2021.3096674.

[19] National Human Genome Research Institute, *A brief guide to genomics*, Aug 2020.

[20] ———, *Chromosomes fact sheet*, Aug 2020.

[21] Sameeh A. Jassim and Alaa K. Farhan, *A survey on stream ciphers for constrained environments*, 2021 1st Babylon International Conference on Information Technology and Science (BICITS), 2021, pp. 228–233.

[22] Somesh Jha, Louis Kruger, and Vitaly Shmatikov, *Towards practical privacy for genomic computation*, 2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA, IEEE Computer Society, 2008, pp. 216–230.

[23] Lynda Kacha and Abdelhafid Zitouni, *An overview on data security in cloud computing*, CoRR **abs/1812.09053** (2018).

[24] Atul Kahate, *Cryptography and network security*, Tata McGraw-Hill Education, 2013.

[25] Jonathan Katz, Amit Sahai, and Brent Waters, *Predicate encryption supporting disjunctions, polynomial equations, and inner products*, Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings, Lecture Notes in Computer Science, vol. 4965, Springer, 2008, pp. 146–162.

[26] Lori M. Kaufman, *Data security in the world of cloud computing*, IEEE Secur. Priv. **7** (2009), no. 4, 61–64.

[27] Ryan Ko and Kim-Kwang Raymond Choo (eds.), *The cloud security ecosystem: technical, legal, business and management issues*, Syngress is an imprint of Elsevier, 2015.

[28] Sabine Lang and Björn Ommer, *Attesting similarity: Supporting the organization and study of art image collections with computer vision*, Digit. Scholarsh. Humanit. **33** (2018), no. 4, 845–856.

[29] Vladimir I Levenshtein et al., *Binary codes capable of correcting deletions, insertions, and reversals*, Soviet physics doklady, vol. 10, Soviet Union, 1966, pp. 707–710.

[30] Md Safiur Rahman Mahdi, Mohammad Zahidul Hasan, and Noman Mohammed, *Secure sequence similarity search on encrypted genomic data*, Proceedings of the Second IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies, CHASE 2017, Philadelphia, PA, USA, July 17-19, 2017, IEEE Computer Society / ACM, 2017, pp. 205–213.

[31] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan, *Can homomorphic encryption be practical?*, Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011 (Christian Cachin and Thomas Ristenpart, eds.), ACM, 2011, pp. 113–124.

[32] National Center for Biotechnology Information, *Influenza a virus*, 2015.

[33] Gonzalo Navarro, *A guided tour to approximate string matching*, ACM Comput. Surv. **33** (2001), no. 1, 31–88.

[34] Pascal Paillier, *Public-key cryptosystems based on composite degree residuosity classes*, Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, Lecture Notes in Computer Science, vol. 1592, 1999, pp. 223–238.

[35] Raluca A. Popa, Frank H. Li, and Nickolai Zeldovich, *An ideal-security protocol for order-preserving encoding*, 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013, IEEE Computer Society, 2013, pp. 463–477.

[36] Abdalbasit Mohammed Qadir and Nurhayat Varol, *A review paper on cryptography*, 7th International Symposium on Digital Forensics and Security, ISDFS 2019, Barcelos, Portugal, June 10-12, 2019, IEEE, 2019, pp. 1–6.

[37] Thomas Schneider and Oleksandr Tkachenko, *Towards efficient privacy-preserving similar sequence queries on outsourced genomic databases*, Proceedings of the 2018 Workshop on Privacy in the Electronic Society, WPES@CCS 2018, Toronto, ON, Canada, October 15-19, 2018, ACM, 2018, pp. 71–75.

[38] Pulkit Singh, Bibhudendra Acharya, and Rahul Kumar Chaurasiya, *A comparative survey on lightweight block ciphers for resource constrained applications*, Int. J. High Perform. Syst. Archit. **8** (2019), no. 4, 250–270.

[39] Douglas R Stinson, *Cryptography: theory and practice*, Chapman and Hall/CRC, 2005.

[40] Yunchuan Sun, Junsheng Zhang, Yongping Xiong, and Guangyu Zhu, *Data security and privacy in cloud computing*, International Journal of Distributed Sensor Networks **10** (2014), no. 7, 190903.

[41] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik, *Privacy preserving error resilient dna searching through oblivious automata*, Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007, ACM, 2007, pp. 519–528.

[42] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al., *The sequence of the human genome*, science **291** (2001), no. 5507, 1304–1351.

[43] Robert A. Wagner and Michael J. Fischer, *The string-to-string correction problem*, J. ACM **21** (1974), no. 1, 168–173.

[44] Bing Wang, Wei Song, Wenjing Lou, and Y. Thomas Hou, *Privacy-preserving pattern matching over encrypted genetic data in cloud computing*, 2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, May 1-4, 2017, IEEE, 2017, pp. 1–9.

[45] Boyang Wang, Yantian Hou, and Ming Li, *Practical and secure nearest neighbor search on encrypted large-scale data*, IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9.

[46] Rui Wang, XiaoFeng Wang, Zhou Li, Haixu Tang, Michael K. Reiter, and Zheng Dong, *Privacy-preserving genomic computation through program specialization*, Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009 (Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, eds.), ACM, 2009, pp. 338–347.

[47] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu, *Efficient genome-wide, privacy-preserving similar patient query based on private edit distance*, Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015, ACM, 2015, pp. 492–503.

[48] Wai Kit Wong, David Wai-Lok Cheung, Ben Kao, and Nikos Mamoulis, *Secure knn computation on encrypted databases*, Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009 (Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, eds.), ACM, 2009, pp. 139–152.

[49] Andrew Chi-Chih Yao, *Protocols for secure computations (extended abstract)*, 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982, IEEE Computer Society, 1982, pp. 160–164.

[50] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba, *Secure pattern matching using somewhat homomorphic encryption*, CCSW'13, Proceedings of the 2013 ACM Cloud Computing Security Workshop, Co-located with CCS 2013, Berlin, Germany, November 4, 2013, ACM, 2013, pp. 65–76.

[51] Haitao Yuan and Guoliang Li, *Distributed in-memory trajectory similarity search and join on road network*, 2019 IEEE 35th international conference on data engineering (ICDE), IEEE, 2019, pp. 1262–1273.

[52] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko, *Similarity search: The metric space approach (advances in database systems)*, Springer-Verlag, Berlin, Heidelberg, 2005.

[53] Haoyu Zhang and Qin Zhang, *Minjoin: Efficient edit similarity joins via local hash minima*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, ACM, 2019, pp. 1093–1103.

[54] ———, *Minsearch: An efficient algorithm for similarity search under edit distance*, KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020, ACM, 2020, pp. 566–576.

[55] Songnian Zhang, Suprio Ray, Rongxing Lu, Yandong Zheng, Yunguo Guan, and Jun Shao, *Achieving efficient and privacy-preserving dynamic skyline query*

in online medical diagnosis, IEEE Internet of Things Journal (2021), 1–1, doi=10.1109/JIOT.2021.3117933.

[56] Yandong Zheng and Rongxing Lu, *Efficient privacy-preserving similarity range query based on pre-computed distances in ehealthcare*, IEEE Global Communications Conference, GLOBECOM 2020, Virtual Event, Taiwan, December 7-11, 2020, IEEE, 2020, pp. 1–6.

[57] Yandong Zheng, Rongxing Lu, Yunguo Guan, Jun Shao, and Hui Zhu, *Efficient and privacy-preserving similarity range query over encrypted time series data*, IEEE Transactions on Dependable and Secure Computing (2021), 1–1.

[58] Yandong Zheng, Rongxing Lu, Yunguo Guan, Jun Shao, and Hui Zhu, *Efficient privacy-preserving similarity range query with quadsector tree in ehealthcare*, IEEE Transactions on Services Computing (2021), 1–1, doi=10.1109/TSC.2021.3081350.

[59] Ruiyu Zhu and Yan Huang, *Efficient privacy-preserving edit distance and beyond*, IACR Cryptol. ePrint Arch. (2017), 683.

Vita

Candidate's full name: Jiacheng Jin

University attended (with dates and degrees obtained):

- Bachlor of Computer Science, First Class Honor, University of New Brunswick, NB, Canada, 2015-2020

- Master of Computer Science, University of New Brunswick, NB, Canada, 2020-2022

Conference Presentations:
J. Jin, Y. Zheng and P. Xiong, "EPSim-GS: Efficient and Privacy-Preserving Similarity Range Query over Genomic Sequences," 2021 18th International Conference on Privacy, Security and Trust (PST), 2021, pp. 1-10, doi: 10.1109/PST52912.2021.9647830.