8-18-2020 1:00 PM

A New Approach for Homomorphic Encryption with Secure Function Evaluation on Genomic Data

Mounika Pratapa, *The University of Western Ontario*

Supervisor: Essex, Aleksander E., *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Engineering Science degree in Electrical and Computer Engineering

Abstract

Additively homomorphic encryption is a public-key primitive allowing a sum to be computed on encrypted values. Although limited in functionality, additive schemes have been an essential tool in the private function evaluation toolbox for decades. They are typically faster and more straightforward to implement relative to their fully homomorphic counterparts, and more efficient than garbled circuits in certain applications. This thesis presents a novel method for extending the functionality of additively homomorphic encryption to allow the private evaluation of functions of restricted domain. Provided the encrypted sum falls within the restricted domain, the function can be homomorphically evaluated "for free" in a single public-key operation. We will describe an algorithm for encoding private functions into the public-keys of two well-known additive cryptosystems.

We extend this scheme to an application in the field of pharmacogenomics called Similar Patient Query. With the advent of human genome project, there is a tremendous availability of genomic data opening the door for a possibility of many advances in the field of medicine. Precision medicine is one such application where a patient is administered drugs based on their genetic makeup. If the genomic data is not kept private, it can lead to several information frauds, so it needs to be encrypted. To tap the full potential of the encrypted genomic data, we need to perform computations on it without compromising its security. For SPQ, we pick a *query* genome and compare it across a *hospital* data base, to find patients similar to that of the *query* and use the information to apply precision medicine, all of this is carried out under privacy preserving settings in the presence of a semi-honest adversary in a single transaction.

Keywords: Secure Function Evaluation, Public Key Cryptosystem, Homomorphic Encryption, Prime Number Generation, Genomic Data, Secure Genomic Computations, Similar Patient Query

Summary For Lay Audience

In an increasingly data driven world, information availability and its distribution are exploding exponentially. The access to personal information has also invited a lot of threats such as hacking, denial of service attacks, injecting malicious code or gaining illegal access to confidential information. As these threats are becoming a matter-of-course with ever increasing sophistication- planning, implementing and maintaining information security systems at any organization is becoming a challenging task. There are a lot of systems in place to protect data at various levels such as physical, application, network and cloud. There are also myriad legal regulations and compliances that help the organizations to improve their security strategy by providing guidelines depending on the type of data these organizations are handling with. Violating these standards can result in severe penalties such as fines and law suits, or even worse, personal information breach. The most challenging aspect of information security thus lies in making the best possible use of available data while ensuring privacy.

One of the popular approaches of storing information securely is to encrypt it or convert it into random looking numbers so when hackers have access to it, they will not be able to figure out what the data is about. Now, it is hard to perform computations on this random looking numbers. If we need to make use of the full potential of the data, we need to decode those random looking numbers. Nonetheless, there are methods in the field of computer science and pure math that enable us to perform analysis without breaking this data. To make the best use of data in an encrypted form, we use concepts from math and design functions in such a way that, we can perform computations on the encrypted data without having to decode. Our thesis presents one of such algorithms which help us perform computations without decoding the data. Once we perform computations and decode this data, we get the results in the same way as if we would get if the computations on plain text data.

We use these algorithms for applications in the field of medicine to conduct search across genomic data bases. Genomic databases are stored in a secure way to avoid leaking any sensitive information. Using our algorithms, we can search across these databases without decoding the data. We perform some mathematical functions and retrieve similarity scores and obtain similar records to that of our search query. This is used in the field of precision medicine where, we administer medication or therapy to a patient based on their genetic make up. Since

not all databases are available openly, the medical practitioners cannot tap the full potential of genomic data. Using our algorithm, we help the medical practitioners to access across all the databases even if encrypted, giving them tremendous potential to make advances in the field of precision medicine.

Acknowlegements

This thesis would have been impossible without the support and mentorship of my advisor, Dr. Aleksander Essex. For me who is a novice in the field of Cryptography to produce a meaningful thesis like this is something I have never imagined. And it is an indisputable truth that his constant edification and encouragement are solely responsible. Dr. Essex, whose amazing intellect can only be surpassed by his unwavering optimism, nurtured me in every stage of this research and made sure I never faltered. I would like to wholeheartedly thank him for his never-ending guidance in the completion of this thesis.

To my husband Sunil Borra for his unlimited love and support through this academic journey, I could not have done this without him. To my best friend Santhi Swaroop, who tirelessly motivated me, thank you very much. I would also like to thank my little brother Dr. Aditya Pratapa for being an inspiration and giving me general advice whenever I needed.

My parents Arundhati Nambi and ChandraShekar Pratapa have provided me unlimited positivity and encouragement through this process which kept me strong. I would like to thank my parents in-law Vakula Borra and Prakash Borra and my extended family for their love and support.

Finally I would also like to express my gratitude to all the faculty at Western University who provided me with tremendous knowledge.

Contents

List of Figures

List of Tables

List of Abbreviations, Symbols, and Nomenclature

Abbreviations:

SFE - Secure Function Evaluation

PFE - Private Function Evaluation

SMC - Secure Multiparty Computation

MPC - Multiparty Computation

SPQ - Similar Patient Query

HE - Homomorphic Encryption

PHE - Partial Homomorphic Encryption

FHE - Fully Homomorphic Encryption

GC - Garbled Circuit

UC - Universal Circuit

Mathematical Notations:

$\mathbb{Z}$ - Set of integers

$\mathbb{Z}_n$ - Set of integers modulo n

$\mathbb{Z}_n^*$ - Set of multiplicative group of integers modulo n

$\mathbb{P}$ - Set of primes

$p \mid a$ - p divides a

$p \nmid a$ - p does not divide a

gcd - Greatest Common Divisor

$a \equiv b \pmod{m}$ - a is congruent to b modulo m

$a^{-1} \pmod{p}$ - multiplicative inverse of $a \pmod{p}$

$x \xleftarrow{R} X$ - Indicates the fact that $x \in X$ is sampled independently and uniformly from X

Symbols related to Set Theory

$\forall$ - For all

$\exists$ - There Exists

$|$ - such that

$\in$ - Belongs to

$A \subseteq B$ - means A is a subset of or equal to B

Chapter 1

Introduction

In an increasingly data-driven world, information availability and its distribution are surging exponentially. The access to personal information has also invited a lot of threats such as hacking, denial of service attack, injecting malicious code, or gaining illegal access to confidential information. These threats are becoming a matter-of-course with ever-increasing sophistication due to which planning, implementing, and maintaining information security systems at any organization is becoming a challenging task. There are many systems in place to protect data at various levels like physical, application, network, and cloud. The myriad legal regulations and compliances help the organizations to improve their security strategy by providing guidelines depending on the type of data these organizations are handling. Violating these standards can result in severe penalties such as fines and lawsuits, or even worse, personal information breach. The most challenging aspect of information security thus lies in making the best possible use of available data while ensuring privacy.

The basic components of information security are often abbreviated as the CIA triad[22], which can be described as:

- **Confidentiality** of data is achieved only when the people who are authorized to access the data can do so.
- **Integrity** is the maintenance of data in its original state without modifying it unintentionally or maliciously.
- **Availability** is just a mirror image of Confidentiality, as confidentiality aims to hide

the message from unauthorized users, availability property ensures that the message is available to the authorized users.

While ensuring these basic components, the information security is maintained in various ways through physical, application and operational security. The most popular way of maintaining data security is through Cryptography, where we perform data encryption. Encryption is the method by which a message is converted into a secret code by hiding the original message. With respect to a protocol that uses data encryption, the letter *A* in CIA triad will be defined as **Authenticity** as opposed to availability. Authenticity means the ability to prove that the message is coming from the original sender. Sticking to the goals of Confidentiality, Integrity and Authenticity, data encryption is carried out by various cryptographic protocols, out of which this thesis deals with asymmetric or public key encryption. In public key encryption, we use mathematical concepts from number theory and convert the plain text message into some random looking numbers. Many researches have been conducted to make the best use of encrypted data without having to convert them to their original plain text format.

1.1 Motivation

The ability to perform computations on encrypted data has always been one of the most sought after challenges in the field of cryptography. Private computations on data are becoming increasingly significant as they have applications in many fields such as privacy preserving machine learning [54], private information retrieval [12], similarity search in private databases such as genotype and other medical data [56], online voting [1], auctions [13] and private credit checking [21]. The key idea behind secure computations is that two parties want to evaluate some function on their encrypted inputs without leaking any information about their input. When some special cases require privacy of the function to be evaluated, it is called private function evaluation (PFE). Generally, when a PFE is carried out, the function will be a private input of one of the parties and nothing can be leaked about this function to any adversary that does not hold this function. In this thesis, we shall introduce a cryptosystem that offers a novel approach for secure computations and we can use it for Private Function Evaluation in some limited applications. Current methods used for secure and private function include

	Garbled Circuits	Fully HE	Partial HE
Speed	Very Fast	Slow	Fast
Re-usability	No	Yes	Yes
Round Complexity	Constant	Constant	Constant
Communication Complexity	Proportional to size of the circuit	Independent	Independent

Table 1.1: Performance Comparison of existing approaches for SFE and PFE

Garbled Circuits and homomorphic encryption schemes. We opted for Partial Homomorphic Encryption (PHE) to solve the computations on encrypted data due to the several advantages it offers over Garbled Circuits (GC) and Fully Homomorphic Encryption schemes (FHE). PHE schemes are generally more efficient because they support only one type of operation: either addition or multiplication. FHE usually allows us to do unbounded number of homomorphic operations but it requires an expensive bootstrapping procedure. With respect to Garbled Circuits, the parties performing Secure or Private Function Evaluation need to interact many times for a single transaction, which makes them unsuitable for applications that may require minimum communication. The table 1.1 summarizes the properties of PHE, FHE and GC. From these properties, it can be established that Homomorphic Encryption has advantages of being reusable for any input where as one GC can be used only for one garbled input. Also, the communication complexity increases in GC with the increase in the depth of circuit function, whereas HE scheme's communication complexity remains independent. So despite the speed advantage offered by GC, we opted for Secure Function Evaluation using HE. In Homomorphic Encryption, partial HE is faster and of low cost compared to full HE, which makes it an ideal choice for Secure Function Evaluation.

1.2 Contributions

This thesis introduces a new framework where certain functions mapping Integer to Boolean co domain can be privately evaluated. Imagine some organization developed a special algorithm to identify a patient's reaction to particular medicine based on their genetic make up. Owing

to proprietary issues, the algorithm needs to be private. In special cases, some hospitals do not want to disclose elaborate answers as it may leak sensitive genetic information, rather they just want the parties to learn some Boolean answers such as "true" or "false". This is one of the several applications that the cryptosystem introduced through our research can be used for. The contributions of the thesis include:

- A new private key-generation algorithm that helps in generation of prime numbers that contain long, pre-defined sequences of Legendre symbols of integers modulo an odd, large prime.
- A secure evaluation scheme based on these arbitrary patterns of Legendre symbols that extends Additively Homomorphic Encryption schemes to privately evaluate functions with Boolean co-domain.
- A protocol for applying this new scheme by proving that all the evaluations can be performed freely and securely in a single public-key operation.
- Proof of security and hardness for the new scheme.
- Privacy proof for Secure Function Evaluation protocol in a two-party, honest but curious adversary setting.
- An application based on the new cryptosystem for carrying out Similar Patient Query using genomic data in a privacy preserving manner.
- Protocol for secure SPQ along with hardness proof, implementation, and results.

1.2.1 Black Box Methodology

The key generation protocol presented in this paper paves a way for a user to implement a generic methodology to evaluate some functions in a restricted domain without leaking the output or input. This can be extended across any public key cryptosystem that supports linear homomorphic encryption operations. To be precise, our scheme can be used like a black box across these cryptosystems - Okamoto Uchiyama, Paillier, Goldwasser-Micali, and DGK. All these have different sets of security parameters with respect to public keys, but all of them

encrypt the plain text messages by raising it to a generator g, which is one of the public key parameters. Mathematically, cipher texts obtained are in the form g^m, along with a hiding factor that varies from scheme to scheme. All these schemes carry out operations in modulus n which is a composite number formed as a multiple of two large primes, i.e., $n = pq$. Our scheme presents a novel method to generate one of these large primes, p. p is used to eliminate the hiding factor during decryption. We make use of simple properties from number theory and helps us generate the prime factor p, which would in no way alter any of the encryption or decryption steps of the aforementioned cryptosystems. But the special way of prime generation would help us dive into a new dimension during decryption process where private function evaluation can be carried out in restricted domains.

1.3 Organization of Thesis

The rest of the thesis is organized as follows:

- **Chapter 2** covers background on basics of Secure Multiparty Computation, Secure Function Evaluation, Private Function Evaluation and secure computation methods. It goes on to introduce the idea behind existing public key cryptosystems along with an introduction to Partial Homomorphic Encryption and describes homomorphisms under these encryption schemes. The last section in this chapter gives the key idea behind the cryptosystem.

- **Chapter 3** talks about important mathematical preliminaries necessary for the cryptosystem with some basic definitions and theorems necessary for prime number generation.

- **Chapter 4** focuses on the cryptosystem that is built using the basics from linear functional embeddings. For this, a special key generation algorithm will be provided, the encryption and decryption schemes will be the same as a standard cryptosystem that we would like to implement, i.e., either Okamoto-Uchiyama or Paillier.

- **Chapter 5** has two parts. The first part discusses the necessary definitions and theorems necessary to prove the security of our cryptosystem. Second part consists of security and hardness proofs.

- **Chapter 6** presents the protocol for Secure Function Evaluation introduced in the chapter 4. It also consists of correctness and hardness proofs for this protocol.

- **Chapter 7** is about the privacy of the protocol. This chapter consists of definitions necessary to prove protocol privacy and privacy proofs with respect to the parties involved in the protocol transactions.

- **Chapter 8** contains the application for the Secure Function Evaluation protocol introduced in chapter 6. It throws light on details of carrying out Similar Patient Query under privacy preserving settings. Then we introduce a new algorithm to compute Euclidean Distance among genomic data under privacy preserving settings. Finally a protocol for carrying out SPQ will be described.

- **Chapter 9** discusses the implementation of our cryptosystem along with the SPQ application. The first part introduces algorithms to find the best α, β for prime generation and prime generation itself based on the required Legendre symbol sequences. The second part of the chapter talks about the SPQ application. Implementation steps along with results and analysis for both the processes is provided in this chapter.

- **Chapter 10** is a conclusion chapter that presents discussion and scope for future work on this thesis along with the concluding remarks.

Chapter 2

Background

In this chapter, we go through some basic concepts that are necessary to understand the cryptosystem we developed. It also highlights the literature available in the area of secure multiparty computation and its related topics. We deal with concepts of Secure Multiparty Computation where we give a generic overview of the concept and go on to explain about Secure Function Evaluation and Private Function Evaluation and the terms that can be used interchangeably depending on the application. We then proceed to explain the existing SMC protocols which include Yao- based protocols namely Garbled Circuits and Universal Circuits, and the protocols that are built tapping the mathematical properties of public key cryptosystems namely Homomorphic encryption schemes. The scope of this thesis is limited to extending additively homomorphic encryption schemes for secure function evaluation and hence the next sections deal with the details of these schemes. To give a detailed understanding of how homomorphic encryption can be used to carry out, we proceed the rest of the chapter with an explanation on public key cryptosystems. Then we describe two probabilistic cryptosystems, Okamoto-Uchiyama and Paillier cryptosystems in detail which can be used for applying our scheme. We discuss the concept of homomorphism and give brief details of partial homomorphism that is displayed by the probabilistic cryptosystems. Partial homomorphism is exhibited mainly in case of addition, multiplication and scalar multiplication, so we explain how all of the three operations work on encrypted data. The last section of this chapter introduces the backbone of our research- Residues based homomorphic encryption scheme. Its basic idea will be discussed along with the related work done in this aspect.

2.1 Secure Multiparty Computation

The key idea behind secure multiparty computation, which we shall denote as MPC hereafter, is that two or more parties collaborate on computing some agreed upon function on their private inputs. None of the parties in this transaction are allowed to learn about others' input and by the end of the transaction only the output of the function is revealed. The security in a MPC is established by considering a Real-Ideal paradigm. The main aim of cryptography is to build protocols that secure in the presence of corrupt participants. So the ideal world setting is such that a secure computation of some function $F(x)$ is carried out by some trusted party T. The inputs to these function are provided by multi parties P_i. The assumption in this ideal world is that an adversary can manipulate anyone from P_i but not the trusted party T. This makes it easier for us to understand security in an ideal world because an adversary learns no more than $F(x)$ from T. The real world notion is much more complex as there are no trusted parties. All P_is communicate with each other using some protocol π. An MPC is secure if the π carrying out the MPC in real world can provide security that is equivalent to that of ideal world.

2.1.1 Secure Function Evaluation

This is a process of calculating a function on inputs of multiple parties without sharing the inputs of both the parties with each other. This started way back in the 1980s during when it was used only for theoretical purposes [11]. The first problem was a secure two-party computation that started with a millionaire's problem introduced by Andrew Yao [69]. It was generalized to more generic secure computations in [70] and practical applications were developed at a later point in time. There is tremendous amount of research conducted in this field to lower the time and computational complexity required for performing secure function evaluation cryptographically. The more formal definition for secure function evaluation is explained in the next paragraph. We first give the notion of secure function evaluation between two parties, then extend the definition to SFE among multiple parties.

Secure Function Evaluation between two parties can be carried out as follows:
Each party holds inputs $\{X, Y\}$ respectively. They want to evaluate a function $f(X, Y)$ on their inputs, without actually sharing their respective inputs with each other. Secure Function Eval-

uation will be carried out in such a way that upon its completion both the users know $F(X, Y)$ and nothing about the inputs for $X \neq Y$.

Based on this definition, we give a more formal definition for secure function evaluation that involves multiple number of users. Let n be the number of users, u be the number of inputs possessed by each individual user. These users want jointly evaluate some function $F : \{0,1\}^{un} \mapsto \{0,1\}^m$. Each *ith* user possesses an $x_i \in \{0,1\}^u$. Secure Function Evaluation will be carried out in such a way that upon its completion all the n users will receive $F(x_1, x_2, x_3, ..., x_n)$ and nothing about the x_j for $j \neq i$. Secure Function Evaluation can be carried out under three adversary settings: *Honest-but-curious*, *Malicious* and *dynamic*. More formal notions of these adversaries are discussed in chapter 6. The tools used for secure function evaluation are discussed under secure computation methods. Mostly SFE is used interchangeably with secure computation or private function evaluation. But there are some subtle differences between PFE and SFE, which will become clearer with the upcoming sections.

2.1.2 Private Function Evaluation

As discussed earlier, Private Function Evaluation is a special case of Multiparty computation. The difference between PFE and SFE is that in SFE, the function F need not be hidden and only the respective inputs are hidden from each other, whereas in PFE, while the security constraints of SFE are maintained, it also hides F. In [41] it was highlighted that, while the function to be computed will be a private input of one of the parties in secure multiparty computation process, the key security requirement is that the only information an adversary who does not control F can learn is about the size of the circuit and nothing else about F. Here, size of the circuit indicates the number of gates and distinct wires in the circuit. The circuits used in Private Function Evaluation are called as Universal Circuits, and more details about these are discussed in the next section. Formally, we can define Private Function Evaluation as:
Given n users, with u inputs, let an user p hold $F : \{0,1\}^{un} \mapsto \{0,1\}^m$ as one of their inputs. PFE is carried out in such a way that upon completion users will only learn the outputs and no information will be revealed about F and x_j for $j \neq i$. Any adversary trying to control p cannot learn anything about F other than its depth.

2.1.3 Secure Computation Protocols

The two most popular approaches for secure multiparty computations are using Homomorphic Encryption and Garbled circuits. Homomorphic Encryption often requires little interaction but is computationally complex. Garbled Circuits make efficient use of symmetric key operations but require multiple interactions between the two parties. Both the methods work by involving a trade-off between communication and computational complexity. Some applications require minimum communications between the parties involved, which is why they cannot rely on Garbled Circuits. And for those operations that can be implemented using unbounded number of additions and scalar multiplications, Additively Homomorphic Cryptosystems are more suitable than their Fully Homomorphic Counterparts as they are relatively fast and straightforward to implement. The upcoming subsections survey some of the secure computation protocols. Any secure computation protocol can be viewed as a form of computation carried out under encryption.

Garbled Circuits

These were initially proposed by Yao [69] for solving the Millionaire's problem. We convert the input function into a Boolean circuit, where each input into this circuit will be encrypted by two separate keys. These encrypted values are then added to a truth table by randomly permuting them- also known as "garbling". This truth table will be exchanged between the parties using oblivious transfer, where in the sender transfers many pieces of information to the receiver, but remains oblivious to what piece of information has been received. Then an evaluation will be applied to extract the output from the key information obtained in OT protocol. There are many different types of garbled circuits available for secure function evaluation, but Yao's garbled circuit is a classical approach and rest of the garbled circuits such as point and permute [6], free XOR [35], Garbled row reductions [43] [52], fleXOR [34], half gates [72], and garbled gadgets [5] are optimizations to Yao's circuit. Whatever the scheme may, the functionality remains constant with following steps:

- A garbling step, where a security parameter is provided, along with the function that needs to be evaluated and returns a garbled circuit as an output along with information

about encoding and decoding. This garbled circuit, encoding and decoding information will be used in the upcoming steps.

- Encoding information will be used to garble an input.
- Garbled circuit will take the garbled input and returns a garbled output. This step is called evaluation.
- The final step would be where decoding information will be used to decode the garbled output to plain text output.

Homomorphic Encryption

Apart from garbled circuits, Homomorphic encryption is another type of protocol which is used to carry out secure computations under private settings. Homomorphic encryption is a form of encryption that allows a user to perform computations on encrypted data. The result of homomorphic encryption will be another encrypted text which when decrypted will match results of operations as if they were carried out on plain text. Homomorphic encryption schemes allowed for a wide range of applications that require privacy preserving computations such as outsourcing cloud storage and predictive analytics on encrypted data to name a few. A lot of secure function evaluation schemes have been proposed by [26] [66] [51] [37] [17] [9] which deal with homomorphic encryption scheme. Broadly, there are two types of HE schemes namely partial HE and fully HE, which will be discussed briefly in the next sections.

Partial Homomorphic Encryption

The rest of our thesis will deal with the cryptosystem based on Partial Homomorphic Encryption scheme. So, this section will highlight some generic aspects with existing literature in PHE that deals with secure function evaluation. The technical aspects of PHE are discussed through various sections across this thesis. A PHE based scheme usually performs limited mathematical operations on encrypted data, which is why we use PHE for some linear Secure Function Evaluation protocols that mostly rely on either simple Addition or multiplication. It is difficult to carry out multiple operations using a simple PHE scheme, which is why they

have less computational complexity. [51] [37] and [17] tried to extend simple PHE schemes to support multiple operations such as addition, multiplication, exponentiation and natural log. Nonetheless, a variety of health care data analytics which do not have too many computations or rely only on linear operations like additions and scalar multiplications, have a tremendous amount of potential in resorting to partial homomorphic encryption schemes.

Fully Homomorphic Encryption

Since PHE has a potential to perform limited number of computations on encrypted data, researches were conducted to explore the possibilities of performing multiple computations on encrypted data. It began with [25] developing a HE scheme using lattice based cryptography. Later, several schemes were introduced to carry on complex operations like multiplication and other complex arithmetic circuit operations. Though there are many sub-types in case of HE schemes, we just broadly consider them as partial or full depending on the linearity of the computations the schemes perform. Nonetheless, Fully homomorphic encryption is the strongest notion of homomorphic encryption there is, as it is supposed to allow unbounded number of arithmetic circuit operations. This way, secure function evaluation can be carried out in an easy way with FHE. But the computational complexity is too high for FHE, making PHE a more suitable option for real time applications.

Universal Circuits

These are a special type of arithmetic circuits specifically dealing with Private Function Evaluation. Since PFE involves hiding the function to be evaluated, generic SFE techniques may become too simple for usage. Let there be two parties P_1 and P_2 such that P_1 holds input x and P_2 holds a circuit that represents the function f that needs to be evaluated. From the definition of PFE, we need the f to remain private. This implies that the circuit C_f holding f, itself should remain private. PFE stands in contrast to an SFE meaning that in SFE for inputs x, y, we take an f which is agreed upon previously by the participants. Now we calculate $f(x, y)$ using an agreed upon C_f without knowing the inputs. In case we need to hide this C_f, we try to develop some Universal Circuit U_n with n gates such that $U(C, x) = C_n(x)$. This notion of Universal Circuits have been used in the literature by [36] [57] [49]. The implementations are tedious,

error prone, and challenging when it comes to writing code for Universal Circuits. Certain optimizations have also been introduced in [33] [41], but the computational complexity is always dependent on the depth of the circuit making it very complex for real time applications.

2.2 Public Key Cryptography

The idea of public key cryptography is to share secrets over a public network or platform. The participants in a transaction done using public key cryptosystem have never met to exchange the secret key. It uses a pair of keys: Public and Private, where the public key is distributed to everyone trying to encrypt using the cryptosystem and private key is known solely to the owner. The hardness of the breaking up a public key cryptosystem is based on the mathematical problems that help us to produce one-way. Basically we take up some hardness assumptions from mathematics and use them to build public key cryptosystems that are hard to break using a polynomial time algorithm. The most widely used hardness assumptions are Discrete Logarithm problem [30], Quadratic Residuosity Problem introduced by [24] and applied by cryptosystems like Goldwasser-Micali [27], Decisional composite residuosity assumption [48], higher Residuosity Problem [73], and Integer Factorization problem [55]. Based on these hardness assumptions many public key cryptosystems were designed. To apply our scheme, we need the public key schemes that support Additive Homomorphism, which include Goldwasser-Micali [27], Paillier [48], Okamoto-Uchiyama [47], DGK [14], and Exponential ElGamal [19]. The scope of this thesis is limited to applying Okamoto-Uchiyama and Paillier Cryptosystem for secure function evaluation. A detailed description of these two cryptosystems is given in the coming subsections. Any cryptosystem usually contains three parts: a key generation algorithm, an encryption step, and a decryption step. Our description for all the cryptosystems in this thesis follow the same steps. Also, we use M as a message space of size $\mathbb{Z}_k$, where k denotes the number of bits of prime p. One of the greatest advantages in using these cryptosystems is that both of them are probabilistic. When we say a scheme is probabilistic, it indicates that each message encrypts into different cipher text. In fact, same message is encrypted into different cipher text each time as we encrypt due to the presence of a random factor. This makes these cryptosystems suitable for limited message spaces like $\{0,1\}^n$ or $\{0,1,2\}$, as we may need to

encrypt the same number multiple times. If we use a non-probabilistic cryptosystem like RSA to encrypt these message spaces, an attacker can easily figure out the plain text messages by using the cipher text that is published.

Before proceeding further we would like to give some basic definitions that are necessary to understand the cryptosystems we are going to discuss next.

Group A group is an algebraic structure that has two components- set of elements G and a binary operator $(*)$. These groups display the following properties:

- ***Closure Property***: $\forall a, b \in G, (a * b) \in G$
- ***Associative Property***: $\forall a, b, c \in G, (a * b) * c = a * (b * c)$
- ***Identity Property***: $\exists i \in G \mid \forall a \in G, (i * a) = a$, where i is identity element
- ***Existence of Inverse***: $\forall a \in G, \exists b \in G \mid a * b = i$, where i is the identity element
- Some groups called ***Abelian Groups*** have a property called ***Commutative Property*** which ensures that $\forall a, b \in G, a * b = b * a$

Cyclic Group A cyclic group is an abelian group that can be generated from a single element. These elements are called generators of the group and form the backbone of public key cryptography. For example, if G is a cyclic group with some generator g, this implies that every element in G is equal to g^k for some $k \in \mathbb{Z}$. For groups with finite order n we can have $k \in \{0, 1, 2, 3, ..., n-1\}$, in these cases we can have more than one generators. These generators form the base of the probabilistic cryptosystems which perform encryption using exponentiation of plain text message.

2.2.1 Okamoto-Uchiyama Cryptosystem

Developed by Tatsuaki Okamoto and Shigenori Uchiyama in 1998 [47], this cryptosystem depends on the hardness assumption called p-subgroup assumption which is discussed later in chapter 5. We tried to apply our secure function evaluation protocol for the case of Okamoto-Uchiyama cryptosystem. It works for integers in the multiplicative subgroup of integers

(mod n). The decryption of this cryptosystem relies on the group homomorphism of the type: $L : \Gamma \mapsto \mathbb{Z}_p$ which is given by $L(xp + 1) = x$, which can be re written as $L(x) = \dfrac{x-1}{p}$.

Encryption

The encryption and decryption functions are implemented same way as described in [47]. For a plain text message $m \in M$, public key $PK = \{n, g, h\}$ where:

- n = p^2q, (p, q are two large primes)
- Select a $g \in \mathbb{Z}_n^*$ and $g^{p-1} \not\equiv 1 \pmod{p^2}$.
- This implies the encryption base $B = \{g \in \mathbb{Z}_n^* \mid ord_{p^2}(g \pmod{p^2})^{p-1} = p\}$.
- [47] proves that if $g \xleftarrow{R} \mathbb{Z}_n^*$, then $g \in B$ with overwhelming probability.
- $h \equiv g^n \pmod{n}$.

We generate a ciphertext C by applying the Encryption function $Enc(M, PK) = C$

$$c \equiv g^m h^r \pmod{n}.$$

Decryption

To decrypt a ciphertext C we use the private keys $SK = \{p, q\}$. We compute:

$$a = L(c^{p-1} \pmod{p^2}) = \frac{(c^{p-1} \pmod{p^2}) - 1}{p}.$$

$$b = L(g^{p-1} \pmod{p^2}) = \frac{(g^{p-1} \pmod{p^2}) - 1}{p}.$$

Using Extended Euclidean algorithm we compute:

$$b' = b^{-1} \pmod{p}.$$

Finally giving:

$$m = ab' \pmod{p}.$$

2.2.2 Paillier Cryptosystem

This cryptosystem considers $n = pq$ where p, q are large primes $\in \mathbb{Z}_k$. It was developed by Pascal Paillier in 1999 and relied on the hardness assumption that computing nth residue classes is computationally difficult. As described in [48], this probabilistic public key scheme displays additive homomorphism.

Encryption

Public key = n, g where:

- $n = pq$.
- select some $g \in \mathbb{Z}^*_{n^2}$

This gives: $c \equiv g^m r^n \pmod{n^2}$.

Decryption

The private (decryption) keys are λ and μ where

$$\lambda = LCM(p-1, q-1).$$

$$\mu = (L(g^\lambda \pmod{n^2}))^{-1} \pmod{n}.$$

$$m = L(c^\lambda \pmod{n^2}).\mu \pmod{n}.$$

where L is a function of the form $L(x) = \dfrac{x-1}{n}$ In other words, consider two groups G and H:

$$G = \{x = g^m h^r \pmod{n} \text{ for } m \in \mathbb{Z}_p \text{ and } h \in \mathbb{Z}_n\}.$$

$$H = \{x = h^n \pmod{n} \text{ for } y \in G\}.$$

The p-subgroup problem is to distinguish elements of H from elements of $G \backslash H$.

2.3 Partial Homomorphic Encryption

Of all the secure computation methods discussed in section 2.1, we decided to develop a cryptosystem based on Partial Homomorphic Encryption properties keeping in mind the time and computational complexities. As discussed earlier, PHE encompasses evaluation of a single type of computation: either addition or multiplication. Additionally, we can also perform scalar multiplication along with addition in some cryptosystems that use a generator g and raise the plain text m to g to form a cipher-text. The method to carry out addition and multiplication in these type of cryptosystems is discussed in the following sections.

2.3.1 Addition

Additive homomorphism is displayed by the cryptosystems that generate cipher text of the form $c = g^m$. This is the case with cryptosystems discussed in the previous sections namely, Paillier, Okamoto-Uchiyama, DGK, and exponential El-Gamal. Let $n,\ g$ be public keys and $m_1,\ m_2$ be plain text messages[1], then additive homomorphism can be expressed as follows:

$$\begin{aligned}\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) &= g^{m_1} g^{m_2} \pmod{n} \\ &= g^{m_1+m_2} \pmod{n} \\ &= \mathsf{Enc}m_1 + m_2).\end{aligned}$$

Goldwasser-Micali

Given $n,\ a,\ m,\ r$, where $\left(\frac{a}{n}\right)$ is a quadratic non-residue, $r \xleftarrow{R} \mathbb{Z}_n$ and $m \leftarrow \{0, 1\}$, $\mathsf{Enc}(b) = a^m r^2 \pmod{n}$. The addition in this cryptosystem results in a *XOR* function, i.e., an addition (mod 2). Mathematically:

$$\begin{aligned}\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) &= a^{m_1} \cdot a^{m_2} \cdot r_1^2 \cdot r_2^2 \pmod{n} \\ &= a^{(m_1+m_2)} \cdot (r_1 \cdot r_2)^2 \pmod{n} \\ &= \mathsf{Enc}(m_1 \oplus m_2).\end{aligned}$$

[1]for the ease of explanation, we are using the common expressions from the cryptosystems by avoiding the hider h and randomizer r

2.3.2 Scalar Multiplication

For the same settings as discussed above, let k be some constant, then homomorphism for scalar multiplication can be expressed as follows:

$$\begin{aligned} \mathsf{Enc}((m)^k) &= (g^m)^k \pmod{n} \\ &= (g)^{mk} \pmod{n} \\ &= \mathsf{Enc}(km). \end{aligned}$$

2.3.3 Multiplication

Multiplicative homomorphism is displayed by the cryptosystems which produce $c = g^m$. In this case, multiplication between encrypted m_1 should be carried out like scalar multiplication by raising the cipher-text to plain text m_2. A different set of cryptosystems where we do not use exponentiation to obtain cipher text, specifically whose security dependent on the hardness of Discrete Logarithm Problem [30] can perform multiplication on two cipher texts. These are unpadded RSA and El-gamal. Multiplication for exponentiation based cryptosystem can be expressed as follows:

$$\begin{aligned} \mathsf{Enc}(m_1)^{(m_2)} &= (g^{m_1})^{m_2} \pmod{n} \\ &= g^{m_1 m_2} \pmod{n} \\ &= \mathsf{Enc}(m_1 \cdot m_2). \end{aligned}$$

Unpadded RSA

Recall the basic RSA cryptosystem where, n is the public key modulus, e is the exponent and $Enc(m) = m^e \pmod{n}$. The homomorphism in this cryptosystem can be expressed for messages m_1, m_2 as:

$$\begin{aligned} \mathsf{Enc}(m_1).\mathsf{Enc}(m_2) &= (m_1)^e \cdot (m_2)^e \pmod{n} \\ &= (m_1 \cdot m_2)^e \pmod{n} \\ &= \mathsf{Enc}(m_1 \cdot m_2). \end{aligned}$$

ElGamal

This cryptosystem, proposed in [19], uses a generator g, $h = g^x$ such that x is the secret key, for some random number $r \in ord_g$ the cipher text is formed as $Enc(m) = (g^r, m.h^r)$. For this, the multiplicative homomorphism is calculated as:

$$\begin{aligned}\mathsf{Enc}(m_1).\mathsf{Enc}(m_2) &= (g^{r_1}, m_1 h^{r_1})(g^{r_2}, m_2.h^{r_2}) \\ &= (g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2}) \\ &= \mathsf{Enc}(m_1 \cdot m_2).\end{aligned}$$

2.4 Residues Based Homomorphic Evaluation

The homomorphism presented in this thesis is based on an Evaluation Function which is discussed later in section 4.3, where we evaluate the functions of the form $\mathbb{Z} \mapsto \{0,1\}^\ell$. The basic idea for this cryptosystem is derived from [20]. Identifying patterns in the runs of consecutive Quadratic Residues and Non residues $\pmod p$ has always intrigued number theorists. If we consider Legendre Symbols of a sequence of numbers of the form $\{x, x+1, x+2, ..., x+n\}$ $\pmod p$, the pattern looks very random. In [20] an interesting pattern was identified with respect to distribution of quadratic residues. They seem to imitate threshold functions which are of the form:

$$T(x) = \begin{cases} 1 & \text{if x} \geqslant \text{threshold} \\ 0 & \text{Otherwise.} \end{cases}$$

Let $QR_p(x)$ be a function representing Quadratic Residuosity of an integer $x \in \mathbb{Z}$. It can be defined as:

$$QR_p(x) = \begin{cases} 1 & \text{if x is a quadratic residue mod p} \\ 0 & \text{Otherwise.} \end{cases}$$

Combining both, the threshold function can be privately evaluated during decryption as follows:

$$QR_p(x+f) = T(x).$$

The limitation with the above idea is that, a search based approach should be used in identifying the required patterns for Legendre Symbols $\pmod p$. In other words, the domain for the

function to be evaluated will be taken, say the domain is $f(x) = \{1, 1, 1, 0, 0, 0\}$. This *f(x)* will be used to search for same length of Residues and Non-residues iteratively across a range of prime numbers, because remember a *Residue* $\pmod p = 1$ and *Non-Residue* $\pmod p = 0$. This search based approach is difficult as the search needs to be carried out iteratively across each prime $\in \mathbb{P}$ until the required pattern is found. This would become very difficult to find larger runs of consecutive Residues and Non-Residues in the polynomial time as the domain size of the function increases.

Many researches [68] [50] [31] [16] [8] [7] have been conducted in identifying patterns in consecutive runs of Quadratic Residues. Although these studies were able to establish certain bounds on the length of the sequences found, they only deal with consecutive residues or non residues, but not arbitrary patterns where we can expect a combination of both residues and non-residues. In fact, it is this randomness in occurrence of *R* and *N* is what makes it useful for a wide range of applications in cryptography.

Our idea is that, instead of looking for arbitrary patterns in consecutive set of *Rs* and *Ns*, we can go for numbers $\pmod p$ by taking a random step size or go for arithmetic sequences of fixed form. This may help us to find out the prime that has required patterns in *Rs* and *Ns*. As opposed to applying a search across infinitely many numbers, it is easier to pick a sequence and use the sequence to generate the necessary primes. This helped us in building a new cryptosystem altogether, which can now be extended to privately evaluate functions with a domain size up to $512 - bits$. The functions that can be evaluated privately using this special type of cryptosystems are of the following form:

$$f : \mathbb{Z}^+ \mapsto \{0, 1\}^\ell$$

Every function has an input set and target set. We define this input set as Domain. The function performs certain operations on the input set, which produce the output that belongs to the target set. The target set is known as co-domain. In the current scenario, we are looking at functions whose domain X falls in a set of sequence of integers that can be expressed as $(\alpha i + \beta) \mid \alpha, \beta \in \mathbb{Z}, i \in \{0, 1, 2, ..., n\}$. The co-domain $f(x)$ belongs to the message space of $\{0, 1\}^n$. The Boolean Function is chosen as co-domain because, the Legendre Symbols symbols as defined above merely represent Boolean values indicating whether the given number is a

quadratic residue $\pmod{p}$ or not. Finding the apt α, β forms the heart of our scheme because, these values determine the start of the required sequence of Quadratic Residues.

Chapter 3

Mathematical Preliminaries

This chapter deals with some definitions, mathematical basics required to discuss our cryptosystem used for Secure and Private Function Evaluation. The definitions in the first section will help us understand the type of functions that we are going to deal with for function evaluation under privacy preserving settings. These definitions will be used to prove two theorems that are the basic foundation for the key generation described in section 4.1. The theorems stated are presented along with their proofs built using basics from number theory. We use the same notations for the functions and other elements used in PFE throughout the thesis. For example β denotes the beginning of the sequence used for prime generation and α is the step size of the sequence.

3.1 Basic Definitions

Before proceeding with basic definitions, please note that the function g discussed in the following sections is different from the generator $g \in \mathbb{Z}_n$ that used for encryption. The generator g is an integer which will be used as a base for encryption of plain-text messages. Where as function $g : \mathbb{Z} \mapsto \{0, 1\}$ denotes a linear embedding function upon which we carry out our secure function evaluation.

Quadratic Residues A number $n \in \mathbb{Z}_1$ is called a Quadratic Residue mod p, where p is an odd

prime $\iff$ there exists some x such that:

$$n \equiv x^2 \pmod{p}.$$

Otherwise n is a non-residue.

Properties of Quadratic Residues Quadratic Residues (QR) and Non-Residues (NR) have the following properties:

$$QR \cdot NR = NR$$
$$QR \cdot NR = QR$$
$$NR \cdot NR = QR.$$

Legendre Symbol The Legendre symbol is a function of a and p and is defined as:

$$\left(\frac{a}{p}\right) \equiv \begin{cases} 1 & \text{if a is quadratic residue} \pmod{p} \\ -1 & \text{if a is quadratic non residue} \pmod{p} \\ 0 & \text{if } a \equiv 0 \pmod{p}. \end{cases}$$

Relatively Prime Integers Two integers are said to be relatively prime to each other or Co-prime integers if they do not have a common factor other than 1. Mathematically, let $a, b \in \mathbb{Z} \mid a \neq 0, b \neq 0$, then a, b are co-prime $\iff gcd(a, b) = 1$. As a corollary, every distinct pair of prime numbers are relatively prime to one another.

Chinese Remainder Theorem Let p_1, p_2 be pairwise co-prime. Then the system of equations:

$$a \equiv b_1 \pmod{p_1}$$
$$a \equiv b_2 \pmod{p_2}$$

has a solution in a.

Quadratic Reciprocity Law The Quadratic Reciprocity Law states that:

$$\left(\frac{a}{p}\right) \equiv \begin{cases} \left(\frac{p}{a}\right) & \text{if } a \text{ or } p \equiv 1 \pmod{4} \\ -\left(\frac{p}{a}\right) & \text{if } a, p \equiv 3 \pmod{4}. \end{cases} \tag{3.1}$$

Euler's Criterion For an odd prime p and an integer $a \in \{1, p-1\}$:

$$a^{\frac{p-1}{2}} \equiv \begin{cases} 1 & \text{if a is quadratic residue} \pmod{p} \\ -1 & \text{if a is quadratic non residue} \pmod{p}. \end{cases}$$

Function A function is rule established between two sets such that it assigns each element of the first set to exactly one element in the second set. These two sets are also otherwise known as input and target sets respectively. Consider two sets X, Y such that:

$$f : X \to Y$$

From the above, we can define the input set **X** as domain and the target set **Y** as co-domain.

Functional Embedding Let $f : \mathbb{Z}_p^* \to \{0, 1\}$ and $g : \mathbb{Z}_k \to \{0, 1\}$ with $k \leq p$. We say function g is embedded in a function f if there exists a domain mapping function $h : \mathbb{Z}_k \to \mathbb{Z}_p^*$ such that for all $i \in \mathbb{Z}_k$,

$$f(h(i)) = g(i).$$

Linear Embedding We say function g is linearly embedded in the function f if the domain mapping function h is linear i.e., there exists an α, β such that,

$$f(\alpha i + \beta) = g(i).$$

Dirichlet's theorem Also known as Dirichlet's prime number theorem, this theorem states that for any two positive integers a, d that are co-prime, there exist infinitely many primes of the form $ka + d$, where k is a positive integer. Simply put, there are infinitely many primes that are congruent to $a \pmod{d}$.

3.2 Linear Embeddings in Residue Symbol Sequences

The central idea of our method is to find linear embeddings of Boolean functions using the sequence of Legendre symbols modulo a prime p. Let $g : \mathbb{Z}_k \to \{0, 1\}$ be a Boolean function. Let $f : \mathbb{Z}_p^* \to \{0, 1\}$ be the Legendre symbol of an integer $0 < x < p$ mapped to the Boolean domain, i.e.,

$$f(x) = \frac{\left(\frac{i}{p}\right) + 1}{2}.$$

Finally, let the linear domain mapping function $h : \mathbb{Z}_k \to \mathbb{Z}_p$ be defined as

$$h(i) = \alpha i + \beta.$$

for some prime p and $0 < \alpha, \beta < p$ such that

$$f(h(i)) = g(i).$$

Using this approach we can implement Boolean function g as a linear embedding within the sequence of Legendre symbols modulo prime p. With the following two theorems we prove that such a p, α and β exist for all Boolean functions g.

3.2.1 A new technique for Prime Number Generation

Theorem 3.2.1 *Consider a list of k distinct primes* $\{a_1, \ldots, a_k\}$ *and a list of residue symbols* $\{\ell_1, \ldots, \ell_k\}$ *where* $\ell_i \in \{-1, 1\}$. *For all* $1 \leq i \leq k$, *there exists a prime p such that*

$$\left(\frac{p}{a_i}\right) = \ell_i.$$

Proof The proof proceeds in two parts. In the first part, we prove the existence of some integer p' that satisfies these conditions. In the second part, we prove the existence of a p' implies the existence of a prime p with the same properties. For each ℓ_i and a_i, pick some $0 < b_i < a_i$ such that

$$\left(\frac{b_i}{a_i}\right) = \ell_i.$$

A solution for all $\ell_i \in \{-1, 1\}$ is guaranteed to exist given there exist both $(a_i - 1)/2$ quadratic residues and non-residues modulo a_i. Let p' be defined by the following system of equations:

$$\begin{aligned} p' &\equiv b_1 \pmod{a_1} \\ &\vdots \\ &\equiv b_k \pmod{a_k} \end{aligned}$$

Because each a_i is prime, each $0 < b_i < a_i$ is to be co-prime to a_i. Therefore a solution for p' exists via the Chinese remainder theorem. Since

$$\left(\frac{b_i}{a_i}\right) = \ell_i,$$

and $p' \equiv b_i \pmod{a_i}$, then for each $1 \leq i \leq k$ we have

$$\left(\frac{p'}{a_i}\right) = \ell_i.$$

Now we show the existence of an integer p' implies the existence of a prime p with the same congruences. Let $A = \prod a_i$ and $p = kA + p'$ for some integer $k \geq 0$. Since $p \equiv p' \pmod{A}$, and therefore $p \equiv b_i \pmod{a_i}$, then

$$\left(\frac{p}{a_i}\right) = \ell_i.$$

Finally, since p' is relatively prime to A, Dirichlet's theorem guarantees there are infinitely many primes of the form $kA + p'$. ∎

Theorem 3.2.2 *For all $k > 0$ and all functions $g : \mathbb{Z}_k \to \{0, 1\}$ there exists a prime p and two integers $0 < \alpha, \beta < p$ such that for all $0 \leq i < k$,*

$$\frac{\left(\frac{\alpha i + \beta}{p}\right) + 1}{2} = g(i).$$

Proof Let α, β, k be positive integers such that $\alpha i + \beta$ is prime for all $0 \leq i < k$. The existence of such an α, β is guaranteed for all $k > 0$ by the theorem due to Green and Tao [28] which proves the primes contain arbitrarily long arithmetic sequences, and, therefore, there exists an α, β for all $k > 0$ such that $\alpha i + \beta$ is prime for all $0 \leq i < k$. Given such a linear sequence of prime valued [1] $(\alpha i + \beta)$'s, theorem 3.2.1 guarantees there exists a prime p such that for all $0 \leq i < k$,

$$\left(\frac{p}{\alpha i + \beta}\right) = 2g(i) - 1.$$

Suppose there existed a p such that $p \equiv 1 \pmod{4}$. Then by the law of quadratic reciprocity,

$$\left(\frac{\alpha i + \beta}{p}\right) = \left(\frac{p}{\alpha i + \beta}\right) = 2g(i) - 1,$$

and therefore,

$$g(i) = \frac{\left(\frac{\alpha i + \beta}{p}\right) + 1}{2}.$$

[1] Requiring all $(\alpha i + \beta)$ be prime is only done to facilitate the existence proof. In practice, Algorithm algorithm 2 can generates suitable primes p in the presence of composite $(\alpha i + \beta)$.

In the alternate case where all such primes p were congruent to 3 (mod 4), Theorem 1 also guarantees there exists a prime p such that

$$\left(\frac{p}{\alpha i+\beta}\right)=\begin{cases}2g(i)-1 & \text{if } \alpha i+b\equiv 1 \pmod 4\\ 1-2g(i) & \text{if } \alpha i+b\equiv 3 \pmod 4.\end{cases}$$

For all $\alpha i+\beta\equiv 1 \pmod 4$, quadratic reciprocity again gives us

$$\left(\frac{\alpha i+\beta}{p}\right)=\left(\frac{p}{\alpha i+\beta}\right)=2g(i)-1.$$

Finally, for all $\alpha i+\beta\equiv 3 \pmod 4$,

$$\left(\frac{\alpha i+\beta}{p}\right)=-\left(\frac{p}{\alpha i+\beta}\right)=-(1-2g(i))=2g(i)-1.$$

Therefore

$$g(i)=\frac{\left(\frac{\alpha i+\beta}{p}\right)+1}{2} \tag{3.2}$$

for all $0\le i\le k$. ∎

Chapter 4

Our Cryptosystem

This chapter presents a public-key method for homomorphically evaluating functions of the form $f : \mathbb{Z} \rightarrow g(i)$, where $g(i)$ is a linear embedding found in f. Based on the fundamentals of public key cryptosystem,let us define this cryptosystem as $\mathsf{CS} = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$. The three components indicate:

- Gen - Key generation algorithm
- Enc - Encryption function
- Dec - Decryption function

Apart from these three components, we add a fourth functionality to our cryptosystem and label it as an evaluation function. This evaluation function enables us to successfully perform secure function evaluation in a unique way upon decryption of the ciphertext. In some scenarios, we do not need to read the components of the second party to perform the secure function evaluation due to the presence of this unique evaluation function described in section 4.3.

4.1 Key Generation

This section describes the key generation algorithm which is developed using the mathematical preliminaries presented in the chapter 3. We are trying to generate p, which is one of the large prime factors of the composite modulus $n \mid n = pq$. p is also a private key used for decryption

s_i	Factors
31	31
1969	$11 \cdot 179$
3907	3907
5845	$5 \cdot 7 \cdot 167$
7783	$43 \cdot 181$
9721	9721
11659	$89 \cdot 131$
13597	13597
15535	$5 \cdot 13 \cdot 239$
17473	$101 \cdot 173$
19411	$7 \cdot 47 \cdot 59$

Table 4.1: First 10 elements in a sequence length of 512 with at least one unique factor for $\alpha = 1938$ and $\beta = 31$

in various public key cryptosystems, so once generated, this p remains a secret. Depending on the length of $g(i)$ to be evaluated, we pick the α, β values such that $f(\alpha i + \beta) = g(i)$.

- Two parties that want to securely evaluate some function of the form $g : \mathbb{Z} \mapsto \{0,1\}^{\ell}$ agree upon a fixed co-domain $f(\alpha m + \beta)$, such that for some plain text message $m \xleftarrow{R} \mathbb{Z}_k$, $f(\alpha m + \beta) \mapsto g(m)$.

- Based on the length of the linear embedding $g(m)$ which is required to evaluate $f(m)$, we generate an arithmetic sequence of numbers S such that, each $s_i \in S$ has at-least one unique factor, i.e., $s_{i-1} = p_1 \cdot p_2$ and $s_i = p_1 \cdot p_3$ and so on. As we proceed through the set of arithmetic sequences, each new element must have at-least one new factor. So to evaluate a $g(m)$ of length 10, we need a sequence of 10 elements with atleast one unique factor. An example for the first 10 elements of such sequence along with their factorizations is displayed in table 4.1.

- Now we find b_i using theorem 3.2.1 to match our required sequence. In order for this to

work, not only does a_i need to have the matching symbols but the unutilized primes in the sequence need to remain as *QR*s so they don't affect the overall Legendre Symbol of s_i. This is established from the properties described in section 3.1.

- Once we find b_i, we recursively apply Chinese Remainder Theorem to until we find a prime p which is guaranteed as per theorem 3.2.2.
- To get $\left(\frac{a_i}{p}\right) \equiv \ell_i \pmod{p}$, depending on section 3.1 we find primes of the form $p \equiv 1 \pmod{4}$. Detailed steps of key-generation algorithms are discussed in section 9.1.1

4.2 Encryption and Decryption

Encryption and decryption mostly follow the same procedure as outlined in Chapter 2.2.1.

We can use both Okamoto-Uchiyama and Paillier cryptosystems in case of encryption scheme. Remember both of them have n, which is a product of two large primes as public key modulus where, $n = pq$ in case of Paillier and $n = p^2q$ in case of Okamoto-Uchiyama. Once we generate p using the key generation algorithm Gen, we can apply it in these cryptosystems and generate the required n. It does not affect the encryptions scheme on the whole.

Depending on the cryptosystem used to carryout the encryption function, decryption will be carried out in the final step of the transaction. Both Paillier and Okamoto-Uchiyama cryptosystem use p as private key for their decryption. As stated earlier, since p is common, this makes our key generation scheme Gen suitable for both the cryptosystems without having to modify too much while decrypting.

4.3 Evaluation Function

This function is used to homomorphically evaluate any Boolean function within the restricted domain which will be pre-specified. It is defined as follows:
$\mathsf{Eval}(PK, c)$: Given ciphertext $c = \mathsf{Enc}(m)$ and random cipher text blinding factor $r_c \in \mathbb{Z}_k$, α, β from the prime generation algorithm, we compute

$$c' = \mathsf{Eval}(c) = ((c^{\alpha}) \cdot \beta)^{r_c^2}.$$

Since our scheme supports only scalar multiplication and homomorphic addition, we use plaintext α and $\mathsf{Enc}(\beta)$ for computing the evaluation function.

4.3.1 Proof of Correctness

Theorem 4.3.1 *For $m \in \mathbb{Z}_k$, $\mathsf{Dec}(\mathsf{Eval}(\mathsf{c}))$ homomorphically evaluates m to $g(m)$, where $g(m)$ is the linear embedding of a function that we are trying to evaluate such that $g : \mathbb{Z}_k \mapsto 0, 1^\ell$*

Please note that the linear embedding g is different from the generator g used for encryption of a plain text message.

Proof For $\mathsf{Enc}(\beta) = \beta'$, expanding $\mathsf{Eval}(c)$ we have:

$$\begin{aligned}\mathsf{Eval}(\mathsf{Enc}(m)) &= ((g^m h^r)^\alpha) \cdot \beta')^{r_c^2} \\ &= ((g^m)^\alpha) \cdot \beta')^{r_c^2} \cdot (h^{r'}) \\ &= \mathsf{Enc}((\alpha m + \beta') \cdot r_c^2).\end{aligned}$$

Upon decryption and applying definition from section 3.2 we get:

$$\mathsf{Dec}(c^{'}) = (\alpha m + \beta) \cdot r_c^2).$$

Recall from definitions 1,2 and 3 that

$$\begin{aligned}f(r_c^2) &= \frac{\left(\frac{r_c^2}{p}\right) + 1}{2} \\ &= 1.\end{aligned}$$

Therefore applying the f-function to the decryption result we have

$$\begin{aligned}f((\alpha m + \beta) \cdot r_c^2) &= g(m) \cdot f(r_c^2) \\ &= g(m). \quad \blacksquare\end{aligned}$$

Chapter 5

Security of our Cryptosystem

The security of any cryptosystem is often established by showing that breaking them is harder than the mathematical problems which are generally considered difficult.This difficulty is generally proven using two notions: one-way encryption and semantic security. We establish these notions by starting with some relevant definitions. If we assume the functions chapter 3 defined in as problems, the following definitions indicate what problems are hard. For implementation we used Okamoto-Uchiyama cryptosystem, so the security proof of $\mathsf{CS} = \mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC}$ is based on the semantic security of Okamoto-Uchiyama Cryptosystem. We define the necessary terms for security proof using them we establish that Okamoto-Uchiyama cryptosystem is semantically secure and then prove that CS is also semantically secure by extension. The proof of Okamoto-Uchiyama cryptosystem security is based on the proof described in [47] and [32].

5.1 Basic Definitions

Negligible Function A function is negligible with respect to some $k \in \mathbb{N}$ if for all $c \in \mathbb{N}$ there exists some $M \in \mathbb{Z}$ such that $f(k) < \frac{1}{k^c}$ whenever $k > M$

Intractable Function A function $f : X \mapsto Y$ is intractable with respect to $x \in X_i$ if for all algorithms A, when the inputs in $\{X_j | j \neq i\}$ are held fixed, the probability function

$$Px[A(x) \in P(x)].$$

is non-negligible with respect to $x \in X_i$

One-way encryption It is the computational problem of computing a plain text message m from public key parameters- n, g and the cipher text $c = Enc(m)$. If inverting the encryption is intractable, the encryption is called a one-way function.

Semantic Security For b = 0, 1, let E_0 and E_1 be two experiments; let m_0, m_1 be two plain text messages and A be an adversary(algorithm) that is trying to break into the system. We have a challenger who outputs m_0 in E_0 and m_1 in E_1. We define the advantage of adversary outputting the same values for both the experiments as:

$$Adv[A, E] = |Pr[w_0] - Pr[w_1]| \in [0, 1]].$$

where w_b defines that an experiment b returned 1.
Now, E is semantically secure if for all efficient adversaries, the Adv[A,E] is negligible, meaning no efficient adversary can distinguish the encryption of m_0 from encryption of m_1.

Factoring Problem It is the problem of computing factors p and q of the composite modulus n.

Sylow's Theorems Let G be a finite group. Let p be a prime dividing $|G|$ such that $G = p^k m$ with $k > 1$ and $p \nmid m$ [23]

- **First Theorem** There is a subgroup $H \subseteq G$ of order p^k, it is called Sylow p-Subgroup.
- **Second Theorem** Any two Sylow p-Subgroups are conjugate, there is an element $g \in G$ such that $g^{-1}Hg = K$.
- **Third Sylow Theorem** Let n_p be the number of Sylow p-Subgroups then:
  - $n_p \mid m$
  - $n_p \equiv 1 \pmod{p}$
  - $n_p = |G|/|N_G(H)|$, where H is a Sylow p-Subgroup and $N_G(H)$ denotes the normalizer of H, i.e., the largest subgroup of G in which H is normal.

5.2 One-way Function

Theorem 5.2.1 *If the factoring problem is intractable, then the encryption is a one-way function.*

Proof The encryption function of this cryptosystem is same as that of Okamoto-Uchiyama cryptosystem. The proof works out by verifying that the distribution of the ciphertext in the cryptosystem can be simulated by an algorithm.We take two algorithms A_1 and A_2. A_1 inverts the encryption with a non-negligible probability and A_2 is a factoring algorithm. From section 4.2 we know that $C = g^m \pmod n$. We omit h^r for the time being from encryption function for the sake of convenience. Now we construct the algorithm A_2 as follows:

Choose $g' \xleftarrow{R} \mathbb{Z}_n$
Choose $z \xleftarrow{R} \mathbb{Z}_n$
Compute $C' = g^z \pmod n$
Compute $m = A_1(n, g', C')$
Compute $d = gcd(z - m, n)$
if $\sqrt{d} \in \mathbb{Z}$ **then**
    $d = \sqrt{d}$
end if
if $d < 2^k$ **then return** $(d, \frac{n}{d})$
end if

It is easy to establish that the distributions of g', C' are similar to g, c detailed proofs can be found from [47].This implies A_1 returns the m corresponding to C' with a non-negligible advantage.

Further, $g^z \equiv g^m \pmod n$ so $g^z \equiv g^m \pmod{p^2}$. As we know $ord_{p^2} g = p$, it gives us $m \equiv z \pmod p$. Now $z \equiv m \pmod p$ implies $z = m$ which occurs with a negligible probability. This implies $n \nmid z - m$ giving us a $gcd(z - m,\ n) = \gamma$, where $p \mid \gamma$. But we already know$n \nmid \gamma$, implying the d from algorithm A_2 may be one among$\{p,\ p^2,\ q\}$ breaking the factoring assumption altogether. Therefore, if A_2 is tractable, A_1 is easy to achieve making its converse is also true. ∎

5.3 Semantic Security

Theorem 5.3.1 *The encryption scheme CS presented in chapter 4 is semantically secure under chosen plain text attack.*

The proof for this theorem works in two parts.First we prove that the Okamoto-Uchiyama cryptosystem [47] is semantically secure chosen plain text attack. Second we extend the proof to our cryptosystem.

Given only n and g where $n = p^2q$ such that $p \nmid q-1$, $g \in \mathbb{Z}_n$ and order of $g^{p-1} \pmod{p^2}$ is p, we know that the order of $\mathbb{Z}_n^* = \mathbb{Z}_{p^2q}^* = (p-1)\cdot(q-1)\cdot(p)$. Recall that by Sylow's First theorem [23] $\mathbb{Z}_n^*$ has exactly one p-subgroup, let us indicate this p-subgroup as Ω, where $\Omega = \{ypq+1 | y \in \mathbb{Z}_p\}$. Consider some element $x \neq 1 \in \mathbb{Z}_n^*$. We can know whether $x \in \Omega$ by verifying if order of $x \in \mathbb{Z}_n^*$ divides p or $a^p \equiv 1 \pmod{n}$. Since p is prime, this is possible only if $|x| = 1$, but this contradicts our assumption, so $|x| = p$.

Definition Let $X \xleftarrow{R} \{\Omega, \mathbb{Z}_n^*\}$. The p-subgroup problem is a decisional problem of given n and $x \xleftarrow{R} X$, deciding whether $X = \Omega$ or $X = \mathbb{Z}_n^* \setminus \Omega$.

Lemma 5.3.2 *The Okamoto-Uchiyama cryptosystem is semantically secure $\iff$ p-subgroup problem is intractable.*

Proof p-subgroup problem is equivalent to distinguishing between the valid encryptions of 0 and 1.The proof for this equivalence with respect to Okamoto-Uchiyama cryptosystem is illustrated in [29]. To prove theorem 5.3.2, we need to understand the relationship between Ω and B. If $x \in \Omega$, then x^{x-1} cannot have order p in $\mathbb{Z}_n^* \implies x^{x-1} \notin B$. Similarly, if $x \in B$ and $x \in \Gamma \setminus 1$, then $x^{x-1} \in B$ because if $x^{x-1} \notin B$ then $x^{(x-1)(p-1)} \equiv 1 \pmod{n} \implies p | x-1$ which contradicts our assumption.

Based on these relations, consider a semantic security algorithm $A = (A_1, A_2)$ with non-negligible advantage. Here A_1 is used to get m_0, m_1 and A_2 returns whether $x \xleftarrow{R} X$ belongs to Ω or $\mathbb{Z}_n^* \setminus \Omega$. Here we compute $g = x^{x-1} \pmod{n}$. If $x \in \Omega$ then $g \notin B$, giving A_2 negligible advantage over guessing. This results in the probability of B giving $x = \Omega$ is negligible. Similarly if $x \in \mathbb{Z}_n^* \setminus \Omega$, then $x \in B$ with overwhelming probability, giving A_2, in turn B an

overwhelming advantage over guessing.By combining these, the overall probability of A_2 being correct is non-negligible. ∎

Proof of theorem 5.3.1 We begin with assumption that Okamoto-Uchiyama cryptosystem is semantically secure. Suppose there existed an algorithm A' that accepts some element $x \overset{R}{\leftarrow} X$, where $X \overset{R}{\leftarrow} \{\Omega, \mathbb{Z}_n^*\}$ and public key parameters of Okamoto-Uchiyama cryptosystem (n, g, h). For breaking CS, A' needs to guess whether $x \in \Omega$ or $x \in \mathbb{Z}_n^* \backslash \Omega$.

By theorem 5.3.2 we proved that Okamoto-Uchiyama system is semantically secure if there is an algorithm that reveals whether $x \in \Omega$ or $x \in \mathbb{Z}_n^* \backslash \Omega$.Since we assumed Okamoto-Uchiyama is semantically secure, it implies that no such algorithm exists, which in turn proves that A' also does not exist. ∎

Chapter 6

Secure Function Evaluation using our Cryptosystem

This chapter introduces the protocol developed for secure function evaluation based on our cryptosystem $\mathsf{CS} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. It is structured by describing the required input for the protocol and expected outputs once the protocol is finished. Let Alice and Bob be the participants involved in this transaction. The communication carried out between Alice and Bob for successful completion of protocol are indicated in the form of steps. One of the key requirements to keep any Multiparty Computation secure would be to keep the communication between the involved parties as minimal as possible. The design of our protocol aims for the same where the transactions between Alice and Bob happen only twice - one forward and one backward message exchange. Alice sends the encrypted text along with public key parameters to Bob. Bob homomorphically evaluates the function using the available information and sends the encryption of the end result back to Alice. The next sections in the chapter discuss about the correctness of the protocol and the hardness proof. Correctness of the protocol enables us to understand that even if the entire computation is carried out under encryption, the end result is the same as if it were carried under plain text settings. Hardness proof on the other hand deals with the proof that the protocol is difficult to break. The hardness of our protocol depends on the blinding hiding properties of our evaluation function, so we proceed the discussion by understanding these properties and developing the proof of hardness based on these properties.

6.1 Protocol

This protocol specifies how we can apply our $\mathsf{CS} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for secure function evaluation in restricted domains. There are many public key approaches for secure function evaluation, however, we are able to make a more efficient approach for this by exploiting the additive homomorphic properties of CS.

Public: Composite modulus $n \mid n = p^2q$, generator $g \in \mathbb{Z}^*{}_n$, sequence initial value and step size α, β

Private Keys: p generated by Alice based on the co-domain of $g(i) : \mathbb{Z} \mapsto \{0,1\}^\ell$

Inputs: Alice- x, Bob- $y \mid x,\ y \in \mathbb{Z}_{2^k}$

Output: Alice and Bob want to evaluate a function g where g is the required linear embedding function such that, $g : \mathbb{Z} \mapsto 0, 1^\ell$ on their inputs. Alice will either learn $g(m) \mid m = x + y$ or the truth condition of some function where $\chi(x, y) : \mathbb{Z} \mapsto \mathbb{Z}$, which is in turn evaluated using $g : \mathbb{Z} \mapsto 0, 1^\ell$. Bob outputs $\perp$

1. Alice encrypts x as c_A, β as β' and sends $\{c,\ \alpha,\ \beta',\ n,\ g,\ h\}$ to Bob. Bob encrypts y with the provided public key.

2. Bob computes $\chi(x, y)$ homomorphically tapping the additive and scalar multiplicative properties of the given public key cryptosystems as c_B. Or Bob encrypts m as c_B.

3. Now, given $r_c \in \mathbb{Z}_k$, Bob computes $c^{'} = ((c_B)^\alpha \cdot \beta^{'})^{r_c^2}$.

4. Alice decrypts c' using the decryption algorithm. Even after decrypting, Alice will not recieve the plain text $\chi(x, y)$ or m, rather Alice receives some random number of the form $m' = (\alpha \cdot \chi(x, y) + \beta) \cdot r_c^2$ (or $m' = (\alpha m + \beta) \cdot r_c^2$)as hidden by r_c^2.

5. Alice computes the Legendre Symbol of m' giving us 0 or 1 depending on the plain text value of the function.

6.1.1 Correctness of the protocol

We wish to prove that the protocol introduced in this section correctly evaluates a function of the form $\mathbb{Z} \mapsto 0, 1^{\ell}$ during the final step. From theorem 4.3.1 we established that $Dec(c^{'})$ returns $g(m)$. Replace m with $\chi(x, y)$ which gives us

$$Dec(c^{'}) = g(\chi(x, y)).$$

∎

6.2 Blinding Hiding Properties of Evaluation Function

The security of our evaluation depends on the blinding hiding properties of evaluation function. During evaluation we hide our ciphertext using the square of a random element $r_c \overset{R}{\leftarrow} \mathbb{Z}_{2^k}$. To establish security, we need to prove that, given the decryption of cipher text hidden by r_c^2 it is hard to tell whether it belongs to the set of *QR* or *NR*. Mathematically, we rely on the decisional problem of given c' and the public elements, it is hard to decide whether $Dec(c') = \mathbf{QR}$ or $Dec(c') = \mathbf{NR}$, where **QR** indicates the set of Quadratic residues $\pmod p$ and **NR** indicates the set of Non-Residues $\pmod p$. If we can establish that this decisional problem is hard, then the blinding hiding property of Evaluation function is established.

6.2.1 Hardness Proof

Theorem 6.2.1 *Given $r_c \overset{r}{\leftarrow} \mathbb{Z}_k$, $h(x) = \alpha x + \beta$ and $c^{'}$, $Dec(c^{'})$ is uniform across the set of Quadratic Residues $\pmod p$ if $h(\chi(x, y))$ is a Quadratic Residue and $Dec(c^{'})$ is uniform across the set of Non-Residues $\pmod p$ otherwise.*

Proof From the section 3.2 we know $f(x) \neq 0 \pmod p$. Let us assume that after applying key generation algorithm from section 4.1 we get p which is of the form $2^k i + 1$ for some odd i. This implies the prime p consists of two subgroups $\mathbb{G}_{2^k}$ and $\mathbb{G}_i$ with orders 2^k and i and generators g_k and g_i respectively. Now, for some $0 \leq y < 2_k$ and $0 \leq z < i$, any element $a \in \mathbb{Z}_p$ will be of the form

$$a = g_k^y \cdot g_i^z \pmod p.$$

Applying this to $Dec(c^{'})$ we get the following set of equations:

$$h(\chi(x,y)) = g_k^{y_1} \cdot g_i^{z_1} \pmod{p}.$$

$$r_c = g_k^{y_2} \cdot g_i^{z_2} \pmod{p}.$$

This gives us

$$\begin{aligned} h(\chi(x,y)).r_c^2 &= (g_k^{y_1} \cdot g_i^{z_1}) \cdot (g_k^{y_2} \cdot g_i^{z_2})^2 \pmod{p} \\ &= (g_k^{(y_1+2y_2)} \cdot g_i^{(z_1+2Z_2)}) \pmod{p}. \end{aligned}$$

We know since $r_c^2 \xleftarrow{UR} \mathbb{Z}_{2^k}$, and $2z_2 \xleftarrow{UR} \mathbb{Z}_i$ which implies $z_1 + 2z_2 \xleftarrow{UR} \mathbb{Z}_i$. Similarly, since $2y_2 \xleftarrow{UR} \mathbb{Z}_{2^k}$, $y_1 + 2y_2$ is uniformly random in the set of even numbers modulo $2^k \iff y_1$ is even and $y_1 + 2y_2$ is uniformly random in the set of odd numbers modulo $2^k \iff y_1$ is odd. ∎

Chapter 7

Privacy of the Secure Function Evaluation Protocol

This chapter will throw a light on the privacy of the secure function evaluation protocol introduced in chapter 6. A private protocol ensures that the computation takes place correctly while protecting the information of the parties that are involved. We begin with introducing various adversary models present in a security setting. We then carry out our privacy proofs under Semi-Honest Adversary setting also known as Honest-But-Curious Model. We then provide the basic definitions necessary to discuss our privacy proofs. We relied on simulation based proofs for the privacy of protocols both in Alice and Bob's perspectives. The last section focuses on the privacy proofs for protocol by considering Alice and Bob's data privacy separately.

7.1 Adversary Models

Apart from defining the essential components that constitute any Secure Function Evaluation, it is important to define the settings under which these properties will hold true. There are many models that exist in literature depending on the honesty level exercised by the adversary. Defining these honesty levels will change the definitions of the privacy expected under these settings.There are two main types of adversaries, which are discussed below along with what are the pre-requisites for privacy under these adversaries.

Semi-Honest Adversary

This adversary adheres to all the steps specified in the protocol, but looks for any information that may be leaked in the intermediate steps. These adversaries are assumed to be selfish, meaning they will take any steps to benefit themselves from the advantageous information leaked during the intermediary processes.

Malicious Adversary

This adversary is assumed to deviate anytime from the protocol as per their benefit. Deviation includes many aspects as mentioned in [61] such as proving deceptive values, aborting a protocol in their own time and taking any action that helps the achieve the desired results. It is difficult to achieve security against these kind of adversaries. Hence, it can be easily stated that any system that is secure against *malicious adversaries* is secure against *semi honest adversaries.*

Now, let us assume there are n number of users. Each *ith* user possesses some x_i such that:

$$x_i \mapsto \mathbb{Z}$$

In order to calculate a function F_i which in our case $F_i : \mathbb{Z} \mapsto (0,1)^n$. Our goal is to construct a protocol such that each of the user knows $F(x_1, x_2, ..., x_n)$ but none of them knows anything more about x_j for $j \neq i$. Based on the privacy for both the adversary models described above can be defined.

7.2 Basic Definitions

Privacy of the protocol Perfect Privacy under semi honest adversary can be achieved by hiding all the intermediate data and processes to other party. Also known as Honest But Curious adversary model, privacy under this model assumes that all the n users follow the protocol. Now, the protocol is k private, if any of the k parties collude with each other learn nothing more than the outputs.

Privacy under Malicious Adversary Under this setting, the adversary will control k users. This implies the rest of the n-k users will continue to remain honest. Now the protocol under

Malicious Adversary will be k private if the adversary does not learn anything from the parties that they do not control; i.e., the inputs of the n-k users will not be learnt at all. But the adversary may learn about the outputs of the corrupt parties.

Zero Knowledge Proofs These are protocols by which one party can prove to another party that it has the knowledge of x without letting the other party know anything about x, other than the fact that the knowledge of x is true.

View We define a view to work under semi-honest setting. Let χ be a function evaluated by a protocol π. A view of a party can be defined as the set of elements that can be seen by the party during the execution of π. Let X, r, Y be inputs,random values and outputs exchanged during the complete transaction of π. We define views with respect to each parties as :

$$View_{Alice} = (X_A, r_A, Y_A).$$

$$View_{Bob} = (X_B, r_B, Y_B).$$

7.3 Privacy Proofs for π

We now prove that π performs secure function evaluation under semi-honest adversary using a simulation based approach. The key idea behind this simulation is to create a protocol view of the parties, without the knowledge of any keys. To prove the security of the π, we establish that the distribution of the resultant protocol view $View_\pi$ and π are indistinguishable. This is equivalent to the cipher-text indistinguishability of a cryptosystem. Based on section 7.2 the privacy in a two party setting in the presence of a semi-honest adversary is established if neither *Alice* nor *Bob* do not learn anything other than the intended outputs Y_B and Y_A respectively. Our proofs are divided into two parts: Alice's privacy, demonstrated by simulating $[View_{Bob}, Y_A]$ and Bob's privacy demonstrated in the applications where *Bob* outputs $\perp$ and *Alice* only learns the truth value, which will be done by simulating $[View_{Alice}, \perp]$.

7.3.1 Alice's Privacy

Theorem 7.3.1 (Alice's privacy) *There exists polynomial time algorithm A_B such that it simulates $View^*_{Bob}$ whose distribution is indistinguishable from that of original view.*

Proof The proof of *Alice*'s privacy is simple because *Bob* only receives the cipher-text $c = Enc(x)$, i.e., $Y_A = c$. This implies, the indistinguishability of distribution of $View_{Alice}$ is equivalent to the cipher-text indistinguishability of the cryptosystem itself. Since the cryptosystem is semantically-secure as established in theorem 5.3.1, the ciphertext is also indistinguishable from any random number. The view of Y_A can be simulated by simply choosing any random number $r_n \overset{R}{\leftarrow} \mathbb{Z}_n^*$. Due to semantic security of the cryptosystem itself, the advantage of an adversary guessing whether it is r_n or Y_A is negligible. ∎

7.3.2 Bob's Privacy

Theorem 7.3.2 (Bob's privacy) *There exists polynomial time algorithm A_A such that it simulates $View^*_{Alice}$ whose distribution is indistinguishable from that of the original view* = [$View_{Alice}$, $\perp$].

Proof The privacy of *Bob*'s output, c', relies on the hider $r_c \overset{R}{\leftarrow} \mathbb{Z}_k$ where k is the bit-size of the prime p. *Alice* decrypts c' and receives $m' = h(\chi(x, y))r_c^2$. Now, A_A picks some $m_r \overset{R}{\leftarrow} \mathbb{Z}_p$, as *Alice* has access to the private key. Applying $f(x)$ from section 3.2 to m_r we get $g(m_r)$ which may be either a Quadratic Residue or a Non-Residue depending on the value of m_r. Applying theorem 6.2.1, we have that $Dec(c')$ is uniformly distributed among the set of Quadratic Residues and Non-Residues modulo p, which implies it is hard to distinguish between m_r and m'. ∎

Chapter 8

Application of our Protocol to Privacy Preserving Similar Patient Query

This chapter introduces a medical application called Similar Patient Query, which is based on our $\mathsf{CS} = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$. The increase in the availability of genomic data holds a great promise for the advancement of personalized medicine. The approach in personalized medicine relies on understanding how the genetic make up of each person impacts their reaction to different medications and makes them susceptible to certain diseases. Even though genomic data is not intrinsically exceptional, there is a belief that it needs to be handled with care because it has a lot of features that make it extremely useful in a wide range of applications. The privacy issues associated with genomic data are quite complex due to this very reason. The problem here is, the breach of genomic data can reveal more information than the information from which the genome was sequenced. A genomic data can link a whole family tree or populations together. This will result in information about diseases that a person or an entire community is susceptible to, garnering unnecessary attention from insurance companies and scientific research communities. There are numerous researches conducted in the aspect of genomic data privacy by [63] [62] [39] [15] [44][4]. In fact, [46] conducted a detailed survey on the issues and concepts related to privacy in the era of genomic data explosion emphasizing the need for having privacy preserving mechanisms for secure handling, storage and computations on genomic data. Similar Patient Query (SPQ) is one such privacy preserving application that involves performing Secure Multiparty Computations on Genomic data. The coming sections

focus on the basics of SPQ, previous related work in this field, limitations of the existing approaches, system model, similarity measures, protocol for secure computation of Euclidean distance and finally, we present a protocol to apply our cryptosystem introduced in the previous chapters to perform Similar Patient Query and its advantages.

8.1 Similar Patient Query

Imagine there is a doctor that wants to compare an individual patient's data, let us call it a *query*, across a whole database *DB* that consists of individual genomic sequences that are labelled with their medical history. Now the doctor wants to find out which of the sequences in the database are similar to the *query* sequence. The result is a set of similar patients whose genomic sequences resemble the query sequence. The doctor can use this information either to estimate the possible disease onset, reactions to medications, and a wide range of other medical applications. In short, Similar Patient Query can be defined as a query carried out by some client/ user over a server which is a database of patients. The SPQ can also be carried out between two different individuals, the comparison need not be between just a query and database. This sort of applications are extremely useful in identifying cancer sub types as they are unique. SPQ will help in recognizing the mutations behind these cancers and also the related reactions for the treatments offered. That way, the doctor will know in which direction they can proceed to treat the patient. The similar patient query need not be restricted to medical setting alone, it an be used for other applications such as exchanging genomic data between different research groups and private genetic testing companies.

The most popular measures for calculating similarity between two patients' genomic data are Euclidean distance, Pearson correlation, and edit distance. There are several privacy implications related sending the patients' DNA as a query in plain text. Hence, SPQ needs to be carried out under private settings. Thus most of the literature surrounds around formulating efficient methods to calculate the similarity measures privately. In our research, we used a special technique to calculate Euclidean distance in a privacy preserving manner, which will be discussed in the coming sections.

8.1.1 Related Work

Similar Patient Query under privacy preserving settings has been extensively researched in [10] [3] [67] [58] [38]. The key observation is that all of these studies focus on calculating similarity between two genomes by taking the overall genomes into consideration. This can adversely affect the time and computational complexity. The major idea for this application was drawn from [56], where similarity in genotype data is calculated only by focusing on particular loci in the chromosomes as opposed to the entire genome. As discussed earlier, alleles present in this loci have a tremendous impact on patients' response to a treatment. To establish similarity in the genotype data many have been conducted to calculate Euclidean distance and Pearson correlation in privacy preserving manner by [53] [65] [42] [71] [18] [56]. Several other studies such as [3] [74] [2] [67] computed Edit distance as a similarity measures in privacy preserving settings. Our protocol only relies on Euclidean Distance as it allows secure computation of similarity in a much simpler way contrary to the approaches mentioned in the existing studies.

8.1.2 Disadvantages of Existing Approaches

In Similar Patient Query, there are two parties in picture: User who poses the query and a Database held by the hospital. So privacy for a simple Secure computation can be established by considering two scenarios : User privacy and database privacy. All the studies discussed in section 8.1.1 ensure user privacy completely, as it is dependent on the cryptosystem itself. This topic is discussed in detail in chapter 7 while working out Alice and Bob's privacy proof's with respect to our protocol. Assume here that Alice the user is holding the query and Bob holds the hospital database. Coming to Bob's privacy, the existing protocols return similarity scores once Alice decrypts them. Now, the data held by Bob may not be leaked directly, but there is a potential risk of inferring the database contents from the similarity scores. Two potential attacks that can be carried out using similarity scores have been discussed by [60]. These attacks include Regression attack and illegal query attack.

Under Regression attack, the similarity score between some data point and a target can be used to identify the contents of a target. As the number of these data points increase, the probability of identifying the target content will also increase accordingly. For example if

Bob returns the exact value of Euclidean distance each time Alice sends a query, it can be eventually narrowed down to finding out what exactly was Bob's data using multiple attempts. Hence, it is important to give out as minimum information as possible from Bob's side to avoid regression attacks. Similarly, illegal query attack can cause Bob's response to keep varying with increasing queries. These variations can again be used to retrieve the target data.

8.2 System Model

Our protocol for SPQ assumes that the two parties involved in the setting are Hospital and User. Here the Hospital possess the database related to multiple patients' genome. In practice there could be any number of hospitals and the entity need not be a hospital at all, it can be any organization or research institute that can collect and store genomic data. User could be a doctor or some researcher, who holds a query sequence which has information about: the location on the genome which they want to compare. Here we are aiming to calculate similarity between two sets of data. Hospital holds the data about few patients and the user holds data about some particular patient. The hospital cannot learn about User's data and vice versa. So in our current setting, Hospital receives encrypted User's data which is semantically secure. User receives only the truth condition of the similarity measure. Please note that sometimes the terms User and Querier are used interchangeably for this protocol.

The threshold function is similar to the one discussed in section 2.4. Given similarity measure s and threshold value t, we try to return $0/1$ depending on the value of s. The prime number generation will also be modified according to t because, based on t the domain of $g(i)$ shall become something like $\{0, 0, 0, 0, ..., 1, 1, 1, 1\}$. The protocol is discussed in detail in section 8.5. Ultimately, we are trying to establish that due to the application of evaluation function from section 4.3 the application of SPQ is semantically secure against a Honest but curious user or in a setting of semi-honest adversary.

8.3 Similarity Measures for SPQ using Genomic Data

Extensive research has been conducted in the area of conducting Similar Patient Queries using Genomic data in privacy preserving settings by [67] [3] [45]. But these researches concentrate on finding the similarity between two organisms by comparing the genome as a whole. In reality, the whole genome need not be compared because if we take the case of human genome, 99 percent of the humans are similar. It is the mutations at particular locations on a chromosome that make the humans different. Each organism consists of a gene, also known as genotype, at a specific location on their genome called locus. The variations in these loci cause variation in the physical traits, also known as phenotype, of the organism. These variations are called as allelic variations, where alleles are the variant form of a gene. These alleles represent mutations at the same locus, which are usually two or more versions. Different populations consist of different alleles, so we use something called allele frequency to measure the relative frequency a particular allele at that locus in a given population.

Whole genome sequencing has enabled researchers to identify that there is a relation between a myriad number of diseases and the genetic variations at a particular locus in an organism. They are known to affect diseases like diabetes, cancer and other cardio vascular disorders. Hence, it becomes easier to compare patients using specific variants at a given locus rather than comparing two whole genomes that contain about 3 billion base pairs. This functionality is useful in various areas such as patient's response to treatments or onset of a disease to name a few. For example, the application using Similar Patient Query based on comparing genotype variants only can typically work on a query like: "What is the reaction of an HIV patient to the medicine abacavir with a genotype variant HLA-B*5701?" Research has established that HIV patients with HLA-B*5701 gene variant indeed have a hypersensitive reaction to abacavir than those patients that do not possess this gene variant. So it is important to find out the similarity across the loci where these variants occur.

To calculate similarity, we use Euclidean distance. Let I define the set of all the loci that a user is interested to compare. x, y are the vectors for which we want to calculate the similarity. So the Euclidean distance can be represented as:

$$\sum_{i \in I} (x_i - y_i)^2$$

Technically, the actual Euclidean distance is the square root of the value stated above. But we establish similarity between two patients only if the Euclidean Distance between them is less than or equal to some predefined threshold value. So, it suffices to square the threshold value and use it for comparison.

8.4 Protocol for Secure Computation of Euclidean Distance

In this section we describe a protocol for secure computation of Euclidean Distance denoted as SecED. As we discussed in the previous section, we calculate similarity between two patients by taking on specific loci on the genomes. For obtaining variant data, we use genomic data from a file format called as Variant Call Format denoted as VCF. Homozygous allele means the chromosome has two same copies of the same trait (dominant or recessive) and Heterozygous allele has two different copies. In these files, the genotype variants at specific loci are primarily of three types : Homozygous Dominant, Homozygous recessive and Heterozygous alleles which are indicated as $0, 1, 2$ respectively. Dominant or recessive alleles indicate the type of trait. So we convert the data into a new message space : $\{0, 1, 2\}^n$. n indicates the number of locations at which the genotype data will be compared.

Since the message space is limited, we pre-build a look-up table that consists of various combinations of x_i, y_i from the formula for Euclidean distance. Note that our cryptosystem only supports Partially Homomorphic Encryption scheme which supports addition and scalar multiplication. Since Euclidean distance is a sum of squares, we try to pre-calculate all the possible squares and encrypt them newly for each string, so that hospital only needs to perform homomorphic addition to calculate similarity. Let ℓ be the length of user's *query*. The look up table would be as shown in table 8.1. The user encrypts each string separately in order to maintain the semantic security. And each time the user wants to conduct SPQ, they will send 3ℓ combinations of data as opposed to ℓ for each number, so that the hospital will pick the encryption accordingly to calculate the summation for Euclidean distance. For example, in case the user wants to send hospital an encryption of $\{1, 1, 2\}$, the user will start creating an array of encryptions for each element in the query string. For the first element in query string user picks up row related to 1 as shown in fig. 8.1 and encrypts each element from the

Genotype	0	1	2
0	$(0-0)^2$	$(0-1)^2$	$(0-2)^2$
1	$(1-0)^2$	$(1-1)^2$	$(1-2)^2$
2	$(2-0)^2$	$(2-1)^2$	$(2-2)^2$

Table 8.1: Lookup table

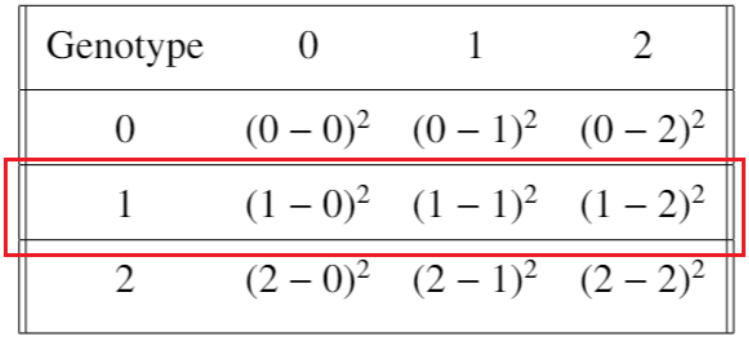


Figure 8.1: Choice of row for string 1

row separately. For the second element in the query string, user chooses the same row and performs the encryption for the second time. Due to the probabilistic nature of the schemes used for encryption, we get different encryptions for $\{(1-0)^2, (1-1)^2, (1-0)^2\}$ each time they are encrypted. Similar process is applied for the third element 2 in the input sequence string. Overall 9 encryptions are sent as opposed to 3.

User sends the 9 encryptions by indexing them for the hospital. Let us denote the notations for an encrypted string as shown in fig. 8.2, double braces indicate the encryption of the string which is a random number. Let the hospital hold a string $\{0, 0, 1\}$, so they will choose from the received encryptions as shown in fig. 8.3. Once all the encryptions are received the next step is just to calculate the summation over the relevant encryptions. Since the scheme we are using is

0	1	2	0	1	2	0	1	2
$[[(1-0)^2]]$	$[[(1-1)^2]]$	$[[(1-2)^2]]$	$[[(1-0)^2]]$	$[[(1-1)^2]]$	$[[(1-2)^2]]$	$[[(2-0)^2]]$	$[[(2-1)^2]]$	$[[(2-2)^2]]$

Figure 8.2: Hospital's view of encrypted string

0	1	2	0	1	2	0	1	2
$[[(1-0)^2]]$	$[[(1-1)^2]]$	$[[(1-2)^2]]$	$[[(1-0)^2]]$	$[[(1-1)^2]]$	$[[(1-2)^2]]$	$[[(2-0)^2]]$	$[[(2-1)^2]]$	$[[(2-2)^2]]$

Figure 8.3: Encryption chosen by the hospital

additively homomorphic, it is easy to calculate summation of all the bits by simply multiplying all the encrypted outputs. Once decrypted, this results in a plain text sum as per the properties of additive homomorphism.

8.5 Private SPQ Protocol

This protocol specifies how we can apply our $\mathsf{CS} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for carrying out Similar Patient Query. Here, hospital and user want to calculate a function to find the similarity between the patients. The user inputs the query string for which they want to calculate the similarity. The query string is drawn from message space $\{0, 1, 2\}^n$. The user follows the protocol for secure calculation of Euclidean distance and then carries the complete transaction similar to the protocol introduced in section 6.1. The complete steps in the protocol along with the input and expected output are described in fig. 8.4.

8.5.1 Advantage of using Our Approach

Let us assume the case where we do not apply the Evaluation Scheme from section 4.3. This would imply, hospital will omit step 4 from the Private SPQ protocol and sends the result calculated from step 3. In other words, the user would directly receive the encryption of Euclidean Distance. As the $User$ applies decryption, they receive: $\mathsf{Dec}(\mathsf{Enc}(ED)) = ED$. For each individual $Query$, q_i, User receives a corresponding ED_i. If User can collect enough ED_is, they

Public: Composite modulus $n \in Z$, generator $g \in \mathbb{Z}^*_n$

Private Keys: The Boolean co-domain is predetermined based on the threshold value. User will generate the large prime number p, according to the section 4.1.

Inputs: User's input is *query* sequence, Hospital inputs the database. Both $x, y \in \{0, 1, 2\}^n$

Output: User wants to find out if the *query* matches with any sequence from DB by securely evaluating the encrypted Euclidean Distance. Hospital outputs $\perp$

1. For each character in a string, user makes use of table 8.1, and sends three encryptions from first row for 0, second row for 1 and third row for 2.The encryption will carried out separately each time, so that semantic security is ensured. Let us assume the string sent by the user is c_U. So the transaction would involve in the transfer of $\{c_U, \alpha, \mathsf{Enc}(\beta) = \beta', n, g, h\}$ to the Hospital.

2. The hospital picks the suitable encryption depending on the value they are holding from the column in table 8.1 and forms a new cipher text c_H.

3. The hospital computes $\mathsf{Enc}(ED) = \sum c_H$

4. Now, given $r_c \in \mathbb{Z}_{2^k}$, hospital computes $c' = ((\mathsf{Enc}(ED))^{\alpha} \cdot \beta')^{r_c^2}$.

5. User decrypts c' using the decryption algorithm Dec. Even after decrypting, User will not recieve the plain text ED, rather Alice receives some random number of the form $(\alpha \cdot (ED) + \beta) \cdot r_c^2$ as hidden by r_c^2.

6. User can find out $g(i)$ depending on the above decrypted value. If the patients are similar, then $g(i) = 0$ otherwise it would return 1

Figure 8.4: Private SPQ Protocol

can easily use the data to extrapolate and find out the accurate data points that hospital owns. This is where applying evaluation function will eliminate the possibility of carrying out these type of regression attacks. From the blinding hiding properties of evaluation function, we can establish that multiplying r_c^2 to the cipher text can actually hide the calculated ED_i. The evaluation function will only display the truth value for similarity. To explain it mathematically, let t be a threshold value used to express the similarity between two patients such that if $ED_i \geq t$ the patients are similar otherwise they are not similar. So our Evaluation function Eval shall display a 0 for $ED \leq t$ and 1 for $ED > t$.

8.6 Computational Hardness of SPQ protocol

We would like to establish that the protocol described in section 8.5 is difficult to break. The entire protocol has the following steps:

- Encrypted query from the user, that uses Secure Computation of Euclidean Distance protocol
- Homomorphic addition by the hospital
- Decryption and Evaluation of Euclidean distance by the user

The hardness of second and third steps from the protocol have already been established. They are the same as the privacy proofs worked out using simulation technique with respect to Alice and Bob's view of the protocol in chapter 7. If we are able to establish that the Secure Computation of Euclidean Distance protocol is hard to break, then it is automatically implied that the SPQ protocol itself is difficult to break using a polynomial time algorithm. So our proof for hardness of SPQ protocol is established by proving the hardness of SecED described in section 8.4. SecED is performed on three plain text elements $\overset{R}{\leftarrow} \{0, 1, 4\}$, which are the final values of the combinations presented in table 8.1. As discussed earlier,the elements are encrypted freshly every time a transaction takes place. Due to the probabilistic nature of the cryptosystem, the encryptions generated will be random each time, making it difficult to break the protocol and ensuring semantic security.

Chapter 9

Implementation

This chapter talks about the implementation of the key generation, or generation of p and the protocol introduced in chapter 8. For p generation, we used Sage math environment, which is an open source library based on python. The entire cryptosystem depends on basics from number theory such as primality testing, CRT and testing for Legendre symbols. All of these have been constructed as separate functions as described in the algorithms. We analyse the performance of our cryptosystem by comparing it with the cryptosystem introduced in [20].

For the second part of implementation, we show the application of Similar Patient Query. One of the crucial aspects with respect to handling computations on genomic data is understanding the genomic data set. So, a subsection is dedicated to briefly describing how a genomic database looks like and how to read its contents. Even though the data set reading in plain text is done by the individual parties before the transaction is carried out, it is important that both the parties understand the features associated with it.

9.1 Experiments and Results for p generation

This section discusses about the backbone of our cryptosystem, which is the key generation. These are dependent on the basics introduced in chapter 3. The first algorithm produces suitable α, β such that sequences of the form $(\alpha i+\beta)$ have elements in them, when factorized, the factors do not repeat. Based on this sequence, we produce p in the second subsection.

9.1.1 Picking the best α and β

From the theorem 3.2.1 and theorem 3.2.2, we can conclude that in order to develop this new cryptosystem, we need an efficient algorithm to generate the following: α, β, p. We discuss two separate algorithms: section 9.1.1 for α, β generation and algorithm 2 for p. The key idea behind Section 9.1.1 is to find the starting number and step size for a sequence where maximum number of values contain unique factors as we factorize. While discussing theorem 2, we assumed all the sequence elements would be prime valued. But this algorithm will also consider a generic case, where the sequence can be prime or composite. For this, a simplistic approach has been applied, where a random $\alpha, \beta \in \mathbb{Z}$ are picked. A sequence will be generated with α as the beginning value and β as step size. The length of sequence is equal to the length of the required Linear Embedding Function to be evaluated. Now we traverse through each element $s_i \in \textit{Sequence}$, we pick only those factors that are not squares to decide the Legendre symbol.

We use a Boolean variable named new factor seen and set it to true as a new factor is noticed in the list every time. Every time a new element is found, it will be added to the list named factors. Then a counter is updated to check for the current size of the list. The list will be exited as soon as the current size will stop changing. Once exited, all the variables are set back to their default values. This way Algorithm 1 can be implemented for a wide range of α, $\beta \in \mathbb{Z}$. In order to implement this, the simplest way is to use a nested for loop and iteratively check across various combinations of α, β and pick the ones that produce the primes of reasonable bit-size for a given Boolean function that needs to be evaluated.

Input: Size of the Quadratic Residue Function- i, a,b

Output: For a random length of QRF upto i their respective α, β, bit-size of the prime p

function Searchloop(α, β)
  $sequence \leftarrow 0$
  $i \leftarrow Length(QRF)$
  for $k \in \{0, i\}$ **do**
    $sequence \leftarrow \alpha i + \beta$
  end for
  $factors \leftarrow [\ ]$
  $j \leftarrow 0$
  for $s_i \in sequence$ **do**
    $y \leftarrow Factorization(s_i)$
    $currentsize = length(factors)$
    $lengthoffactor \leftarrow length(factors)$
    $newfactorseen \leftarrow False$
    for $fact \in y$ **do**
      if $fact \neq square$ **then**
        if $fact \notin factors$ and $newfactor = False$ **then**
          $factors.append(fact)$
          $j = j + 1$
          $newfactorseen = True$
        end if
      end if
      if $length(factors) \not> currentsize$ **then**
        $Exit$
      end if
    end for
  end for
end function

Algorithm 1: Finding α, β

9.1.2 Key Generation Implementation

Input: For a random length k of QRF upto i their respective k, α, β, required QRF ℓ_i

Output: Prime factor p

for $i \in 1, k$ **do**
    $sequence = \alpha i + \cdot\beta$
end for
function SYSTEMOFCONGRUENCES(sequence)
    $M \leftarrow MatrixSpace(GF(2))$
    $Primefactor = factorize(s_i)$
    for $s_i \in sequence$ **do**
        $Mrows \leftarrow s_i$
        if $Primefactor \in s_i$ **then**
            $Mcolumns \leftarrow 1$
        end if
    end forreturn M
end function
$M' \leftarrow EchelonForm(M)$
for $row \in M'$ **do**
    check for consistency of the system
    if Consistent **then**
        return True
    end if
end for

Algorithm 2: Finding p

```
    B ← {}
22: for a_i ∈ Primefactors do
        b_i ∈ 1, a_i − 1
24:     if legendre(b_i, a_i) = ℓ_i then return b_i
        end if
26:     B.append(b_i)
    end for
28: function ComputeP(B, Primefactors)
        p = CRT(B, Primefactors)
30:     A = prod(Primefactors)
        while not prime(p) or p ≢ 1 (mod 4) do
32:         p = p+A
        end while
34:     return p
    end function
```

Algorithm 2 shows a novel method for prime generation, which forms the backbone of the new cryptosystem we are trying to build. Once the client gives out the Boolean Function domain to be evaluated, we apply theorem 3.2.1 and theorem 3.2.2 to find the required prime number. In order for the CRT to work efficiently, we need a consistent system of linear congruences. To test for consistency, we use the Matrix space and convert them into echelon form. Using matrices to solve the system of congruences improves the performance of a system with respect to time and computational complexity, the detailed steps to which are explained in the next section.

Using Matrix Space for the linear system

This section explains how we use a matrix space to test for the consistency of the linear system of congruences we are trying to develop using the key generation algorithm developed in section 4.1.

Inconsistent or Consistent Systems A system of equations is called inconsistent if it has no solution. A system which has a solution is called consistent.

s_i	Factors
23	23
27	$3 \cdot 3 \cdot 3$
31	31
35	$5 \cdot 7$
39	$3 \cdot 13$
43	43
47	47
51	$3 \cdot 17$
55	$5 \cdot 11$
59	59

Table 9.1: Sequence of elements for $\alpha = 4$ and $\beta = 23$

Once we find a sequence of elements with each of them having at least one unique factor, we build a matrix where each row represents the sequence number and each column represents the prime factor. Let us consider a sequence formed with $\alpha = 4, \beta = 23$ that consists of 10 elements having at least one unique factor each. We then proceed to following steps:

- **Step 1** Generate the sequence of elements and factorize each element to compute all the unique factors present in the sequence. The sequence with its factorizations is presented in table 9.1.

- **Step 2** The rules for forming a factor base are as follows:

  - Even powered factors are omitted out, i.e., for some factor a, $\{a^2, a^4, a^6, ..\}$ need not be considered because based on the definitions of Quadratic Residues and Legendre symbols, they do not affect the Legendre symbol of the sequence element.
  - We add every factor to the factor base if it has not appeared previously, so only unique factors are considered by avoiding repetitions.
  - Do not consider a sequence that may consist of 2 in its factor base, as having 2 can lead to inconsistencies in solving the Chinese Remainder Theorem.

Based on these rules we obtain the factor base as:

$$[23, 3, 31, 5, 7, 13, 43, 47, 17, 11, 59].$$

- **Step 3** We convert it into a matrix space in Galois Field, GF(2) such that the matrix consists of all the elements (mod 2). The rows of the matrix represent the elements of the sequence and the columns represent prime factors from the factor base list. A 0 or 1 indicates the absence or presence of the factor in the sequence elements respectively. So the matrix for the sequence elements from table 9.1 and factor base displayed in step 3 looks as follows:

$$\begin{array}{c} \\ 23 \\ 27 \\ 31 \\ 35 \\ 39 \\ 43 \\ 47 \\ 51 \\ 55 \\ 59 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 23 & 3 & 31 & 5 & 7 & 13 & 43 & 47 & 17 & 11 & 59 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

As per the algorithm, we append the required function to be evaluated as the last column in the matrix. For example, we would like to evaluate a threshold function, with $t = 0.5$ for SPQ whose co-domain would look like:

$$[0, 0, 0, 0, 0, 1, 1, 1, 1, 1].$$

In other words, we are looking for a prime p such that

$$\begin{aligned}
\left(\frac{23}{p}\right) &= 1 \\
\left(\frac{27}{p}\right) &= 1 \\
\left(\frac{31}{p}\right) &= 1 \\
\left(\frac{35}{p}\right) &= 1 \\
\left(\frac{39}{p}\right) &= 1 \\
\left(\frac{43}{p}\right) &= -1 \\
\left(\frac{47}{p}\right) &= -1 \\
\left(\frac{51}{p}\right) &= -1 \\
\left(\frac{55}{p}\right) &= -1 \\
\left(\frac{59}{p}\right) &= -1.
\end{aligned}$$

So the matrix would be represented as:

$$A = \begin{array}{c} \\ 23 \\ 27 \\ 31 \\ 35 \\ 39 \\ 43 \\ 47 \\ 51 \\ 55 \\ 59 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 23 & 3 & 31 & 5 & 7 & 13 & 43 & 47 & 17 & 11 & 59 & g(i) \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

- **Step 4** We need to convert matrix A into a row reduced echelon form (RREF) based on [64] to test for consistency by performing certain elementary row operations. The requirements for a matrix to be in REF are as follows:
  - The first non-zero number entry of the first row should be 1, this is the leading entry of the row present in the matrix position $(1, 1)$.
  - The second row should also have 1 as its leading entry, but this entry should be further to the right of leading entry in the first row. Similar process is repeated for every subsequent row, where the first non-zero number in every row is 1 and is placed further to the right of the leading entry of previous row.
  - The leading entry of each row must be the only entry of its column.
  - Any non-zero rows are usually placed at the bottom of the matrix.

  Keeping these requirements in mind, we try to obtain the matrix in RREF using the following steps:
  - We begin with finding the pivot row, i.e., from the matrix A we identify the first non-zero entry in the first column of the matrix. This is present in the first row

itself and the leading element is 1, i.e., the first row itself is pivot row with leading entry as 1. So we do not need to pivot.

- We repeat pivot for the remaining rows by ignoring the previous ones by continuing the process until no more pivots are left. For the row where element = 39, where the row is R_5 we perform a operation such that the first entry would be in $5th$ column. So we apply row operations and carry on the same procedures for the rest of the rows.
- We check if all the elements of the rows are zero, except the last column, if that is the case, then the system is inconsistent. The complete row operations are present in chapter A.

- **Step 5** We check for consistency of the equations in this step. Matrix A in echelon form would be represented as:

$$A_{echelon} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

If a system is inconsistent, the REF matrix obtained will consist of a row of the form $\{0, 0, 0, ..., 1\}$, i.e., will have a leading 1 in its right most column. Such a row corresponds to an equation of the form:

$$0x_1 + 0x_2 + 0x_3 + ... + 0x_n = 1.$$

which definitely has no solution. Since the matrix $A_{echelon}$ has no such rows, the system of equations are consistent and certainly have a solution.

- **Step 6** We take the $A_{echelon}$ matrix and traverse through it row wise and check for each non-zero row.The leading entry of each of these rows indicate the "deciding factors". So the Legendre symbol of the sequence element is decided by the factors represented in these leading entries. The remaining entries can be categorized as "Unutilized factors" as they do not impact the overall Legendre symbol of the sequence element. We convert the last column in these rows to Legendre symbols such that $0 \mapsto 1$ and $1 \mapsto -1$. So the matrix $A_{echelon}$ looks as follows now:

$$A_{echelon} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 23 & 3 & 31 & 5 & 7 & 13 & 43 & 47 & 17 & 11 & 59 & LS \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \end{array}$$

The last element in each row represents the required Legendre symbol for each unique factor of the sequence. In other words, all the leading entries of each row, the "deciding factor" of each row is assigned with the Legendre Symbol from the last element of the row. All the remaining elements called "Unutilized factors" receive Legendre symbol 1. To explain further, let us examine each row from the $A_{echelon}$ displayed above. The first element has one factor 23, the last column has Legendre Symbol 1, this implies 23 needs symbol 1. Similarly, the next row has only 3 as factor and 3 needs symbol 1. Let LS denote Legendre symbol, we repeat the same observations across all the rows and we get the following results:

  - The third row has one deciding factor 31, 31 needs LS 1

- Fourth row has 5 and 11. Now due to row operations, 11 is not deciding factor so, 11 can be assigned LS 1, based on the last column 5 gets -1.
- Fifth row has 7 as leading entry, it receives LS -1. 11 has already been assigned 1.
- Sixth row has 13 as the main factor, which receives 1 as LS based on the last column value.
- Seventh row has 43 which gets LS -1
- In Eighth row 47 gets -1
- Ninth row has 17 as deciding factor which gets LS -1
- Tenth row has 59 which receives -1

Let us group all the factors into one list called A. This A has all the deciding factors in the leading entries and unutilized factors are added towards the end of the list. So our A can be represented as:

$$\{23, 3, 31, 5, 7, 13, 43, 47, 17, 59, 11\}.$$

Applying theorem 3.2.1 from chapter 3, we need to find some list of elements B such that for some $b_i \in B$ and some $a_i \in A$, for $i \in$ row number we have

$$\left(\frac{b_i}{a_i}\right) \equiv LS_i.$$

We computed B =

$$\{8, 1, 19, 3, 5, 12, 26, 23, 12, 39, 1\}.$$

Step 7 This is the final step where we use the B generated and apply theorem 3.2.1 and theorem 3.2.2 to find p. We first carry on CRT on the list of A, B. Let p' be some integer

satisfying the following system of congruences:

$$\begin{aligned} p' &\equiv 8 \bmod 23 \\ &\equiv 1 \bmod 3 \\ &\equiv 19 \bmod 31 \\ &\equiv 3 \bmod 5 \\ &\equiv 5 \bmod 7 \\ &\equiv 12 \bmod 13 \\ &\equiv 26 \bmod 43 \\ &\equiv 23 \bmod 47 \\ &\equiv 12 \bmod 17 \\ &\equiv 39 \bmod 59 \\ &\equiv 1 \bmod 11. \end{aligned}$$

Applying the CRT we find $p' = 3693863479318$. We use this p' and add it to a multiple of the product of all the elements in list A. Let

$$\begin{aligned} a &= \prod A' \\ &= \prod\{23, 3, 31, 5, 7, 13, 43, 47, 17, 59, 11\} \\ &= 21701118223785. \end{aligned}$$

Finally we search for some prime p of the form

$$\begin{aligned} p &= ai + p' \\ &= 21701118223785i + 3693863479318 \end{aligned}$$

and repeat it until we find some prime p such that $p \equiv 1 \pmod 4$. We find such a value for $i = 27$. So the final value of p we obtained for this particular example is:

$$\begin{aligned} p &= ai + p' \\ &= 21701118223785 \cdot 27 + 3693863479318 \\ &= 589624055521513. \end{aligned}$$

Of course if p needs to be larger (e.g. 2048-bits), a larger i can be randomly chosen. For example setting

i = 1 017 937 320 413 204 122 451 734 640 586 856 516 712 923 890 121 301 671 467 352 739 768 901 601 034 734 630 832 751 118 559 864 741 589 479 211 939 668 116 999 587 629 431 639 985 506 785 452 222 216 485 698 036 930 569 180 641 349 520 170 764 490 817 745 302 813 311 839 204 222 745 193 307 798 612 603 125 325 657 959 543 903 988 326 984 707 125 038 762 006 839 792 627 761 852 376 498 932 551 558 509 485 925 700 120 315 017 613 219 283 710 374 581 138 546 006 967 575 971 919 007 701 242 921 013 861 469 939 857 529 358 272 079 976 636 646 751 318 293 979 285 823 770 237 152 011 343 977 002 541 605 358 005 041 753 039 635 320 592 260 657 587 613 348 098 464 773 927 971 867 002 193 905 515 488 588 223 518 509 656 791 111 748 877 005 997 426 934 324 370 066 507 793 919 208 324 507 021 187 899 889

yields the 2048-bit prime

22 090 378 134 689 854 668 080 427 282 924 399 061 985 819 057 143 074 636 905 749 502 669 759 243 627 343 109 946 059 649 406 823 888 192 127 133 451 352 131 764 486 660 389 009 911 360 595 055 993 664 603 227 128 050 574 790 768 665 706 293 542 736 860 338 887 946 544 783 444 853 828 040 892 381 096 722 422 382 282 208 703 853 107 887 071 741 270 179 612 861 146 502 611 233 027 524 669 896 250 225 919 308 714 002 851 453 195 821 246 252 248 594 014 614 336 885 721 706 509 818 061 443 476 514 009 134 473 905 290 198 283 343 891 119 346 531 846 306 946 446 935 635 519 599 396 204 634 046 380 053 571 297 272 810 718 798 253 473 115 268 635 805 443 863 703 912 782 069 180 419 966 100 459 558 431 826 475 599 268 846 253 100 958 565 217 359 310 385 285 591 586 732 639 093 868 978 919 193 526 919 403 603 942 139 183.

Finally we can check that p implements the intended function. Recall $\alpha = 4, \beta = 23$ and

the arithmetic sequence generated by it (see table 9.1). We confirm that indeed:

$$\begin{aligned}
\left(\frac{23}{p}\right) &= 1 \\
\left(\frac{27}{p}\right) &= 1 \\
\left(\frac{31}{p}\right) &= 1 \\
\left(\frac{35}{p}\right) &= 1 \\
\left(\frac{39}{p}\right) &= 1 \\
\left(\frac{43}{p}\right) &= -1 \\
\left(\frac{47}{p}\right) &= -1 \\
\left(\frac{51}{p}\right) &= -1 \\
\left(\frac{55}{p}\right) &= -1 \\
\left(\frac{59}{p}\right) &= -1.
\end{aligned}$$

9.2 Analysis of Results

We generated prime p for various sequence sizes. These sequence sizes are equal to the co-domain size of the functions to be evaluated. We carry out the comparison of our results with [20] and analyze out results accordingly. The maximum bit-size achieved for the d-approximation of threshold functions in [20] was 26. The table 9.2 gives us time values through important steps while implementing algorithm 2 for a function of sequence size 512.

Function size (domain cardinality)	512	256	100	50
Converting Congruences to Matrix Space	0.568	0.1756	0.202	0.0675
Test for consistency	1.842	0.653	0.205	0.0401
Finding b_i	0.1306	0.034	0.2347	0.078
CRT	130	20	0.9853	0.0653

Table 9.2: Run time for various steps in the the key generation in seconds

For evaluating a 512-element (2^9-bit) function, we need a sequence with at least 512 number of elements having unique factors. The α, β of the sequence will be chosen from the pre-calculated values generated from running section 9.1.1 giving us the value of $\alpha = 1938, \beta = 31$. These values may vary each time we run the algorithm. The same applies for the rest of the results as well. As we can observe from the results of various sequence sizes, the maximum time among all the steps is taken by the implementation of Chinese Remainder Theorem. CRT is an exponential time algorithm that has time complexity of $O(S_1 + S_2)^2$, where s_i denotes the number of digits in the modulus we are trying to solve, which is a product of all the moduli present in the system of congruences. So the larger the number of congruences to be solved, the greater is its time and computational complexity. Similarly, we produced results for various sequence sizes and the time taken through various steps of implementation can be observed in the following plot. Note that the largest sequence of 512 took under 2 minutes to calculate. The search based approach introduced in [20] takes more than 20 minutes to produce a sequence of size 26.

Performance Indicator	Residue HE introduced in [20]	Our Protocol
Highest sequence size	26	512
Evaluation Function Domain	$\{0, 0, 0, .., 1, 1, 1\}$	Any function of the form $\{0, 1\}^\ell$
Finding prime for sequence of 20	$>$ 20 minutes	0.003 seconds

Table 9.3: Comparison between protocol introduced in [20] and CS

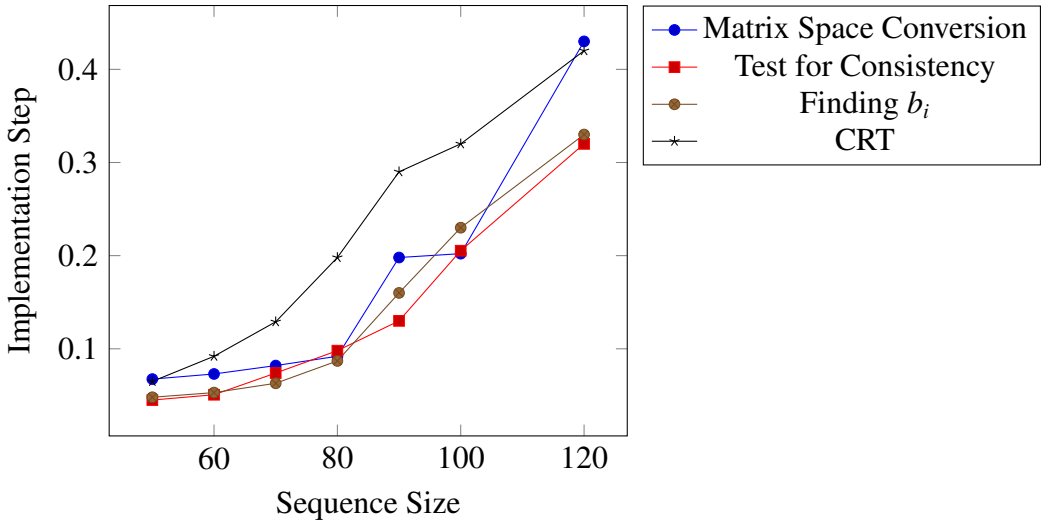


Figure 9.1: Run Time for various steps in p generation algorithm

9.3 Performance Evaluation of SPQ

This section describes the implementation process of SPQ. It is divided into two sections- the first sections contains the description of genomic data and the second section discusses the actual implementation of the SPQ protocol. The implementation was carried out in Python 3.7 environment. In the given time frame we were able to carry out the implementation only using Okamoto-Uchiyama cryptosystem. Nonetheless, the implementation using Paillier cryptosystem is likely to yield the same results.

9.3.1 Data set Description and Reading Genomic Data

The data set we used for the development of SPQ application was derived from the human genome project. All the data were de-identified by removing the information related to the humans that carry the genome, just leaving the information on the locus of the gene and the genotype present at the locus. The data is present in Variant Call Format, a file format specifically used to store information about a particular position on genome. Dealing with VCF files is difficult, so we try to extract the data from these files into readable format by converting it to either an Excel or csv. The sample data we deal with looks something like shown in fig. 9.2 after transformation. The first two columns are patient related information, which are

	Unnamed: 0	CHROM	POS	ID	REF	ALT_1	Genotype
0	0	22	16050075	rs587697622	A	G	0
1	1	22	16050115	rs587755077	G	A	0
2	2	22	16050213	rs587654921	C	T	0
3	3	22	16050319	rs587712275	C	T	0
4	4	22	16050527	rs587769434	C	A	0

Figure 9.2: Head shot of HG00096 genotype data

de-identified. Chromosome and Position locate to the exact position of the genotype. RSID is a unique ID given to Single Nucleotide Polymorphisms, the ref column indicates the reference gene at the location and Alt is the alternative polymorphism that could be found at the location, genotype column refers to the actual genotype found in the individual. As discussed in the chapter 8, the value 0 for the column named "Genotype" indicates a homozygous dominant genotype. So if we take the first row from fig. 9.2, the genotype at the given location would be *AA*. Similarly, 1 indicates a heterozygous genotype, meaning at the given locus we could find *AG* genotype. Finally the value 2 for the genotype column indicates a homozygous recessive genotype, meaning two copies of alternate alleles can be found in this case we may find the genotype *GG*. These numbers were assigned based on a library named "Scikit-Allele" [40] from python, so the notations may vary depending on the library/programming language used.

9.3.2 SPQ Implementation

To implement the SPQ protocol, we integrated all the functionalities developed from the previous algorithms. The SPQ protocol's output will be evaluated such that, let T be the threshold value,

$$f(ED) = \begin{cases} 0 & \text{if } ED \leq T \\ 1 & \text{if } ED > T. \end{cases}$$

Depending on the threshold value and the size of the sequence we implement the key generation algorithm. For example, if the threshold value is 30, then $ED \leq 30$ implies that patients are similar. Let the length of the sequence be 100, then we generate a binary sequence of the form

:

$$[0] * 30 + [1] * 70.$$

We use section 4.1 and algorithm 2 to generate p, which will be the secret key of Okamoto-Uchiyama cryptosystem. At first the encryption function will be applied directly by the *querier* by importing the cryptosystem. Here all the public and private keys are generated before even running the application. The crucial step in SPQ protocol is implementation of protocol for secure computation of Euclidean distance. For implementing this, we relied on pandas data frames in python. The *querier*'s input will be read iteratively across the entire string and three encryptions for each string are generated in each iteration. This is sent to the hospital, which then performs addition homomorphically. Then we multiply it with $alpha$ and add β homomorphically. The entire result is raised to a random $r_c^2 \mid r_c \overset{R}{\leftarrow} \mathbb{Z}_{2^k}$, which is homomorphic multiplication that can be evaluated upon decryption.

The protocol implementation has three algorithms:

- section 9.3.2 is the algorithm for User encryption for implementing the protocol for secure computation of Euclidean distance
- In the algorithm presented in section 9.3.2, Hospital chooses the relevant cipher text based on the indices and performs homomorphic addition, converting the cipher text c to $\alpha c + \beta$ and then re-randomizing it by multiplying with r_c^2.
- The algorithm displayed in section 9.3.2 consists of User Decryption and applying evaluation function to find if Euclidean distance is $\leq$ threshold value.

User SPQ

In this step the user will contain a plain text $query \in \{0, 1, 2\}^n$. The user will input the query into algorithm in section 9.3.2. The algorithm will scan across each value in the user's query and will encrypt the related bit from table 8.1 , that way for each bit 3 different encryptions are produced based on the table 8.1. All of these are stored in an array named res. Now the user sends this res array, along with files containing the public key modulus n, g values the hospital in separate files as comma separated values.

Input: Pre-computed Euclidean Distance Values in Plain text, input query sequence,pk, OkamotoUchiyama.enc

Output: For each bit in query sequence, encrypted Euclidean distance combinations- Encr

```
Lookup ← [[0, 1, 4], [1, 0, 1], [4, 1, 0]]
function SPQENCRYPT(query, pk)
    c ← OkamotoUchiyama.enc(pk, query) return c
end function
function SPQUSER(query)
    for n ∈ query do
        df ← int(n)
    end for
    res ← [[ ]]
    for i ∈ df do
        for x ∈ Lookup[i] do
            data ← spqencrypt(x, pk)
            res.append(data)
        end for
    end for
    return res
end function
```

Algorithm 3: User SPQ

Hospital SPQ

Presented in section 9.3.2, this algorithm the hospital receives three files related to res, n, g and the $RSID$ of particular gene locations. Using these files the hospital will now calculate the Euclidean distance as per the Secure ED protocol. In the first step, the hospital will form an empty array to store the Encrypted Euclidean Distance values denoted as EED in section 9.3.2. Now we read across each file in the database held by the hospital. In every single patient file,

we compile the necessary sequence based on the $RSIDs$ and these sequences will be used for comparison. For each patient, we pick one sequence. In the beginning of search through every patient's sequence we form an empty array titled *arr*. Now we iterate through every single sequence and using the sequence value and the iteration step, we form the column and row respectively of the encrypted results. For example if we are in second row and the sequence value with hospital is 2, we pick the value at location $res[2][2]$ in res array. That way for each sequence, we form a complete array filled with differences of squares. Now we homomorphically sum this array for obtaining the Euclidean distance by taking the prod value of entire array. We take the encrypted sum for each individual record and blind it by homomorphically multiplying α and adding β. To do this, we raise the *EncryptedSum* to **plain text** $\alpha \pmod{n}$ using the *pow* function in python. This is the fastest implementation of exponentiation $\pmod{n}$. Where as, to homomorphically add β, we need to encrypt β first and then perform multiplication of cipher texts. We then re-randomizing using some r^2, where $r \in \mathbb{Z}_n$. This way, EEDs of all the patients are sent back to the user.

Input: RSIDs, res - the Encrypted Euclidean distance combinations that were outputted from section 9.3.2, n, g, α, β, OkamotoUchiyama.enc, hospital database

Output: Encrypted Euclidean Distance

$Encr \leftarrow res$

function SPQHOSP(hospdata)

$EED \leftarrow [\ ]$

for $seq \in hospdata$ **do**

$df \leftarrow [int(i)$ for $i \in seq]$

$arr \leftarrow [[\ ]]$

for $j \in len(df)$ **do**

$data \leftarrow encr[j][df[j]]$

$arr.append(data)$

end for

$pk \leftarrow n, g$

$EncryptedSum \leftarrow math.prod(arr)$

$ct = (EncryptedSum)^{(\alpha)} \cdot (enc(\beta)) \pmod{n}$

$r \xleftarrow{R} \{1, 2, 3, ..., n\}$

$c' \leftarrow ct^{r^2} \pmod{n}$

$EED.append(c')$

end for

return EED

end function

Algorithm 4: Hospital SPQ

User Decryption and Evaluation

As presented in section 9.3.2 this is the final step in the protocol where the user receives a file full of re-randomized Encrypted Euclidean Distance values that were calculated between the user *query* and all the patients in the hospital database. User applies Dec and retrieves the plain

text message that is of the form $m' = ((\alpha ED + \beta) \cdot r^2)$. Now the user applies *kronecker* symbol to m', which returns the Legendre symbol of m' based on the threshold value of Euclidean Distance.

Input: c' from section 9.3.2, sk, g, OkamotoUchiyama.dec()

Output: Similarity of patient

$ct \leftarrow c'$

function SPQDEC(ct,sk,g)

$pt \leftarrow OkamotoUchiyama.dec(sk, g, ct)$

return pt

end function

function EVAL(pt,p)

$result \leftarrow kronecker(pt, p)$

if $result == 0$ **then**

return "Patients are Similar"

else

return "Check for next patient"

end if

end function

Algorithm 5: User Decryption and Evaluation

9.4 SPQ Results Analysis

Upon implementing the three algorithms described in the previous section, we obtained the results pertaining to the time taken to implement various protocols with respect to different threshold values. Also, we included the accuracy scores and precision scores by comparing the results from Secure Function Evaluation with Plain Text calculations. We conducted SPQ across various database sizes ranging from 60,000 records to a few 100 records. The results are displayed fro 60,000, 1000 and 100 records.

Protocol Name	60,000 records	1000 records	100 records
User SPQ	5.45	5.45	5.45
Hospital SPQ	530	0.653	0.205
User Decryption and Evaluation	0.1306	0.034	0.02347

Table 9.4: Results for various steps in the SPQ protocol in seconds

User SPQ takes the same time in all three cases as the input query remains the same irrespective of the number of records we are trying to compare. For implementation, we considered genotype sequences of the size 10, i.e., the query and the hospital database belongs to message space $M \xleftarrow{R} \{0,1,2\}^{10}$. Amongst all the steps, Hospital SPQ happens to be the critical as the Euclidean distance has to be calculated homomorphically across every single record. The bigger the size of hospital database, the longer will be time consumed in this aspect. User decryption and Evaluation process take the least amount of time. The reason is that decryption is implemented using Extended Euclidean Algorithm and evaluation just carries out symbol verification.

To test for the efficiency of our matching algorithm, we want to compare it with a Bloom Filter implementation of Similarity Matching using Euclidean Distance. Bloom Filter Encodings are approximate matching methods introduced by [59] that employ Bloom Filter Data Structures. They are ℓ-bit vectors with k secret hash functions. An element $e \in \{0,1\}^n$ is inserted into the filter by pointing to the outputs of the secret hash-functions. We used precision scores in similarity matching for carrying out the comparison. We changed threshold values for Euclidean Distance Comparison each time and carried on the Similar Patient Query. Precision Scores are in the scale of 0-1 and high precision scores indicate that the false positive rate is very low. To proceed further with Bloom Filters, we need a special protocol to compute Euclidean Distance which is described in the next section.

Protocol for Euclidean Distance Computations with Bloom Filters

This segment sketches a secure protocol to compute Euclidean Distance using Bloom filters. Bloom filters are ultimately a data structure for set membership testing, and thus can only

be applied toward calculating Euclidean distance with some modification. Recall that, when using Genotype data we are dealing with the character/message space of $\{0, 1, 2\}^n$. So we cannot directly apply Euclidean distance computations in the conventional sense as there will be repetitions of these limited characters across the bloom filters, resulting in poor similarity matching rates and accuracy scores. To avoid this, we propose a special protocol where instead of comparing the characters in the sequence, we compare the RSID by forming a bloom filter based on the genotype character. Recall that before forming the genotype sequences, the user publicly declares the RSID positions that they want to compare. We gather all these RSID numbers and pick the related genotype. For this protocol, we pick the RSID numbers instead. The following are the detailed steps to carry on the protocol:

- We form 3 Bloom Filters based on the three genotype characters and label them as BF0, BF1 and BF2.

- We take each RSID and insert it into the Bloom filter corresponding to the genotype at that location. For example $rs587697622$ has a genotype 0, we apply BF0 to it. We repeat the same process across all RSID positions.

- Consider two genotype sequences, Seq_1 and Seq_2, each with three associated Bloom filters $BF0, BF1, BF2$. Initialize a counter $b \leftarrow 0$. We can (approximately) compute their Euclidean distance between Seq_1 and Seq_2 as follows:

  - Compare $BF0$ of Seq_1 with $BF1$ of Seq_2. For every for each bit position containing a 1 in both Bloom filters, increment b by 1.
  - Compare $BF0$ of Seq_1 with $BF2$ of Seq_2. For every for each bit position containing a 1 in both Bloom filters, increment b by 4.
  - Compare $BF1$ of Seq_1 with $BF2$ of Seq_2. For every for each bit position containing a 1 in both Bloom filters, increment b by 1.
  - Compare $BF1$ of Seq_1 with $BF0$ of Seq_2. For every for each bit position containing a 1 in both Bloom filters, increment b by 4.

  - Compare $BF2$ of Seq_1 with $BF0$ of Seq_2. For every for each bit position containing a 1 in both Bloom filters, increment b by 1.
  - Compare $BF1$ of Seq_1 with $BF2$ of Seq_2. For every for each bit position containing a 1 in both Bloom filters, increment b by 4.

- If $b < t$ for threshold $0 < t$, output 'match.' Otherwise output 'non-match'.

Matching Accuracy Comparison Experiment

Implementing and analyzing the heuristic security of a novel Bloom filter construction is outside of the scope of this thesis, which we will leave to future work. In this interest of a more direct-comparison between our system and Bloom filter constructions previously established in literature, instead of a Euclidean distance measure of sequence similarity/dissimilarity, we return to the set-similarity approach of [20], and compare the matching accuracy of our scheme against a Bloom filter approach based on the Dice coefficient.

Without loss of generality, consider an $\ell = 1000$ bit bloom filter, with $k = 20$ hash functions. For $1 \leq k \leq 20$ let $h_k : \{0,1\}^* \rightarrow \{0,\ldots,\ell-1\}$ be a set of hash functions. Consider a genotype sequence of n-characters. Since the genotype alphabet consists of three characters $\{0,1,2\}$, if we use bi-grams for hashing, the set of all possible bi-grams has a cardinality of only 9, meaning two random genotype strings of a relatively short length would be likely to share all bigrams in common (and therefore look like identical strings in a Bloom filter), even if the two strings are trivially quite quite dissimilar by any other reasonable metric.

Our approach, therefore, is to use a larger window size. We chose to use 6-gram sequences, because the set cardinality to $3^6 = 729$ is comparable to the standard use in literature involving bi-grams of alphabetic characters (i.e., $26^2 = 676$). Additionally we used padding characters. For example, the genotype sequence '2222222222' has one nominal 6-gram ('222222'). However with padding, the 6-gram set would look like: ['_____2', '____22', '___222', '__2222', '_22222', '222222', '222222', '222222', '222222', '222222','22222_', '2222__', '222___', '22____', '2_____'], where the "_" indicates a padding character. We denote a 6-grams as s_{6g}

Next, we define the following experiment:

- **Step 1** For a given genotype sequence s, declare a bloom filter bf of 1000 bits with 0s.

So $bf = \{0, 0, 0, ..., 0, 0\}$.

- **Step 2** Decompose the genotype sequence s into a set of 6-grams, S.
- **Step 3** For each 6-gram $s_{6g} \in S$, insert into the Bloom filter as follows. For each $1 \leq k \leq 20$ hash function h_k, compute $j = (h_k(s_{6g}))\pmod{1000}$. Set $bf[j] = 1$.
- **Step 4** Repeat steps $1 - 3$ to construct a bloom filter for all the genotypes sequences in the hospital database.
- **Step 5** Given a *query* bloom filter , compute the Dice coefficient between the query and each record in the database. The Dice coefficient between two sets a, b is computed as:

$$Dice(a, b) = \frac{2 \cdot (a \cap b)}{|a| + |b|}$$

where $|a|$ indicates the cardinality of set a.

- **Step 6** For threshold t, query sequence *query* and a single sequence from hospital database *record*, If $Dice(query, record) \geq t$, return 1. Other wise 0.
- **Step 7** For obtaining the truth, we take the 6-grams from plain text data and compute the Dice coefficient between plain-text query and plain-text records.
- **Step 8** We repeat steps $1-7$ for rest of the records in the hospital data base. We collect all the results using Bloom Filters and the plain-text data and compute the precision scores.
- **Step 8** We need precision scores because we want to evaluate how the false positive rate (n.b., $P[False\ Negatives = 0]$ with Bloom filters) is behaving with respect to Bloom filters. For a given query, if the ground truth returns a record as a match and the Bloom Filter also shows it as a match, then it is a True match. In case the ground truth returns that the query is not a match to the record but the Bloom Filters returns this as a match, then we see that record as a false match. It is essential to eliminate false positives, because we would not want the wrong patient data to be returned as a match as it can have adverse impact with respect to medical information. We compute the precision scores using the following formula:

$$Precision = \frac{True\ Matches}{TotalMatches} = \frac{True\ Matches}{True\ Matches + False\ Matches}$$

The value of precision scores range from $0 - 1$, a score 1 indicates there are no false positives.

We carried out the comparison of these similarity results with our SPQ protocol. Our protocol outperformed Bloom Filters in this aspect with precision scores equal to 1 for threshold values ranging from $0.50 - 0.90$ as displayed in all the results below. This comparison was implemented across 100 records in the hospital database. The precision scores for our SPQ protocol were also calculated based on the Euclidean Distance scores measured on the plain-text data set. Where as for Bloom Filters protocol, we calculated the precision scores by comparing the dice coefficient values on the plain text data. Finally, we took the average of the precision scores from 100 different runs of the SPQ protocol in Bloom Filters and our protocol. i.e., We queried 100 different patients data across a database of 100 records. In different implementations, we varied the query size and the hospital database accordingly. So for example, when we had to use a query of sequence size 20, we used the database related to sequence size of 20 and vice versa.

The experiments were conducted across different genotype sequence sizes of 10, 12, 15, 20, 25. The graphs are presented in fig. 9.3 indicate that our protocol implements ideal functionality. The precision score of 1 means there are no false positives or false negatives and the transaction carried out to calculate Euclidean Distance is exactly the same the one carried out under plain text. Note that to generate the prime numbers, the maximum sequence size need to be disclosed. For example, in the 10-character sequence, the maximum Euclidean Distance would be $ED(2222222222, 0000000000)$, which is equal to 40, where $ED(A, B) = \sum(a - b)^2$ for $a \in A, b \in B$. So the prime number generated needs to have the linear embedding in the size of $g(i) \xleftarrow{R} \{0, 1\}^{40}$. Based on the threshold value, we decide on the co-domain order within this size of 40. So each time we change the threshold value, we had to generate a new prime depending on the Legendre Symbol sequences.

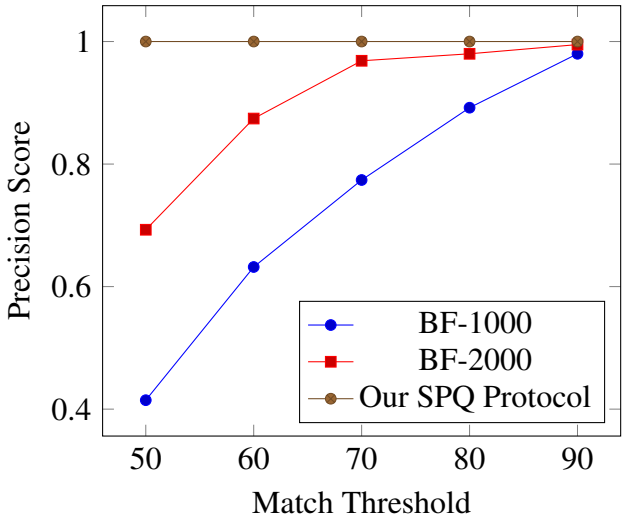
Coming to bloom filters, we used the same threshold values as that of our protocol for dice-coefficient calculations. Since the scope of this thesis is limited to our protocol alone, the parameters from Bloom Filters were chosen based on the industrial standards. Accordingly, we applied two different bloom filters:

1. Bloom Filter of 1000 bits with 20 hash functions

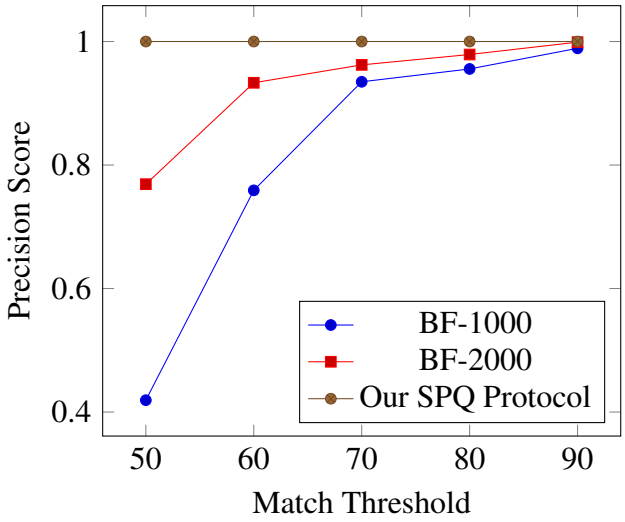
2. Bloom Filter of 2000 bits with 30 hash functions

The results for all the 100 experiments are displayed in the figure shown in fig. 9.3. These results indicate that the Bloom Filters showed improved performance only with increased bit sizes and increased number of hash function. Even though Bloom Filter Encodings are known for their speed in search algorithms, they have a problem of returning large number of false positives. The precision scores improve in the bloom filters only when the threshold values keep increasing. However, one should observe that, in case of threshold function at 50% the in both classes, i.e., above or below threshold has similar distribution. As it keeps increasing there is skewed distribution. So the change in precision do not reflect the improvement in performance of bloom filter rather they reflect the fact that the number of hits are easier to capture for precision at skewed distribution of data. Whereas, our protocol implements ideal functionality across all sequence sizes and entire range of the threshold values.

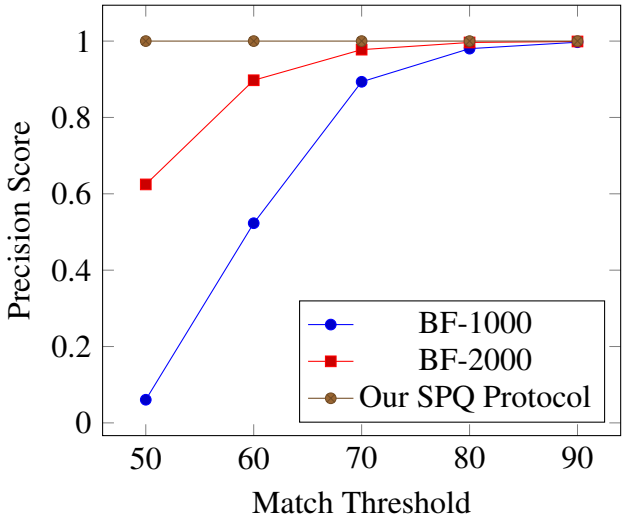
(a) Results for SPQ on genomic sequence size 10



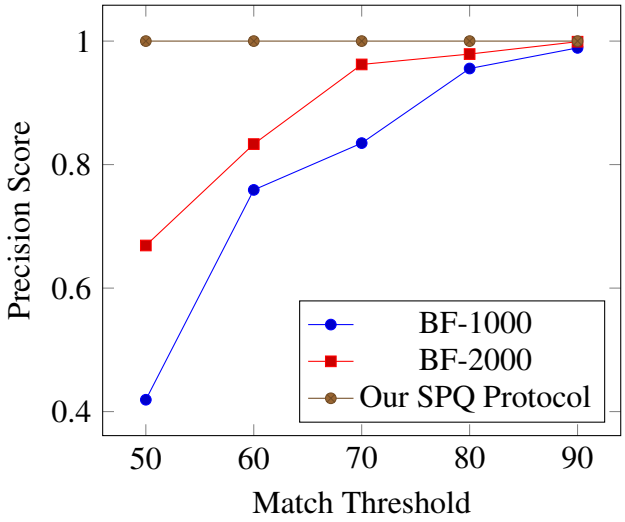
(b) Results for SPQ on Genomic sequence size 12



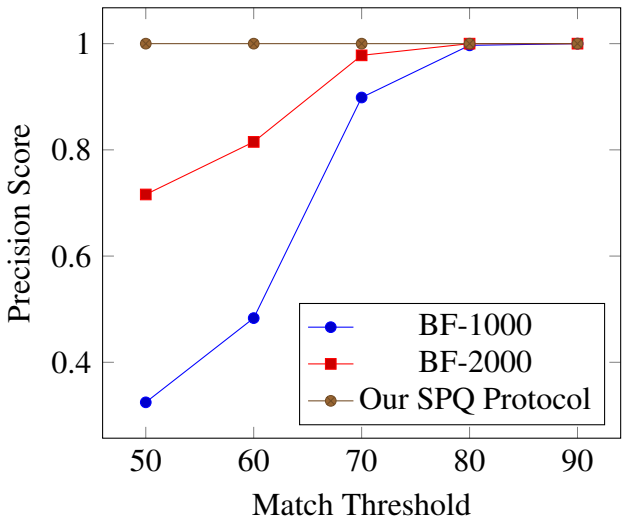
(c) Results for SPQ on genomic sequence size 15



(d) Results for SPQ on Genomic sequence size 18



(e) Results for SPQ on genomic sequence size 20



(f) Results for SPQ on genomic sequence size 25

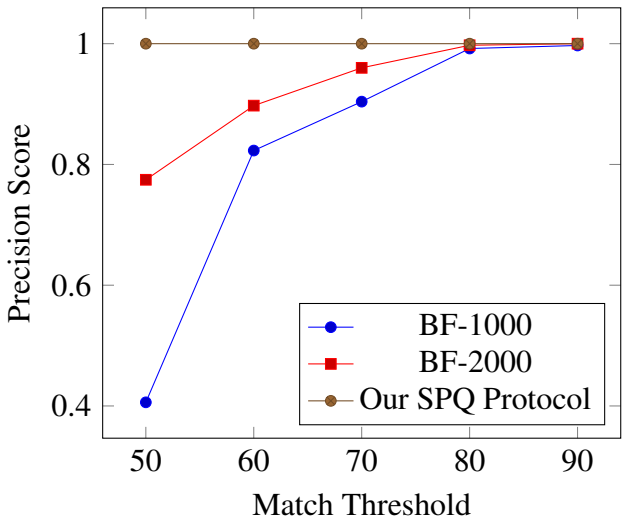


Figure 9.3: Results across experiments on various genotype sequence sizes

Chapter 10

Conclusion and Future Work

This chapter summarizes our thesis and throws a light on the contributions that can be used in this field. It also examines the limitations of our current research work by highlighting certain complexities that exist in the implementation of the cryptosystem. This way, the potential areas for future work can be identified to optimize this cryptosystem further. We stated that the cryptosystem can be used for private function evaluation for the functions of the form $\mathbb{Z} \mapsto \mathbb{B}$ and we implemented it on threshold functions. There are still undiscovered functions of this type to which our cryptosystem can be efficiently extended. We split this chapter into three sections:

- Section 10.1 summarizes the thesis by briefly restating the key aspects of the research including the application.

- Section 10.2 presents the contributions of our cryptosystem, and highlights the limitations that we faced through the implementation process.

- Section 10.3 highlights the prospects of our cryptosystem and other applications that it can possibly extended to. It also highlights the areas where more work needs to be done to improve the time and computational complexity.

10.1 Summary

This thesis discussed the importance of secure computations and the related work that is carried out in this area. We briefly examined two probabilistic cryptosystems that exhibit additive homomorphism. Their description was done by examining the three components of any cryptosystem namely:

- Key Generation algorithm- Gen
- Encryption - Enc
- Decryption -Dec

We extended these additive homomorphisms to perform secure or private function evaluation in a unique way by introducing a new Key Generation algorithm Gen. Precisely, this Gen presents a unique way of generating prime p, which forms the secret key as it is one of the factors of n for $n = pq$. To develop this key generation algorithms, we introduced new form of functions called functional embeddings and extended this concept of functional embeddings to Linear embeddings in Residue Symbol sequences. Based on the properties of such linear embeddings in Quadratic residues and non-residues $\pmod{p}$, we developed two new theorems along with proofs. The first theorem emphasized the possibility of discovering pattern of linear embeddings in some modulus and the second theorem discusses on the possibility of such modulus being a prime number. Using this key generation, we introduced a fourth component into the cryptosystem called an evaluation function, indicated as *Eval* that helps us to perform secure or private function evaluation in a unique way upon decryption. We then discussed the security of our cryptosystem by explaining the necessary jargon surrounding these concepts. We ran through necessary theorems and proofs that help us establish the hardness of the developed cryptosystem.

We developed a new protocol based on the developed cryptosystem with four steps namely Gen, Enc, Dec, Eval. The correctness and hardness proof of this protocol were also described along with it. Then we discussed the privacy of the participants involved in the protocol. Since we developed two party computation, the privacy is discussed for both of these parties separately by introducing the concept of view and developing a simulation based privacy proof.

Then we extended this cryptosystem to an application known as Similar Patient Query. We discussed the basics of SPQ, the related work done in this field, and a brief description of the similarity measures used in genomic data. We developed a new protocol for secure computation of Euclidean distance and used it in the development of SPQ . We highlighted the advantages of our protocol over the existing systems and also developed a hardness proof alongside with it. The final chapter threw light on the implementation of the cryptosystem and the application. We presented the algorithms required for finding α, β required for the generation of prime p and key generation algorithm itself. We presented the analysis of results in comparison with previous work in [20]. Implementation of SPQ is also presented by describing the details of genomic databases, implementing user-spq, hospital-spq, user-decryption and evaluation. We carried out analysis of results by comparing our protocol to similarity matching using Bloom Filter Encodings. We also introduced a new methodology to compute accurate Euclidean distance among genomic data bases using Bloom Filter Encodings. We finally presented the precision scores for both the protocols.

10.2 Contributions and Scope for Future Work

The main contributions of our thesis are development of a unique way for secure function evaluation using the properties from number theory and SPQ application using Euclidean Distance. We modified the existing probabilistic cryptosystems that display additive homomorphisms in a way to be able to implement private function evaluation using the possibility of finding arbitrary patterns in the runs of Quadratic and Non-Quadratic Residues modulo p. Even though this property has been described in previous work by [20], the paper used a search based approach. In search based approach, the prime number with the required Residue and Non-residue sequences is iteratively searched. Due to this, the application was limited to threshold function alone and also the implementation was time consuming. So, we optimized it by generating the required prime number instead. While cutting down the time required for finding such primes drastically, we were also able to extend to the functionality to any kind of domain that has a message space of $\{0,1\}^n$ where n is the size of the required co-domain.

The prime generation requires finding out sequences that have elements consisting of unique

factors when factorized. In other words, if we want a function for size of 2^9 bits, i.e., 512 characters in the co-domain, we need to find a sequence that has 512 elements with each element consisting at-least one unique factor so we can manipulate its Legendre symbol. To find the start of such sequence and its relevant step size, we iteratively generate multiple sequences, factorize each element in this sequence and iterate it until a unique factor is no longer found. More work can be conducted in this area to develop more optimized algorithms for generation of such sequences. This can also be extended to find primes of smaller bit lengths. With respect to practical use for this cryptosystem, we discussed only SPQ application. We believe that the versatility of this secure function evaluation has not been fully realised. Future researches can be conducted in this aspect to find out applications for any functions with a co-domain space of $\{0, 1\}^n$.

10.3 Conclusion

The increase in the availability and accessibility of personal information is demanding more sophisticated ways of processing that information while ensuring privacy. The research in the area of Secure Computations is never ending owing to the increase in applications involving encrypted data. There is always a scope for improvement in devising Secure and Private Function Evaluation algorithms that offer lower computational and time complexities. While our protocol offered a solution to carry out function evaluation for Linearly embedded domains, continued research in this aspect may improve the scope for evaluating more complex functions using simpler homomorphic encryption schemes and other cryptographic algorithms.

Bibliography

[1] Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[2] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. Secure approximation of edit distance on genomic data. *BMC medical genomics*, 10(2):41, 2017.

[3] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. Privacy-preserving search of similar patients in genomic data. *Proceedings on Privacy Enhancing Technologies*, 2018(4):104–124, 2018.

[4] Erman Ayday, Emiliano De Cristofaro, Jean-Pierre Hubaux, and Gene Tsudik. The chills and thrills of whole genome sequencing. *Computer*, 2013.

[5] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 565–577, 2016.

[6] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

[7] Duncan A Buell and Richard H Hudson. On runs of consecutive quadratic residues and quadratic nonresidues. *BIT Numerical Mathematics*, 24(2):243–247, 1984.

[8] David A Burgess. The distribution of quadratic residues and non-residues. *Mathematika*, 4(2):106–112, 1957.

[9] Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1518–1529, 2015.

[10] Ke Cheng, Yantian Hou, and Liangmin Wang. Secure similar sequence query on outsourced genomic data. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 237–251, 2018.

[11] Joseph I Choi, Dave Tian, Grant Hernandez, Christopher Patton, Benjamin Mood, Thomas Shrimpton, Kevin RB Butler, and Patrick Traynor. A hybrid approach to secure function evaluation using sgx. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 100–113, 2019.

[12] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[13] Ivan Damgard, M. Geisler, and M. Kroigaard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1(1):22–31, 2008.

[14] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian conference on information security and privacy*, pages 416–430. Springer, 2007.

[15] Emiliano De Cristofaro. Genomic privacy and the rise of a new research community. *IEEE security & privacy*, 12(2):80–83, 2014.

[16] Cunsheng Ding. Pattern distributions of legendre sequences. *IEEE Transactions on Information Theory*, 44(4):1693–1698, 1998.

[17] Wenxiu Ding, Zheng Yan, and Robert H Deng. Encrypted data processing with homomorphic re-encryption. *Information Sciences*, 409:35–55, 2017.

[18] MB Eisen, PT Spellman, PO Brown, D Botstein, P Spellman, P Brown, PO Browndagger, D Bostein, O Brown Patrick, D Bolstein, et al. Cluster analysis and display of genome-wide expression patterns. 1999.

[19] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[20] Aleksander Essex. Secure approximate string matching for privacy-preserving record linkage. *IEEE Transactions on Information Forensics and Security*, 14(10):2623–2632, 2019.

[21] Keith Frikken, Mikhail Atallah, and Chen Zhang. Privacy-preserving credit checking. In *Proceedings of the 6th ACM conference on Electronic commerce*, pages 147–154, 2005.

[22] Josh Fruhlinger. What is information security? definition, principles, and jobs, Jan 2020.

[23] Joseph Gallian. *Contemporary abstract algebra*. Nelson Education, 2012.

[24] Carl Friedrich Gauss. *Disquisitiones arithmeticae*, volume 157. Yale University Press, 1966.

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

[26] Craig Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, 2010.

[27] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 365–377, New York, NY, USA, 1982. Association for Computing Machinery.

[28] Ben Green and Terence Tao. The primes contain arbitrarily long arithmetic progressions, 2004.

[29] Kevin Henry. The theory and applications of homomorphic cryptography. Master's thesis, University of Waterloo, 2008.

[30] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. An introduction to mathematical cryptography. 2014.

[31] Richard H Hudson. On the first occurrence of certain patterns of quadratic residues and non-residues. *Israel Journal of Mathematics*, 44(1):23–32, 1983.

[32] Jinho and Jonathan. Probabilistic cryptosystems, Aug 2018.

[33] Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 556–571. Springer, 2011.

[34] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for xor gates that beats free-xor. In *Annual Cryptology Conference*, pages 440–457. Springer, 2014.

[35] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[36] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *International Conference on Financial Cryptography and Data Security*, pages 83–97. Springer, 2008.

[37] Ximeng Liu, Robert H Deng, Kim-Kwang Raymond Choo, and Jian Weng. An efficient privacy-preserving outsourced calculation toolkit with multiple keys. *IEEE Transactions on Information Forensics and Security*, 11(11):2401–2414, 2016.

[38] Md Safiur Rahman Mahdi, Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. Secure similar patients query on encrypted genomic data. *IEEE journal of biomedical and health informatics*, 23(6):2611–2618, 2018.

[39] Bradley A Malin. An evaluation of the current state of genomic data privacy protection technology and a roadmap for the future. *Journal of the American Medical Informatics Association*, 12(1):28–34, 2005.

[40] Alistair Miles. Allel - explore and analyse genetic variation, 2015.

[41] Payman Mohassel, Saeed Sadeghian, and Nigel P Smart. Actively secure private function evaluation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 486–505. Springer, 2014.

[42] Burkhard Morgenstern, Bingyao Zhu, Sebastian Horwege, and Chris André Leimeister. Estimating evolutionary distances between genomic sequences from spaced-word matches. *Algorithms for Molecular Biology*, 10(1):5, 2015.

[43] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139, 1999.

[44] Muhammad Naveed. Hurdles for genomic data usage management. In *2014 IEEE Security and Privacy Workshops*, pages 44–48. IEEE, 2014.

[45] Muhammad Naveed, Shashank Agrawal, Manoj Prabhakaran, XiaoFeng Wang, Erman Ayday, Jean-Pierre Hubaux, and Carl Gunter. Controlled functional encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1280–1291, 2014.

[46] Muhammad Naveed, Erman Ayday, Ellen W Clayton, Jacques Fellay, Carl A Gunter, Jean-Pierre Hubaux, Bradley A Malin, and XiaoFeng Wang. Privacy in the genomic era. *ACM Computing Surveys (CSUR)*, 48(1):1–44, 2015.

[47] Tatsuaki Okamoto, Shigenori Uchiyama, and Eiichiro Fujisaki. Epoc: Efficient probabilistic public-key encryption (submission to p1363a). *IEEE P1363a*, page 18, 1998.

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

[49] Annika Paus, Ahmad-Reza Sadeghi, and Thomas Schneider. Practical secure evaluation of semi-private functions. In *International Conference on Applied Cryptography and Network Security*, pages 89–106. Springer, 2009.

[50] Rene Peralta. On the distribution of quadratic residues and nonresidues modulo a prime number. *Mathematics of Computation*, 58(197):433–440, 1992.

[51] Andreas Peter, Erik Tews, and Stefan Katzenbeisser. Efficiently outsourcing multiparty computation under multiple keys. *IEEE transactions on information forensics and security*, 8(12):2046–2058, 2013.

[52] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure two-party computation is practical. In *International conference on the theory and application of cryptology and information security*, pages 250–267. Springer, 2009.

[53] John Quackenbush. Computational analysis of microarray data. *Nature reviews genetics*, 2(6):418–427, 2001.

[54] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, page 707–721, New York, NY, USA, 2018. Association for Computing Machinery.

[55] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[56] Ahmed Salem, Pascal Berrang, Mathias Humbert, and Michael Backes. Privacy-preserving similar patient queries for combined biomedical data. *Proceedings on Privacy Enhancing Technologies*, 2019(1):47–67, 2019.

[57] Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. 2007.

[58] Thomas Schneider and Oleksandr Tkachenko. Episode: Efficient privacy-preserving similar sequence queries on outsourced genomic databases. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 315–327, 2019.

[59] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. Privacy-preserving record linkage using bloom filters. *BMC medical informatics and decision making*, 9(1):41, 2009.

[60] Kana Shimizu, Koji Nuida, Hiromi Arai, Shigeo Mitsunari, Nuttapong Attrapadung, Michiaki Hamada, Koji Tsuda, Takatsugu Hirokawa, Jun Sakuma, Goichiro Hanaoka, et al. Privacy-preserving search for chemical compound databases. *BMC bioinformatics*, 16(S18):S6, 2015.

[61] Peter Snyder. Yao's garbled circuits: Recent directions and implementations, 2014.

[62] Frank Stajano. Privacy in the era of genomics. *netWorker*, 13(4):40–ff, 2009.

[63] Frank Stajano, Lucia Bianchi, Pietro Liò, and Douwe Korff. Forensic genomics: kin privacy, driftnets and other open questions. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pages 15–22, 2008.

[64] Star Trek. How to change a matrix into its echelon form.

[65] Bo Wang, Aziz M Mezlini, Feyyaz Demir, Marc Fiume, Zhuowen Tu, Michael Brudno, Benjamin Haibe-Kains, and Anna Goldenberg. Similarity network fusion for aggregating data types on a genomic scale. *Nature methods*, 11(3):333, 2014.

[66] Wei Wang, Yin Hu, Lianmu Chen, Xinming Huang, and Berk Sunar. Exploring the feasibility of fully homomorphic encryption. *IEEE Transactions on Computers*, 64(3):698–706, 2013.

[67] 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. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 492–503, 2015.

[68] Steve Wright. Patterns of quadratic residues and nonresidues for infinitely many primes. *Journal of Number Theory*, 123(1):120–132, 2007.

[69] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[70] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[71] Jianchao Yao, Chunqi Chang, Mari L Salmi, Yeung Sam Hung, Ann Loraine, and Stanley J Roux. Genome-scale cluster analysis of replicated microarrays using shrinkage correlation coefficient. *BMC bioinformatics*, 9(1):288, 2008.

[72] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 220–250. Springer, 2015.

[73] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. Cryptographic applications of th-residuosity problem with an odd integer.

[74] Ruiyu Zhu and Yan Huang. Efficient privacy-preserving general edit distance and beyond. *IACR Cryptology ePrint Archive*, 2017:683, 2017.

Appendix A

Row Operations for Matrix A

This appendix presents row operations that were carried on to convert Matrix A from section 9.1.2 into row echelon form.

Row Operation 1

Add -1 times the second row to the fifth row:

$$\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)
\xrightarrow{r_5+(-1)r_2}
\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)$$

Row Operation 2

Add -1 times the second row to the eighth row:

$$\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}\right)
\xrightarrow{r_8+(-1)r_2}
\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}\right)$$

Row Operation 3

Add -1 times the fourth row to the ninth row:

$$\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}\right)
\xrightarrow{r_9+(-1)r_4}
\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}\right)$$

Row Operation 4

Interchange the fifth and ninth row

$$\left(\begin{array}{cccc|cccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)
\xrightarrow{r_5 \leftrightarrow r_9}
\left(\begin{array}{cccc|cccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)$$

Row Operation 5

Multiply 5th row by -1

$$\left(\begin{array}{cccc|cccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)
\xrightarrow{(-1)\times r_5}
\left(\begin{array}{cccc|cccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)$$

Row Operation 6

Interchange 6th row and 9th row

$$\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)
\xrightarrow{r_6 \leftrightarrow r_9}
\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)$$

Row Operation 7

Interchange 7th row and 9th row

$$\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)
\xrightarrow{r_7 \leftrightarrow r_9}
\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right)$$

Row Operation 8

Interchange 8th row and 9th row

$$\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}\right)
\xrightarrow{r_8 \leftrightarrow r_9}
\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}\right)$$

Row Operation 9

Add (-1) times the 5th row to the 4th row

$$\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}\right)
\xrightarrow{r_4 + (-1) r_5}
\left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}\right)$$

The matrix on the right hand side of row operation 9 represents the final matrix in Row-Reduced Echelon form. Remember, we are doing all the matrix operations in a Galois Field 2, so in the final matrix presented in chapter 9, all the values are converted into $\pmod 2$. So the rows with elements -1 will be converted into 1.

Curriculum Vitae

Name:	Mounika Pratapa
Education and Degrees:	JNTU Hyderabad 2007–11 B.Tech University of Western Ontario London, ON 2018–20 M.E.Sc.
Related Work Experience:	Teaching Assistant University of Western Ontario 2018–20 Software Engineer Cellarch Technologies 2012-2017

Publications:

Mounika Pratapa, Aleksander Essex. A Novel method for Secure Function Evaluation based on arbitrary patterns in Quadratic Residues. Manuscript in preparation.