September 2017

Secure Integer Comparisons Using the Homomorphic Properties of Prime Power Subgroups

Rhys A. Carlton
The University of Western Ontario

Supervisor
Dr. Aleksander Essex
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering

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

Abstract

Secure multi party computation allows two or more parties to jointly compute a function under encryption without leaking information about their private inputs. These secure computations are vital in many fields including law enforcement, secure voting and bioinformatics because the privacy of the information is of paramount importance.

One common reference problem for secure multi party computation is the Millionaires' problem which was first introduced by Turing Award winner Yao in his paper "Protocols for secure computation". The Millionaires' problem considers two millionaires who want to know who is richer without disclosing their actual worth. There are public-key cryptosystems that currently solve this problem, however they use bitwise decomposition and Boolean algebra on encrypted bits. This type of solution is costly as each bit requires its own encryption and decryption.

Our solution to the Millionaires' problem and secure integer comparison looks at a new approach which doesn't use the decomposition method and instead encrypts the full length of the message in one encryption (within scope). This method also extends in a linear fashion (with respect to the number of encryptions), so larger integers remain efficient to compare.

In this thesis, we present a new cryptosystem with a novel homomorphic property used for secure integer comparison, as well as a protocol implementing the cryptosystem and a simulation security proof for the protocol. Finally, we implemented the system and compared it to systems that are being used today.

Keywords: Prime Power Groups, Public Key Cryptosystem, Secure Integer Comparison, Cryptography, Homomorphic Encryption Properties

Acknowlegements

First, I would like to thank my supervisor, Dr. Aleksander Essex, for his unwavering support throughout the duration of my master's program. There were a number of setbacks through the last two years, but his genuine enthusiasm and his ability to provide a nurturing learning environment were factors that I've found invaluable. Being the first person to open my eyes to the field of cryptography, Dr. Essex has allowed me to seek out my interests in the field and has fostered the curiosities that I have had. Without his guidance and upbeat attitude, the completion of this thesis would be impossible and I may have never delved into the field in the first place.

I would like to thank my parents who have provided me with continuous support and unfailing encouragement throughout my scholastic endeavours and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

To my friends who kept me sane through my thesis. Thank you for listening, offering me advice, and supporting me through this entire process. The nights spent playing trivia, watching Rick McGhee and the general help you have provided over the past two years were all greatly appreciated.

Finally thanks to Krzysztof Kapulkin, who I have co-written a publication (in submission) with. Your help with the underlying mathematics behind our protocol was imperative and it was a pleasure working with you.

Contents

Certificate of Examination i

Abstract i

Acknowlegements ii

List of Figures v

List of Tables vi

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Contributions . . . 2
- 1.3 Organization of Thesis . . . 4

2 Background 5
- 2.1 Secure Computation . . . 5
- 2.2 Public Key Encryption . . . 6
  - 2.2.1 Cryptosystem Breakdown . . . 7
  - 2.2.2 Semantic Security . . . 9
  - 2.2.3 IND-CPA Game . . . 9
  - 2.2.4 Randomized Encryption . . . 10
  - 2.2.5 Math Basics . . . 10
- 2.3 Homomorphic Encryption . . . 11
- 2.4 DGK Public Key Cryptosystem . . . 13
  - 2.4.1 Modern Use . . . 15
- 2.5 Paillier Encryption . . . 15
- 2.6 ElGamal Encryption . . . 18

3 Our Protocol 21
- 3.1 Introduction . . . 21
- 3.2 Mathematical Principles . . . 21
- 3.3 Hardness Assumptions . . . 23
- 3.4 Working in Prime Power Groups . . . 24
- 3.5 Our Encryption Scheme . . . 25
  - 3.5.1 Unique Homomorphic Properties . . . 27
  - 3.5.2 Semantic Security . . . 28

3.6 Our Protocol . . . 29
3.6.1 Correctness . . . 30
3.7 Extension Techniques . . . 32

4 Protocol Security **35**
4.1 Simulation Security Basics . . . 35
4.2 Our Simulation Environment . . . 36
4.3 Simulation Proof . . . 37

5 Implementation **42**
5.1 Environment . . . 42
5.2 Coding Concepts . . . 43
5.3 Algorithms Used . . . 44
5.4 Analysis . . . 47

6 Extension to Geo-spatial Analysis **51**
6.1 Overview . . . 51
6.2 Math Basics . . . 52
6.2.1 Euclidean Distance . . . 52
6.3 Geo-spatial Protocol . . . 53
6.3.1 Distance Calculation . . . 53
6.3.2 Shuffling . . . 53
6.3.3 Comparison Protocol . . . 53
6.3.4 Privacy Proof . . . 55
6.3.5 Correctness . . . 56
6.4 Future Work . . . 57

7 Conclusion and Future Work **60**
7.1 Summary . . . 60
7.2 Discussion . . . 61
7.3 Future Work . . . 62
7.4 Conclusion . . . 62

Bibliography **63**

Curriculum Vitae **69**

List of Figures

3.1 Secure integer comparison protocol evaluating $(m_1 > m_2)$ 31
3.2 Secure integer comparison protocol for large ranges using blocking 34

4.1 Ideal functionality of our protocol for P_A 37
4.2 Ideal functionality of our protocol for P_B 37
4.3 Simulation of P_B 40
4.4 Simulation of P_A 41

6.1 Euclidean distance sub protocol 54
6.2 Geo-spatial comparison protocol 55
6.3 All variations described in geo-spatial correctness 58

List of Tables

5.1 Table for extracting bits using squarings 44
5.2 Comparison of DGK protocol performance vs. our performance 50

6.1 Table for variations in geo-spatial correctness 59

Chapter 1

Introduction

The internet has revolutionized the way that information is distributed on a day-to-day basis. From health informatics to trustworthy voting, law records, Global Positioning Systems, advertising and secure database linkage, personal information is more accessible than ever before. With the availability of personal information there has been tension between solving social problems and protecting individual privacy. This is where the foundation on which secure computation is based; to compute answers to questions without disclosing unnecessary information. For this problem, protocols are put in place to hide information while still allowing coordination and evaluation of the computation. An example situation using secure computation is proving that an election was counted correctly without revealing individual's votes.

The development of protocols to ensure the security of information has steadily improved over time, but with every step forward we make in the field of computation there is a need to reassess the current standards that are being used. Within the scope of this thesis, we look at an arrangement of current public key cryptosystems and create a new system that we believe can compete with the current systems in practice. To do this we look at the field of homomorphic encryption, which allows for functions to be done under encryption, and parties to work together without learning any excess information about the other party. These concepts will be explained in detail in the following chapters.

1.1 Motivation

Secure multi party computation has been a cornerstone in the field of cryptography since it was first brought in the early works of Goldwasser and Micali [31], and Micali and Rogaway [45]. Fischlin [25] used these works to create the first definition of secure computation. Broadly speaking,this process allows a two or more parties to compute a

function involving the private inputs of the respective parties. This function is calculated in a way that reveals no information about another party's input beyond what can be determined from the output of the function itself. For an adversary to learn anything more than the inputs of the corrupted parties and the final output of the functions would require every party to be corrupted.

Secure computation is essential for many efficient secure protocols. One example is its use in secure online voting. It ensures the discretionary nature of each individual vote while also allowing a overall tally for the election to be computed. This acts as a safeguard, affirming these results are not compromised. Without secure computation the ability to perform the aggregation while ensuring the integrity and privacy of each vote would not be feasible. The is just one case amongst many where secure computation enables the integrity of the procedure, and one instance where the protocol within this thesis could be implemented.

Within the set of secure multi party protocols, one of the largest growing areas is the use of data analysis and aggregation. Over the last few years, the amount of data that has been uploaded to the internet has increased drastically. This abundance of information stretches from medical data to web access logs and its size transcends protocols used previously. The distribution of such large amounts of information has forced developers to create distributed protocols that are more efficient when dealing with large sets of data. The boom in this area sparked an interest in the creation and implementation of a secure protocol to use when dealing with comparing information using multi party computation. In particular, within this thesis we are looking at secure integer comparison which entails comparing secret numbers from two parties and securely returning the result of which integer was larger.

One example of this type of application is an application using geo-spatial analysis to identify and order the distances of different locations from a user's current location. This specific example was looked at in depth and the options for secure comparison for the ordering of locations was lacklustre. Secure integer comparisons currently require integers to be broken into their bitwise composition and evaluated using a large number of encryptions and decryptions (one for each bit in the integer). The absence of a general bulk comparison for integers left me unsatisfied, and it was from here that I began looking into the creation of a new cryptosystem that allows for better secure integer comparison.

1.2 Contributions

This thesis makes three distinct contributions to the field of public key cryptography;

1. The creation of a novel public-key cryptosystem exploiting a homomorphic property that can be used for efficient secure integer comparison,

2. The creation of an efficient protocol for the secure comparison of integers based on this new cryptosystem,

3. A simulation based security proof for the protocol,

4. An implementation of both the cryptosystem and the protocol, which allows for comparison with other public key cryptosystems.

These four contributions will be the main focus for the majority of the thesis, with the final chapters discussing future applications for the contributions and fields that we have looked at implementing the new protocol in.

Initially there will be discussion about the encryption scheme of the cryptosystem that was developed throughout this masters degree. The research and use of prime power subgroups allow for addition under encryption that exploits the new threshold function. This allows for the comparison of two encrypted values (m_1, m_2) together to see if they fit a certain mathematical criterion. The threshold discussed in this paper is denoted by t and the function discussed is:

$$f(m_1, m_2) = \begin{cases} 0 & m_1 + m_2 \geq t \\ m_1 + m_2 & \text{otherwise.} \end{cases}$$

Typically comparisons are done by splitting integers into individual bitwise encryptions and computing secure Boolean operations on those encrypted bits. The approach to our integer comparison differs from the standard approach as we are able to compare two encrypted integers through the (nearly) direct application of the homomorphism on a single encrypted value. Our approach avoids the need for binary decomposition which allows us to reduce the number of encryptions and decryptions that are needed to compare these integers. This approach can also expands in a linear fashion, so the comparison for larger values remains efficient.

Finally we will talk about the creation of our protocol, which utilizes this new cryptosystem in a way designed to compute secure integer comparisons using the new homomorphism that was stated above. The structure of this protocol differs from most common protocols because the message space and all of the calculations exist within an exponent. The mathematics related to prime power subgroups is a relatively unexplored direction and this protocol could be one of the first to properly harness the benefits of

such a structure. This also differs from most homomorphic encryption schemes because the groups used for our message space are not large primes or of semi-prime order.

1.3 Organization of Thesis

The remainder of this thesis is organized in the following manner:

Chapter 2 contains an introduction to the principles and mathematics behind public key cryptography. It examines the core concepts of secure computation and looks into three modern public key cryptosystems: DGK, Paillier and ElGamal. This section also introduces the concept of homomorphic encryption and the partially homomorphic properties that each of these cryptosystems posses.

Chapter 3 examines the creation of a new public key cryptosystem (our cryptosystem) that has a unique homomorphic property allowing for efficient secure integer comparison. It explains the mathematical foundation of the cryptosystem and a protocol that was developed to demonstrate the new homomorphic property that was discovered.

Chapter 4 looks at the security behind our protocol and introduces the concept of simulation security. It details the environment that our protocol would be implemented in and talks about the problems this environment causes. This chapter proves that our protocol is secure against passive adversaries.

Chapter 5 discusses the implementation of the protocol. It describes the Sage working environment that was used for the implementation and looks at the performance of the protocol in comparison to other commonly used public key cryptosystems.

Chapter 6 takes a look at the field of Geo-spatial analysis and the possibility of using our cryptosystem within that field. This thesis originated as creating a protocol for geo-spatial analysis but was pivoted when the lack of an efficient secure integer comparison method was noticed.

Chapter 7 talks about the implications of the work and what considerations should be made when using or expanding upon our protocol. It also gives a synopsis of the work that was presented throughout the thesis and lists the contributions made to the field.

Chapter 2

Background

This section will focus on important concepts that will be brought up throughout the duration of this thesis. This includes encryption schemes that were looked at for a basis for the secure integer comparison scheme that we created. The purpose of an encryption scheme is primarily for preserving the confidentiality of information that is being sent from one party to another. One of the main principles behind cryptography is Kerckhoff's principle (which can be found in many cryptography textbooks, for example Cryptography: Fundamentals and applications [43], or Handbook of applied cryptography [44]).

Definition Kerckhoff's Principle. *A cryptosystem should maintain its security regardless of what information, except the secret key, is known to the public.*

This principle states that security of information should not rely on the concealing of the algorithm itself, and that it should instead rely on the secrecy of the decryption key. Within this section we will talk about the basis of sending a secure message through an encryption and expand to talking about common day encryption schemes that were studied for the creation of our custom encryption scheme.

2.1 Secure Computation

Secure computation is the idea that two or more parties can cooperate to compute a value of mutual importance using their respective private inputs such that nothing other than the output is revealed. This concept was first introduced in the form of the Millionaires' problem by Yao [63], and expanded on by Blake and Kolesnikov [7]. Suppose there are m parties who want to be able to compute a function $f(x_1, x_2, x_3, ..., x_m)$ where $x_i \geq 0$ is the input for party P_i [4]. The goal is to be able to compute the function f with all of

the other parties, without letting anyone else know their respective value of x [56]. One of the most well known instances of secure computation is the Millionaires' problem.

Millionaires' Problem: The Millionaires' problem is based on the concept of two millionaires wanting to know who is richer without revealing to each other how much they are worth. In this instance of two party secure computation, the function is a secure integer comparison. Given two integers x_1, x_2 the Millionaires' problem is defined as:

$$f(x_1, x_2) = \begin{cases} 1 \text{ if } x_1 < x_2 \\ 0 \text{ otherwise} \end{cases} .$$

Definition Secure Computation. *A two party protocol is considered a secure computation if no information other than the output $f(x_1, x_2)$ is learned where party's P_1, P_2 only know their own respective inputs. The output should not give the receiver any indication of what the other party's initial input was into the function [24].*

The millionaires problem is directly correlated to the work that we have done for secure integer comparison. Though there are some efficient solutions already in practice (such as the solution described in Lin et al. [40], or Ioannidis et al. [34]), we have developed our own protocol that is able to give the resultant of the millionaire's problem under homomorphic encryption without leaking information about either party's inputs. The purpose of designing our own cryptosystem was to increase the efficiency by avoiding the bitwise decomposition method used by previous systems.

2.2 Public Key Encryption

The purpose of public key cryptography is to provide secure communication over a public network. With public-key encryption two people are able to communicate privately without ever having to have met to exchange a secret key. Public key cryptography uses two separate keys: one for encrypting messages that is public, and one used to decrypt messages that is private. For this type of encrpytion the public keys that are exchanged are visible on the network without obfuscation. Security in this environment is difficult because the information that is passed through the network can be intercepted by adversaries [60]. To be able to be truly secure over the network, a protocol using the encryption must uphold the three key aspects of security: confidentiality, integrity and authenticity. To do this, a protocol must preserve:

- **Confidentiality:** Not allow information to be readable while on the network

- **Integrity:** Not allow the message to be modified while on the network
- **Authenticity:** Prove that the message comes from the original sender

With these three principles, an adversary who was able to intercept the information should not be able to understand or modify the contents of any of the messages that were sent. The purpose of a public key cryptosystem is to preserve the integrity of the messages.

For a public key cryptosystem, each party has their own set of cryptographic keys. The public key is used for encryption and sending messages to other parties, while the private key is used for decryption and deciphering the messages that others have sent. These keys are called asymmetric keys because the keys used by each party within the exchange of a message are different (sender uses public key of recipient, receiver uses own private key). Distribution of one's public key is important as it is used by other parties within the encryption of their messages. In contrast, private keys should be kept secret as the decryption of a message should only be feasible for the owner of the private key. An allegory for this system is the physical mail system, where anyone should be able to send you mail, while only you should be able to read the mail addressed to you. The keys are generated together, which means that they are related mathematically, however there are certain principles that are upheld to make finding the private key from the public key impractical.

2.2.1 Cryptosystem Breakdown

Definition Public Key Cryptosystem: *A public key cryptosystem (referred to from here as a cryptosystem for the sake of brevity) can be broken down into the three separate functions that are required of it [13]. Firstly, a cryptosystem must be able to generate a set of keys in order for secure communication across the public network. Next, a cryptosystem must allow users to encrypt messages that are sent to others. Finally, a cryptosystem must allow for the decryption of messages that are received. The algorithms for each of these functions are listed below.*

$\mathsf{Gen}(\lambda) \rightarrow (p_k, s_k)$*: Given the security parameter* λ *(the length of the keys that will be created) [10],* Gen *returns a set of public and private keys,* p_k, s_k *respectively. This is the first action that takes place when first using a cryptosystem, as the keys are the basis for the security.*

$\mathsf{Enc}(m, p_k) \rightarrow c$*: This function takes in the plaintext of message* m *and the public*

key p_k of the user that you would like to send the message to and creates a valid ciphertext for the encryption c.

$\mathsf{Dec}(c, s_k) \rightarrow m$*: This function takes in a ciphertext c and uses the private key s_k to decrypt the message into its original form (the plaintext) m.*

For a cryptosystem to maintain these properties, there must not be any overlapping of ciphertexts. Every ciphertext must be able to be reverted into the original plaintext [54] (i.e. no two separate plaintexts should produce the same ciphertext). Let K be the set of all possible distinct key pairs that can be created by the cryptosystem. We say that a cryptosystem's encryption scheme is correct if:

$$\forall\{p_k, s_k\} \in K, Dec(Enc(m, p_k), s_k) = m.$$

Regardless of the key set that you have generated, the key pairing should allow for encryption and decryption to work cooperatively.

We have stated that every key pair generated by the cryptosystem should work, but we have not touched on the idea of the security of the messages themselves. The use of a public network allows for adversaries to view the ciphertexts of the messages that are being sent, yet they must not be able to decipher the plaintext of these messages. Although each ciphertext has a 1:1 relationship with a plaintext, multiple re-encryptions of the same plaintext should result in different ciphertexts. This is done with randomization which is described in Section 2.2.4. One way to provide this is through the use of perfect secrecy.

Definition Perfect Secrecy. *The ciphertexts of a cryptosystem should reveal nothing about the initial messages. This property is called perfect secrecy. For this property to be maintained,*

$$\forall c \in C, \forall m_0, m_1 \in M, Pr[C = c|m_0] = Pr[C = c|m_1].$$

Where C is the set of all possible ciphertexts and M is the set of all possible messages. This means that given two messages m_0, m_1 and a ciphertext c, the probability of the ciphertext coming from m_0 or m_1 should be the same. There should be no advantages to guessing the message based on the ciphertext. Shannon's theory emphasizes that for this to be possible, the secret key must be as long as the message itself [57].

2.2.2 Semantic Security

Definition Negligible Function*. A function $f : \mathbb{N} \to \mathbb{N}$ is called a* negligible function *if for any positive polynomial $p : \mathbb{N} \to \mathbb{N} \iff \exists i_0 \in \mathbb{Z}$ such that $f(i) < 1/p(i)$ for $i \geq i_0$ where i is any integer greater than 0.*

Definition Semantically Secure. *A cryptosystem is considered semantically secure if there exists no probabilistic polynomial time algorithm (PPT), that can decipher the plaintext from a given ciphertext. Given a ciphertext, it should be computationally infeasible to gain information about the plaintext of the message. Given two plaintexts, m_0, m_1, and ciphertext c, adversary A should have no advantage of guessing the correct plaintext.*

$$Adv[A, c] = |Pr(m = m_0|c = c) - Pr(m = m_1|c = c)| \text{ is negligible}$$

2.2.3 IND-CPA Game

Semantic security is equivalent to the Indistinguishable under Chosen-Plaintext Attack (IND-CPA) setting. Security within this setting is often defined using the following game-like structure. For the game, "Alice" will be the adversary looking to gain information about the scheme and "Bob" will be the oracle, who knows the keys for the encryption scheme.

1. A set of keys is generated by Bob $Gen(\lambda) \to (p_k, s_k)$.
2. Alice is given p_k and asks Bob to encrypt plaintexts and return their corresponding ciphertexts.
3. Alice creates 2 messages, m_0, m_1 and sends them with p_k to Bob.
4. Bob chooses one of the messages $b \leftarrow_{\$} \{0, 1\}$ and chooses to encrypt m_b.
5. Bob computes $c_b = \mathsf{Enc}_{\mathsf{p_k}}(\mathsf{m_b})$ and returns c_b Alice.
6. Alice outputs her guess $g \leftarrow \{0, 1\}$ for which message was encrypted, and wins if $g = b$.

The system is considered semantically secure under IND-CPA if Alice is unable to determine which message was picked with a probability that is significantly greater than or less than 50%. For this to be achieved, a semantically secure cryptosystem

must be able to generate different ciphertexts for the same plaintext. Otherwise, in a deterministic function, Alice can simply encrypt the messages herself before sending them off as challenge messages. This means that the encryption of a ciphertext must be probabilistic and change with each encryption (discussed in Section 2.2.4). The public key cryptosystems DGK, Paillier and ElGamal, which we will talk about in further sections are all semantically secure because gaining advantage in this game scenario would be the equivalent of solving hard mathematical problems such as the residuousity problem described later in this thesis.

2.2.4 Randomized Encryption

Randomized encryption is the use of randomness within an encryption algorithm so that the encryption of the same message over and over will result in different ciphertexts. This property is needed in public key cryptosystems for semantic security. In the public key setting randomized encryption can also be called probabilistic encryption because the probability of a ciphertext relating to any two messages should be the same. Cryptosystems usually deal with this randomization factor by splitting their total message space into two distinct subsections. These are the message space and the randomiser space. An example of this is shown below:

$$M(f^n) \rightarrow M(g^r h^s)$$

Where $\mathcal{M}$'s total message space of n-bits can be broken down into r-bits for the message space g and s-bits for the randomiser space h.

2.2.5 Math Basics

Below is a list of some of the key mathematical concepts that are used within public key encryption systems.

Group: *A group is an algebraic structure that is made up of two components: a set of elements $\mathcal{G}$ and a binary operation $(\cdot)$. A group works under closure, such that $\forall a, b \in \mathcal{G}, (a \cdot b) \in \mathcal{G}$. Along with the closure property, there are three main axioms that define the group (denoted as $\mathcal{G}$):*

1. *Groups are Associative: $\forall a, b, c, \in \mathcal{G}, (a \cdot b) \cdot c = a \cdot (b \cdot c)$*
2. *Groups have an Identity Element: $\exists i \in \mathcal{G} \mid \forall a \in \mathcal{G}, i \cdot a = a$. Here i is the identity element.*

3. Elements have an Inverse: $\forall a \in \mathcal{G}, \exists b \in \mathcal{G} \mid a \cdot b = i$. *Here* i *is the identity element.*

Abelian Group: *An abelian group is a group that also has the commutative property. This additional property states:*

$$\forall a, b \in \mathcal{G}, a \cdot b = b \cdot a.$$

Cyclic Group: *A cyclic group, is an abelian group in group theory that can be generated from a single element. This element is called the generator for the cyclic group. These generators are a key concept the field of public key cryptography. Let us say that* $\mathcal{G}$ *is a cyclic group that can be generated by a value* $x \in \mathcal{G}$*. This would mean that every element in* $\mathcal{G}$ *is equal to* x^k *for a value* $k \in \mathbb{Z}$*. This type of group structure can be divided into two different cases, infinite cyclic groups and finite cyclic groups [22]. If* $\mathcal{G}$ *is infinite, then all of the* x^k *are unique and* $\mathcal{G}$ *is isomorphic to* $\mathbb{Z}$*. In the case where* $\mathcal{G}$ *has a finite order* n *every element in* $\mathcal{G}$ *can be represented as* x^k *where* $0 < k < n$ *or* $k \in \{0, ..., n-1\}$*. Here* $\mathcal{G}$ *is isomorphic to* $\mathbb{Z}/n\mathbb{Z}$ *Though it is stated that a single element can be used to generate a cyclic group, there is often more than just one element that can be used as the generator for the cyclic group that was generated [1].*

Computational Indistinguishability: *The concept of computational indistinguishability is based on comparing the two distributions of results based on a security parameter* n *[30]. Let us take a look at two distribution ensembles* $\{R_n\}_{n\in\mathbb{N}}$ *and* $\{S_n\}_{n\in\mathbb{N}}$*. We say that these sets are computationally indistinguishable for a probabalistic polynomial time algorithm* A *if:*

$$\delta(n) = \left| \Pr_{x \leftarrow R_n}[A(x) = 1] - \Pr_{x \leftarrow S_n}[A(x) = 1] \right|$$

is a negligible function in n *[9]. This means that given a specific value that has come from one of these sets, the likelihood of the value coming from both sets should be the same and therefore there should be no advantage in guessing which set it was chosen from [61].*

2.3 Homomorphic Encryption

Homomorphic Encryption is based on a property in algebra known as homomorphisms. This property allows information to be changed into a different set of information, yet still

maintain properties and associations from the first set. While initially garbled circuits were implemented to solve the millionaires problem, homomorphic encryption protocols have taken over as the main was to do secure computation [33].

Definition Homomorphism. *A Homomorphism is a transformation of one set into another that preserves in the second set the relations between elements of the first. This means that there exists a function*

$$f : A \to B$$

such that an operation $\cdot$ from within the A domain, can be done in the B domain using $*$. This can be shown as

$$f(x \cdot y) = f(x) * f(y).$$

Homomorphic encryption is a form of encryption that allows for some computations to be performed on the ciphertext without decrypting the ciphertext. The result of the operations is returned as an encrypted result, which when decrypted is the same as if some operation was performed on the plaintext [17].

Definition Homomorphic Encryption. *Homomorphic encryption is a type of encryption where a set of operations performed on ciphertexts results in some other operation being performed on the plaintexts [26]. This means that we can manipulate plaintexts while they are under encryption. This can be shown as:*

$$E(x) * E(y) = E(x \cdot y).$$

Some applications for such a system are the implementation of secure bidding systems [8], cloud computing [38] and data aggregation [36]. There are many partially homomorphic cryptosystems that allow for some specific operations to be performed (namely addition and multiplication), but due to some major drawbacks of fully homomorphic encryption (such as the intense amount of computational power needed) fully homomorphic encryption is not very practical. Some examples of such drawbacks are processing time and implementation complexity.

Many cryptosystems with homomorphic properties have been around for quite a while. For example, DGK, Paillier, and ElGamal are partially homomorphic with Paillier developed in 1999 [48]. While these systems are partially homomorphic, there are also fully homomorphic cryptosystems that have been developed. An example of a fully homomorphic encryption scheme is the lattice based approach by Craig Gentry [29]. Though this type of system is possible, it is not practical using today's resources [5].

2.4 DGK Public Key Cryptosystem

The DGK protocol was created by Damgard, Geisler and Kroigaard in 2007 [16], [18] as an efficient solution to solving the millionaire's problem. For the creation of the protocol, they were forced to form an entirely new homomorphic cryptosystem. The DGK protocol is currently being used in many applications that deal with signal processing as well as statistical analysis on sensitive data, k-means clustering on encrypted data and bioinformatics (for example Franz et al.'s work on bioinformatics [27]). It is also often improved upon (for example Garay et al.'s work [28]), as over time cryptographers get more acquainted to it. This is the cryptosystem that we will be comparing our cryptosystem with in future chapters, as it is the most commonly used secure integer comparison method.

Key Generation: Choose two t-bit primes v_p and v_q and two distinct primes p, q of equal bit length such that

$$v_p \,|\, p-1$$

$$v_q \,|\, q-1$$

then choose an ℓ-bit prime u and a $g \in \mathbb{Z}^*$ with order uv_pv_q and choose h to have order v_pv_q. In this construction the g generator is considered the message space, where the input of the message will be present and the h generator is of the randomiser space and is used to obfuscate the numbers within g. The public (encryption) key is (n, g, h, u) and the private (decryption) key is (p, q, v_p, v_q). In addition to the key generation, an auxiliary lookup table is made of tuples $(g^{v_pv_q})^i$ for $0 \leq i \leq u$ which have corresponding values of i

Encryption: The encryption of a DGK message is not only tied to the message itself, but also requires a random r to be generated for the ciphertext (r is chosen $r \in \{1, \ldots, n\}$). Let m be a message to be encrypted, the ciphertext for m is

$$c = g^m h^r \bmod n$$

Decryption: Let c be the ciphertext chosen for decryption, where $c \in \mathbb{Z}_n^*$, The plaintext message can be found by computing

$$m' = c^{v_pv_q} \bmod n$$

$$m' = (g^m h^r)^{v_pv_q} \bmod n$$

Recall that the generator h has order v_pv_q, this means that the generator cancels out and

becomes 1, which leaves

$$m' = (g^{v_p v_q})^m \bmod n.$$

m' can then be found within the auxiliary table that was created within the key generation.

Homomorphic Properties

The DGK is additively homomorphic and has the same homomorphic properties as the Paillier cryptosystems as seen below.

$$\forall m_1, m_2 \text{ and } k \in \mathbb{Z}_n^*$$

$$\begin{aligned}
\mathsf{Dec}\Big(\mathsf{Enc}(m_1)\mathsf{Enc}(m_2) \bmod n\Big) &= (m_1 + m_2)^{v_p v_q} \bmod n \\
\mathsf{Dec}\Big(\mathsf{Enc}(m_1) g^{m_2} \bmod n\Big) &= (m_1 + m_2)^{v_p v_q} \bmod n \\
\mathsf{Dec}\Big(\mathsf{Enc}(m_1)^{m_2} \bmod n\Big) &= (m_1 m_2)^{v_p v_q} \\
\mathsf{Dec}\Big(\mathsf{Enc}(m_2)^{m_1} \bmod n\Big) &= (m_1 m_2)^{v_p v_q} \\
\mathsf{Dec}\Big(\mathsf{Enc}(m)^{k} \bmod n\Big) &= (km)^{v_p v_q}
\end{aligned}$$

Secure Integer Comparison One of the large benefits of using the DGK system is that the use of smaller primes reduces the size of the modulus and allow multiple rapid encryptions and decryptions. This specific property was chosen for the approach that Damgard, Geisler and Kroigaard had towards secure integer comparison. When comparing the bitwise decomposition of integers x and y, their approach was to scan both bit rows from left (the most significant part) to right searching for the first differing bit. The outcome of the comparison of these differing bits will determine the comparison result of both integers. Assume both integers contains ℓ bits denoted by x_i and y_i respectively, this leads to a breakdown of

$$x = x_{\ell-1} \ldots x_2 x_1 x_0$$

where the most significant bit of x is denoted as $x_{\ell-1}$. Then the ciphertexts c_i, $0 \leq i < \ell$ are computed which will only be zero when $x_j = y_j$ for each $j, i < j < \ell$ and at the same time $x_i \neq y_i$.

A more intuitive way of looking at the value for c_i is as follows,

$$c_i = s + x_i - y_i + 3 \sum_{j=i+1}^{\ell-1} *(x_j \oplus y_j).$$

The sum of exclusive ors will be 0 exactly when $x_j = y_j$ for each $j, i < j < \ell$. The variable

s can be set to either -1 or 1 depending on the comparison that is performed (greater than or less than). For example when $s = -1$, c_i will only be zero when $x_i = 1$ and $y_i = 0$ (and $x_j = y_j$ for each $j, i < j < \ell$) which means that $x > y$. To avoid one of the parties learning the comparison result, one party will set the parameter s and the other party will learn if $c_i = 0$.

2.4.1 Modern Use

While the DGK scheme was first proposed in 2007, there have been few advancements in the field of secure integer comparison. Recent work done using secure integer comparison like Xiang et al.'s work in the field of secure facial recognition still use this protocol in their studies in 2016 [62]. While the implementation of the protocol represents it's use in industry, it is also important to note that the DGK protocol is still the baseline for which secure comparison protocols compare themselves to [15]. In Nateghizad et al.'s work in trying to create a protocol specifically for metering systems [46] they compare their efficiency to the DGK protocol, stating that for their exact situation, the use of their protocol is marginally better than DGK. Other examples of the DGK protocol being mentioned as today's standard include the works of Li et al. [39] and Couteau [15]. While there have been a few advances in the efficiency of DGK itself such as Veugen's work in 2011 [60], these advancements still have to deal with the drawback of using the binary decomposition within DGK (the large number of encryptions and decryptions) and are not as significant as our attempt to replace this process altogether.

2.5 Paillier Encryption

Pascal Paillier invented the Paillier encryption scheme in 1999 [48]. This scheme used a probabilistic public-key cryptosystem structure. The hardness assumption for this cryptosystem was based on decisional composite residuousity, which deals with the hardness of dedciding if a number is an nth residue modulo n^2 (which is computationally difficult [19]). This cryptosystem was studied as it is often used in conjunction with the DGK protocol for a second layer of security when making slight variations to the protocol itself.

Paillier's encryption scheme can be broken into it's key generation, encryption, and decryption algorithms [64]. They are listed as follows:

Key Generation: Two large prime numbers p and q are chosen at random and are

independent from one another. As long as the primes are of equal length, the property

$$gcd(pq, (p-1)(q-1)) = 1$$

stands. Here gcd means the greatest common divisor (largest number that divides both pq and $(p-1)(q-1)$). From these two primes, n can be computed as

$$n = pq$$

and $\lambda = lcm(p-1, q-1)$. Here lcm means the least common multiple (smallest number that is divisible by both $p-1$ and $q-1$). The next step in key generation is choosing a random integer g where $g \in \mathbb{Z}_n^*$. To guarantee that n divides the order of g we can check to see if there is a modular inverse μ. To do this, first we create a function L where u is the security parameter and

$$L(u) = (u-1)/n.$$

Using this L we can generate the inverse as:

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

For the encryption scheme, the public key is (n, g) and the private key is (λ, μ).

Encryption: Let $m \in \mathbb{Z}_n^*$ represent a message that will be encrypted. Generate r such that $r \in \mathbb{Z}_{n^2}^*$. Using these two parameters, we can compute the ciphertext for a Paillier encryption as:

$$c = g^m r^n \bmod n^2.$$

Decryption: Given a valid ciphertext c where $c \in \mathbb{Z}_n^*$, the plaintext of the ciphertext can be computed as:

$$m = L\Big(c^\lambda (mod n^2)) \cdot \mu\Big) \bmod n$$

This decryption is "essentially one exponentiation modulo n^2" as pointed out by Paillier in his original paper [48].

This encryption scheme harnesses certain discrete logarithms that can be easily computed to it's advantage.

Homomorphic Properties: One of the key characteristics of Paillier's encryption scheme is the homomorphic properties that it possesses. Under Paillier's encryption

scheme, we have both homomorphic addition and homomorphic multiplication.

Homomorphic Addition of Plaintexts: Given $r_1, r_2 \leftarrow_{\$} \mathbb{Z}_n^*$, let the ciphertexts in this section be represented as $E(m_1, p_k) = g^{m_1} r_1^n (mod n^2)$ and $E(m_2, p_k) = g^{m_2} r_2^n (mod n^2)$. Here the multiplication of two ciphertexts under encryption is equivalent to the addition of the plaintexts within the encryption, i.e.,

$$\mathsf{Dec}\Big(\mathsf{Enc}(m_1, p_k) \cdot \mathsf{Enc}(m_2, p_k) \bmod n^2\Big) = (m_1 + m_2) \bmod n.$$

This can be seen when inspecting what actually happens within the multiplication:

$$\begin{aligned}\mathsf{Enc}(m_1, p_k) \cdot \mathsf{Enc}(m_2, p_k) &= (g^{m_1} r_1^n)(g^{m_2} r_2^n) \bmod n^2 \\ &= g^{m_1+m_2}(r_1 r_2)^n \bmod n^2 \\ &= \mathsf{Enc}(m_1 + m_2, p_k).\end{aligned}$$

This homomorphic addition also works if you multiply an encryption by a message within the message space (in the form g^m). This can be shown as

$$\mathsf{Dec}\Big(\mathsf{Enc}(m_1, p_k) \cdot g^{m_2} \bmod n^2\Big) = (m_1 + m_2) \bmod n.$$

These multiplications reduce to the addition of the plaintexts by using the "product rule" which allows exponents of the same base to be added when they are being multiplied. The exponent found within the randomiser space does not effect the decryption because the decryption key nullifies the space and deals only with the message space.

Homomorphic Multiplication of Plaintexts: Another property of Paillier's scheme is that raising a ciphertext to the power of a plaintext results in the product of the two plaintexts under encryption:

$$\mathsf{Dec}\Big(\mathsf{Enc}(m_1, p_k)^{m_2} \bmod n^2\Big) = m_1 m_2 \bmod n.$$

This equation can be broken down further for easier analysis:

$$\begin{aligned}\mathsf{Enc}(m_1, p_k)^{m_2} &= (g^{m_1} r_1^n)^{m_2} \bmod n^2 \\ &= g^{m_1 m_2}(r_1^{m_2})^n \bmod n^2 \\ &= \mathsf{Enc}(m_1 m_2, p_k).\end{aligned}$$

This multiplication of the plaintexts can also be used in a scalar fashion by raising the

encryption to a power p. This will multiply the encrypted plaintext by p.

$$D(E(m_1, p_k)^p \bmod n^2) = pm_1 \bmod n$$

2.6 ElGamal Encryption

The ElGamal encryption scheme was invented by Taher ElGamal in 1985 [23] and uses the Diffie-Helman key exchange as the foundation for it's public-key algorithm. The security of the system is based on Decisional Diffie-Helman assumption. This encrpytion scheme is defined over a cyclic group $\mathcal{G}$. ElGamal encryption is important to our protocol as we can use it to extend the protocol, hiding the difference between the numbers within the secure comparison.

ElGamal's encryption scheme can be broken into it's key generation, encryption, and decryption algorithms [64]. They are listed as follows:

Key Generation: Party P_1 determines an efficient description of a cyclic group $\mathcal{G}$ with q elements and a generator g. Here an "efficient description" just means that not all of the elements of the group are written out for the other party (P_2).
P_1 chooses $x \in \{1, \ldots, q-1\}$. and computes

$$y = g^x.$$

The public key for the cryptosystem is (G, q, g, y) and the private key is (x). This x value must be kept secret.

Encryption: For the encryption of a message m, Party P_2 uses the public key given by P_1 (G, q, g, y) and a value $r \leftarrow_\$ \{1, \ldots, q-1\}$. P_2 computes

$$c_1 = g^r$$

and a shared secret

$$s = y^r.$$

P_2 then transforms it's secret message m, into $m' \in \mathcal{G}$ and computes

$$c_2 = m' \cdot s.$$

P_2 then sends (c_1, c_2) to P_1.

If you know m', it is easy to determine the y^r, because you can use c_2 and divide out m'. This caveat means that to ensure security the value of r should be ephemeral. This means a new r should be chosen for each message causing it to change through the communication.

Decryption: For the decryption of a ciphertext (c_1, c_2), party P_1 computes the shared secret using her private key x as

$$t = c_1^x.$$

Recall that $c_1 = g^r$, which makes $t = g^{rx}$. P_1 then computes

$$m' = c_2 t^{-1} = c_2 (g^{rx})^{-1}$$

where t^{-1} is the inverse of t in the group $\mathcal{G}$ (as explained in Section 2.2.5). P_1 then transforms m' back into the original message m.
The equation below shows that the decryption results in the correct message:

$$\begin{aligned} c_2 t^{-1} &= (m's) c_1^{-x} \\ &= m' y^r g^{-xr} \\ &= m' g^{xr} g^{-xr} \\ &= m'. \end{aligned}$$

This scheme, much like Paillier is probabilistic in that plaintexts are not limited to just one resulting ciphertext. The expansion of the size of plaintexts to ciphertexts is a 2:1 ratio [23]. For the encryption function, ElGamal requires 2 exponentiations (pre-computable as independent from the message) and for the decryption function only 1 exponentiation is needed.

Homomorphic Property: ElGamal has an multiplicative homomorphic property where the multiplication of two ciphertexts results in the multiplication of the two plaintexts. Given the encryptions for two messages $m_1, m_2 \in \mathcal{G}$ using r_1, r_2 which are randomly chosen from $\{1, \ldots, q-1\}$. The encryptions can be shown as

$$(c_{11}, c_{12}) = (g^{r_1}, m_1 y^{r_1})$$

$$(c_{21}, c_{22}) = (g^{r_2}, m_2 y^{r_2}).$$

Using these encryptions it is possible to compute

$$\begin{aligned}(c_{11}, c_{12})(c_{21}, c_{22}) &= (c_{11}c_{21}, c_{12}c_{22}) \\ &= g^{r_1}g^{r_2}, (m_1 y^{r_1})(m_2 y^{r_2}) \\ &= g^{r_1+r_2}, (m_1 m_2) y^{r_1+r_2}.\end{aligned}$$

This ciphertext is the encryption of $m_1 m_2$.

ElGamal can also be implemented where the cyclic group $\mathcal{G}$ is an elliptic curve. This results in a faster computational time than exponential ElGamal, while being based on the same principles. Elliptic curve ElGamal is used in the implementation section of the thesis, but it was not studied in detail and the curve used was chosen from a set provided by the National Institute of Standards and Technology (NIST) [47]. The particular curve that we use is the P-256 curve, also known as the sec256r1 [52].

Chapter 3

Our Protocol

3.1 Introduction

Our approach in finding a more efficient scheme for the secure comparison of integers was to try and create something brand new. When looking at possible options for the structure of the encryption, it was clear to us that we wanted to incorporate homomorphic properties into the encryption so that all of our computations could be done under encryption. While studying the different types of homomorphic properties that are used, we discovered a new property wherein a threshold function could be used within an exponent to expose information about the inputs. After discovering this new property (discussed further in Section 3.5.1), we decided to base the new cryptosystem on showcasing this property and the correct way to exploit the information learned from the threshold function. The subsequent sections in this chapter explain the breakdown of the cryptosystem that was developed and the protocol used for secure integer comparison using the cryptosystem. These sections are an expansion on our paper [12].

3.2 Mathematical Principles

Within our encryption scheme we are working with an RSA modulus of n, where $n = p \cdot q$ and p and q are relatively large primes. Each of these primes is comprised of a set of generators of different order. The three sections of p and q are:

1. The message space b where b is a small prime base (e.g. 3).

2. The randomiser space p_s, q_s which is used for obfuscation.

3. The padding space p_t, q_t which is used to extend the length of the message to fit the parameters.

Put together, these parts make up p and q as follows:

$$p = 2b^d p_s p_t + 1$$

$$q = 2b^d q_s q_t + 1$$

In this construction p_s, p_t, q_s, q_t are pairwise distinct primes (meaning they are all different). We note that we are also working in $\mathbb{Z}_n^*$ ring which encompasses all non negative integers from 0 to $n-1$. By deconstructing $\mathbb{Z}_n^*$, we can see that there are two cyclic subgroups within the the larger space. There is a cyclic subgroup $\mathbb{G}$ of order b^d and a unique cyclic subgroup $\mathbb{H}$ of order $p_s q_s$. These subgroups define the message and randomiser spaces of the encryption.

$$\begin{aligned}\mathbb{Z}_n^* &\cong \mathbb{Z}_{2b^d p_s p_t} \times \mathbb{Z}_{2b^d q_s q_t} \\ &\cong (\mathbb{Z}_2)^2 \times (\mathbb{Z}_{b^d})^2 \times \mathbb{Z}_{p_s q_s} \times \mathbb{Z}_{p_t q_t}\end{aligned}$$

With $g \leftarrow \mathbb{G}$ and $h \leftarrow \mathbb{H}$ as random generators,the public key for the cryptosystem is $p_k = (n, b, d, g, h, u)$. In this key, u represents the bit-length of p_s and q_s and is a security parameter for the protocol. Let the notation $x \leftarrow\!\!\$\, S$ denote a value x sampled uniformly at random from a set S. For the encryption of a message $m \in \{0, \dots b^d - 1\}$, P_1 creates a random $r \leftarrow\!\!\$\, \{1, 2, \dots, 2^u - 1\}$, and sends $c = g^m h^r \bmod n$. The decryption of c takes the value of $p_s q_s$ (which is the order of h) and computes $c^{p_s q_s} = (g^m)^{p_s q_s}$. By finding the inverse of $p_s q_s \bmod n$, which we call x, we can solve the discrete logarithm problem by computing:

$$g^m = (c^{p_s q_s})^x.$$

Since g is an element of order b^d this can be done in $O(d\sqrt{b})$ operations, which is efficient in the size of b and d.

The last aspect of the mathematical description of the scheme, is an explanation on how to efficiently choose the generators g and h of the respective subgroups $\mathbb{G}$ and $\mathbb{H}$. Generator h is chosen in the same manner as the generators of the respective randomiser spaces of the schemes of Groth and Damgård et al. [32], namely we find generator h_{p_s} (resp. h_{q_s}) of the subgroup of $\mathbb{Z}_p^*$ (resp. $\mathbb{Z}_q^*$) of order p_s (resp. q_s). The procedure for finding h_{p_s} and h_{q_s} is straightforward, and is found in most software implementations of the discrete logarithm problem over finite fields (e.g., Diffie-Helmman, RSA, Elgamal,

etc) [23]. Next we use the Chinese remainder theorem to find h such that

$$\begin{aligned} h &\equiv h_{p_s} \bmod p \\ h &\equiv h_{q_s} \bmod q. \end{aligned}$$

g is chosen in the same manner, i.e., find a generator g_{b^d} of a subgroup of order b^d separately in $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ and use the Chinese remainder theorem to compute g in the manner above.

3.3 Hardness Assumptions

Given the parameters as above, it should be infeasible to distinguish between a randomly selected element of $\mathbb{Z}_n^*$ and an element of order $p_s q_s$, without factoring n [59]. This should be hard, even for an adaptive adversary [11]. To prove this, we begin by looking at the structure of the public key that is created using our algorithm.

Definition An *RSA quintuple* is a quintuple (n, b, d, g, u) where:

1. u is an integer such that the Discrete Logarithm Problem is infeasible in a group whose order is a prime of bit-length u;
2. b is a prime of bit-length less than u;
3. d is an integer greater than 1;
4. n is an integer $n = pq$, whose factorization is infeasible, where:

$$p = 2b^d p_s p_t + 1 \quad \text{and} \quad q = 2b^d q_s q_t + 1;$$

   and where p_s and q_s are primes of bit-length u, and p_t, q_t are primes whose bit-length is not u;
5. g is an element of order b^d in $\mathbb{Z}_n^*$, which is the message space generator.

We point out that an RSA quintuple (n, b, d, g, u) is only one number short of a public key in our encryption scheme (Section 3.6). The final parameter is used to define the problem and the corresponding hardness assumption.

Small RSA Subgroup Decision Problem Given an RSA quintuple (n, b, d, g, u) and $x \in \mathbb{Z}_n^*$, determine whether or not x has order $p_s q_s$. From this decision, output '`yes`' if x has order $p_s q_s$ and '`no`' if it does not have the proper order.

Note that due to the requirements on the length of p_s, q_s, p_t, and q_t, this gives a well-defined decision problem. Of course, if we could factor n, then the problem would be easy to solve. However, without the ability to factor n, it appears to be infeasible, which leads us to the following definition:

Small RSA Subgroup Decision Assumption Given an RSA quintuple (n, b, d, g, u) and $x \in \mathbb{Z}_n^*$, we say that $\mathcal{G}$ satisfies the Small RSA Subgroup Decision Assumption if for any polynomial time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in solving the Small RSA Subgroup Decision Problem is negligible, i.e. the only way to determine if x has order $p_s q_s$ is to factor n.

This assumption is very similar to an assumption presented by Groth [32], and an assumption by Pointcheval [50]. Even though they are similar, to our knowledge our assumption cannot be reduced to these assumptions.

3.4 Working in Prime Power Groups

Generator of a prime power subgroup In Section 2.2.5 we talked about the concept of generators and their role in cyclic groups and cryptography. One of the unique properties of our protocol is that we have chosen to work in what is called a 'prime power group'. This differs from a typical cyclic subgroup as the order that we are using is power of a small prime instead of the typical power of a large prime. To create a secure generator of a prime power subgroup, one must find two numbers that are relatively prime to the order of the group. Secondly, you need to make sure that the numbers chosen are also relatively prime to the order of the group as well. Below is an algorithm for finding a proper generator g_{b^d} of a subgroup of $\mathbb{Z}_p^*$ (for a prime p) of order b^d (finding $\mathbb{Z}_{b^d}^*$ for prime p in $n = p \cdot q$):

while $True$:
  $x \leftarrow_\$ \{2 \ldots p-2\}$
  $y = x^{(p-1)/b} \mod p$
  if $y \neq 1$:
    return $x^{(p-1)/b^d}$.

This procedure is repeated to find a generator g_{b^d} of a subgroup of $\mathbb{Z}_q^*$ (for prime q), the Chinese remainder theorem is used to combine the two generators to produce g, a generator of a subgroup of order g_{b^d} of $\mathbb{Z}_n^*$ (where $n = pq$).

3.5 Our Encryption Scheme

This section discusses the algorithms that form the basis of the cryptosystem that we have created. The composition of our protocol is similar to the composition found in Section 2.2.1, as we have structured our protocol to be a public key cryptosystem. Below we list the three functions that comprise our system.

Gen(λ)**:** For our generator $\mathcal{G}$ we take the input $\lambda \in \mathbb{Z}^+$ as the security parameter, which in this case is the length of the keys that are to be generated. This Gen function produces a pair (ℓ, u) where ℓ is the length where factoring the product of two random ℓ bit primes is infeasible for a PPT-bounded machine, and where u is the length where solving a discrete log in a group of u order is infeasible for a PPT-bounded machine. After the generator obtains (ℓ, u), a small prime base b and message space with an upper bound of $d \in \mathbb{Z}^+$ are chosen. Next we need to find a modulus for the message space. This modulus n, will be generated from two primes p and q where $n = pq$. The construction of p and q are as follows:

$$\begin{aligned} p &= 2b^d p_s p_t + 1 \\ q &= 2b^d q_s q_t + 1, \end{aligned}$$

In this example, the composition of the primes are as follows: $2b^d$ is within the message space, p_s is the within the randomiser space, and p_t is within the padding space. Let the length of p_s, q_s be u bits so that solving the discrete logarithm of the randomiser generator is computationally infeasible. Next we need to decide on the length of the padding space. For this we need to look at ℓ and see if the length of $\lceil \log_2(b^d) \rceil + u > \ell$. We need the primes that we are using to be have sufficient length such that factorization is infeasible. If this is the case, we can simply set $p_t = q_t = 1$ and ignore the padding space of the message. If $\lceil \log_2(b^d) \rceil + u < \ell$, we need to choose a value v to pad the encryption so that it is sufficiently secure. To do this we can choose v where $v = \ell - (\lceil \log_2(b^d) \rceil + u)$. Since we now can account for both the ℓ and u parameters to be fulfilled, we can create generators for the groups. Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_n^*$ of order or b^d modulo both p and q. This subgroup will be what we use for the message space of our encryption. Next we can create a subgroup for the randomiser space by finding a subgroup $\mathbb{G}_r$ of $\mathbb{Z}_n^*$ of order $p_s q_s$. We are able to pick random generators $g \leftarrow_\$ \mathbb{G}$ (cf. Section 2.2.5) and $h \leftarrow_\$ \mathbb{G}_s$. Ultimately, find x where $x' = (p_s q_s)^{-1} \bmod b^d$ and $x = p_s q_s x'$. This x value is used to cancel out the randomiser space and recover solely what is found in the message space.

For our protocol, the public key is $p_k = (n, b, d, g, h, u)$, and the private key is $s_k = (x)$.

$\mathsf{Enc}(p_k, m)$**:** The message space for the comparison consists of integers in the range $(0 \dots b^d - 1)$. Any number greater than $b^d - 1$ will fill a spot in the generator that is already occupied, which ruins the 1:1 ciphertext to plaintext relation mentioned in Section 2.2.1. For the encryption of message m, a party uses the values from the public key p_k and picks a random r within the constraints of the protocol, $r \leftarrow_{\$} \{1, \dots, 2^u - 1\}$. With these values, a party is able to create ciphertext c by computing

$$c = g^m h^r.$$

The random r provides the probabilistic distribution required for perfect secrecy and semantic security as referenced in Section 2.2.1 and Section 2.2.2.

$\mathsf{Dec}(s_k, c)$**:** To decrypt a ciphertext c using private key s_k, compute

$$c^x = c^{p_s q_s x'} = (g^m h^r)^{p_s q_s x'} = (g^m)^{p_s q_s x'} h^{r p_s q_s x'} = g^{m p_s q_s x'}.$$

Because the order of the randomiser group is $p_s q_s$, raising the generator h to $p_s q_s$ makes the output of the generator 1. This isolates the message space and allows for the decryptor to view the message without obfuscation. Since x' was chosen such that $x' = (p_s q_s)^{-1} \bmod b^d$ and we know that x' exists because $\mathsf{gcd}(b, p_s q_s) = 1$, $p_s q_s x' \equiv 1 \bmod b^d$, it will cancel itself out within generator g.

$$c^{x'} = (g^{m p_s q_s})^{x'} = g^m.$$

Last of all, we can recover the original message m from g^m by computing the discrete logarithm. Since we have chosen a small prime base b, we will only have to do d computations of the discrete logarithm of order b. This is efficiently computable due to the size of b.

Remark Throughout this section we have been constructing our message space using $2b^d$. In the case where we want $b = 2$, we can reduce the complexity of the creation for the primes by creating p and q in the following manner:

$$\begin{aligned} p &= 2^d p_s p_t + 1, \\ q &= 2^d q_s q_t + 1. \end{aligned}$$

Remark Within our cryptosystem we also found an interesting and unexplored homo-

morphic property that we use for secure integer comparison. To accomplish this, we can restrict the set of possible messages to $m \in \{b^0, b^1, b^2, \ldots, b^{d-1}\}$. This allows us to encrypt a message $m \in \{0, 1, \ldots, d-1\}$ by putting:

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

We use this set restriction as well as the unique homomorphic property in our secure comparison protocol presented in Section 3.6.

Next we will discuss the unique homomorphic property that we found when restricting the message space of our cryptosystem.

3.5.1 Unique Homomorphic Properties

Much like the Paillier and DGK protocols described in Chapter 2, our encryption scheme is one with additively homomorphic properties (also described by Rivest et al. [53]). When working in a group modulo b^d, multiplication of two encryptions results in the addition of their plaintexts under encryption (shown by Bendlin et al. [6]). This can be shown as:

$$\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) \bmod n = \mathsf{Enc}(m_1 + m_2 \bmod b^d).$$

We also retain the scalar multiplicative homomorphic property of these schemes where raising an encryption to the power of a message results in the multiplication of the two messages under encryption:

$$\mathsf{Enc}(m_1)^{m_2} \bmod n = \mathsf{Enc}(m_1 m_2 \bmod b^d).$$

This scalar multiplicative property can be expanded upon when we force our messages to be powers of our small prime base b, as referenced in Section 3.5. Using the scalar multiplicative homomorphic property of our scheme and reducing messages to the form b^m, we can multiply in the message, which results in the addition of their exponents:

$$\mathsf{Enc}(b^{m_1})^{b^{m_2}} \bmod n = \mathsf{Enc}(b^{m_1} b^{m_2} \bmod b^d) = \mathsf{Enc}(b^{m_1+m_2} \bmod b^d).$$

When looking at $b^{m_1+m_2} \bmod b^d$ it is evident that if $m_1 + m_2 > d$, it will be congruent to 0 and $b^{m_1+m_2}$ will stay a value of 1 regardless of the difference (as $b^0 = 1$). This results in a threshold function where we are able to set an upper limit for our secure integer comparison and see which side the addition falls under.

The threshold function of $m_1 + m_2$ under a modulo of b^d looks as follows:

$$b^{m_1+m_2} \bmod b^d = \begin{cases} b^{m_1+m_2} & \text{if } (m_1 + m_2) < d \\ b^0 = 1 & \text{otherwise.} \end{cases}$$

For the purpose of the encryption itself, we have:

$$\mathsf{Enc}(b^{m_1})^{b^{m_2}} = \begin{cases} \mathsf{Enc}(b^{m_1+m_2}) & \text{if } (m_1 + m_2) < d \\ \mathsf{Enc}(1) & \text{otherwise.} \end{cases}$$

Within our protocol, we use a positive value for one message and a negative value for the other. By doing this we are able to tell which value is larger. This modification changes the threshold function into:

$$\mathsf{Enc}(b^{d-m_1})^{b^{m_2}} = \begin{cases} \mathsf{Enc}(b^{d-m_1+m_2}) & \text{if } m_1 > m_2 \\ \mathsf{Enc}(1) & \text{otherwise.} \end{cases}$$

Note: This threshold homomorphism is not related to a *threshold cryptosystem* e.g. the cryptosystem of Schoenmaker [55].

3.5.2 Semantic Security

In this section, we prove the semantic security of our system.

Theorem 3.5.1 *The encryption scheme presented above is semantically secure, provided that the Composite Order Subgroup Decision Assumption of Section 3.3 is satisfied.*

To prove that our system is semantically secure, we will use proof by contradiction. This method theorizes the possibility of an algorithm $\mathcal{A}'$ that is able to break our encryption scheme with an advantage $\varepsilon(\tau)$ (which is not a negligible value). Since our hardness assumption is based off of the Small RSA Subgroup Decision Problem, referenced in Section 3.3, it is sufficient to say that the problem reduces to a polynomial time algorithm $\mathcal{A}$ generated by $\mathcal{A}'$. This algorithm $\mathcal{A}$ must be able to decipher whether a value x can be tied to a valid public key for our scheme.

Proof Suppose there exists a polynomial time algorithm $\mathcal{A}'$ breaking the semantic security of our encryption scheme. Given a (possibly invalid) public key, $\mathcal{A}'$ produces two messages m_0 and m_1. If the key was a valid public key, the probability of correctly guessing which message was encrypted should be exactly $50\% + \epsilon$ where ϵ is the non-negligible advantage that $\mathcal{A}'$ has. If $\mathcal{A}'$ is unknowingly using a key that is invalid, $\mathcal{A}'$ will guess one of the

messages at random. Using $\mathcal{A}'$, we can create an algorithm $\mathcal{A}$ that solves the Small RSA Subgroup Decision Problem. $\mathcal{A}$ is given a typical RSA quintuple (n, b, d, g, u), and an element $x \in \mathbb{Z}_n^*$ and constructs a public key (n, b, d, g, x, u) that it returns to $\mathcal{A}'$. Since x wasn't necessarily a valid element, the key that $\mathcal{A}$ sends to $\mathcal{A}'$ isn't necessarily valid. $\mathcal{A}'$ then produces two plaintexts m_0, m_1 and a random number sampled over the message space $r \leftarrow_{\$} \{1, 2, \ldots, 2^u - 1\}$. $\mathcal{A}'$ randomly chooses message m_i to send and creates a ciphertext $c \equiv g^{m_i} x^r \bmod n$. Looking at this, it is clear that breaking the Small RSA Subgroup Decision Problem is equivalent to figuring out the randomiser generator. For $\mathcal{A}$ to break our encryption scheme, it must be able to tell which message $\mathcal{A}'$ chose as i. If $\mathcal{A}'$ sends $\mathcal{A}$ a random $j \in \{0, 1\}$, we can construct the output of $\mathcal{A}$ as follows:

$$\begin{cases} \mathsf{yes} & \text{if } i = j, \\ \mathsf{no} & \text{otherwise.} \end{cases}$$

The ciphertext c is generated by x which was selected uniformly from the group $\mathbb{Z}_n^*$, this means that c should be independent of the choice i. If this property is upheld the probability of $\mathcal{A}'$ guessing correctly is equal to 50%.

On the other hand, as stated above, $r < 2^u$ and hence crucially $r < p_s, q_s$ which means that there is not uniform distribution over the entire message space. This results in a visible difference between the two messages. This gives $\mathcal{A}'$ a non negligible advantage, say ε, when x is an element of order $p_s q_s$, and this advantage is clearly seen to transfer to $\mathcal{A}$. ∎

3.6 Our Protocol

The initial purpose for the creation of our cryptosystem, was to enable a faster way for the secure comparison of integers. What this protocol allows is the comparison of two integer plaintexts m_1, m_2 in which a resultant of $\mathsf{Enc}(1)$ will occur whenever $m_1 > m_2$. Though this functionality is a threshold function, it is not binary and this leads to a leakage of information in the case where $m_1 \leq m_2$. The leakage in this situation is the difference between the two numbers. This property may have value in certain applications, however our goal for secure integer comparison was to return a binary resultant of *True* if $m_1 > m_2$ and *False* otherwise. For this property another round of encryption was necessary to hide the difference of the two numbers.

Because the desired functionality requires an additional layer of security, for the purpose of secure integer comparison we must use two separate cryptosystems within the

protocol. The first cryptosystem (Cr_1) is the cryptosystem described above that calculates the threshold function in Section 3.5.1. The second cryptosystem (Cr_2) is required to mask the difference between the two numbers, to stop either party from knowing the other's input. For this to work correctly, each party should have access to the private key of one of the cryptosystems. For example P_1 can hold the private key for Cr_1 and P_2 can hold the private key for Cr_2. It is important to note that the Cr_2 must be additively homomorphic and should have a message space of prime order correlating to the message space of Cr_1. The purpose of Cr_2 is to produce *a plaintext equality test* or PET to deduce if the resultant that is received is equal to the encryption of 1. This can be represented by:

$$c = \mathsf{PET}(\mathsf{Enc}'(a), b) = \begin{cases} \mathsf{Enc}'(1) & \text{if } a = b. \\ \mathsf{Enc}'(r) & \text{otherwise.} \end{cases}$$

In the above equation, $r \neq 1$ and $r \leftarrow_{\$} \{2 \dots \mathcal{M}'\}$. r is uniformly distributed across the message space $\mathcal{M}'$. To acquire uniform distribution within the PET, a randomized factor must be integrated. We use the homomorphic properties of Cr_2 to allow for uniform random distribution, in particular the additive and scalar multiplicative properties. Let the PET accept the ciphertext for an encryption $A = \mathsf{Enc}'(a)$, a plaintext b and a randomized value $\Phi \leftarrow_{\$} \mathcal{M}$. Return:

$$\mathsf{PET}(A/b)^r = \Big(\mathsf{Enc}'(a)/b\Big)^r = \mathsf{Enc}'\Big(r(a-b)\Big) = \begin{cases} \mathsf{Enc}'(1) & \text{if } a = b \\ \mathsf{Enc}'(r) & \text{otherwise} \end{cases}$$

The implementation of Cr_2 and the functions $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ can be done with any semantically secure additively homomorphic cryptosystem. When looking at the purpose of Cr_2, two prime examples of acceptable protocols are the exponential variant of ElGamal, and the elliptic curve implementation of ElGamal.

With both Cr_1 and Cr_2 in place we present our secure integer comparison protocol in Figure 3.1. A security proof for the protocol is provided in Chapter 4.

3.6.1 Correctness

For the result of our secure integer comparison, we look at the mathematical structure that lies within our encryption scheme. P_1 starts be sending their message within the exponent of the message space as b^{m_1}. Once P_2 receives the message, they homomorphically compute

$$w = b^{m_1} b^{d-m_2-1} + s = b^{d+m_1-m_2-1} + s.$$

Secure Comparison

Party $\mathbf{P}_1$		**Party $\mathbf{P}_2$**
Private Input:		**Private Input:**
$0 \leq m_1 < d-2$		$0 \leq m_2 < d-2$
Cr_1 private key x		Cr_2 private key sk'
$r_1 \leftarrow_{\$} \{1 \dots 2^u - 1\}$		
$c \leftarrow g^{b^{(m_1)}} h^{r_1}$		
	$\xrightarrow{\quad c \quad}$	
		$r_2 \leftarrow_{\$} \{1 \dots 2^u - 1\}$
		$s \leftarrow_{\$} \{0 \dots b^d - 1\}$ s.t. $s \not\equiv 0 \bmod b$
		$c' \leftarrow (c)^{b^{(d-m_2-1)}} g^s h^{r_2}$
		$D \leftarrow \mathsf{Enc}'(s)$
	$\xleftarrow{\quad c', D \quad}$	
$g^w \leftarrow (c')^x$		
$D' \leftarrow \mathsf{PET}(D, w)$		
	$\xrightarrow{\quad D' \quad}$	
		$z \leftarrow \mathsf{Dec}'_{sk'}(D')$
		Output *True* if $z = 1$
		Output *False* otherwise.

Figure 3.1: Secure integer comparison protocol evaluating $(m_1 > m_2)$

By choosing the exponent $d + m_1 - m_2 - 1$ we are able to set out threshold to the value of d and see how the comparison of m_1 and m_2 affect that threshold. In the case that $m_1 > m_2$, then $m_1 \geq m_2 + 1$, $m_1 - m_2 - 1 \geq 0$. Since we are adding this value to d and we know that the value is greater than 0, the resultant of the exponent for b will be greater than d. This means that the $b^{(\cdot)}$ term will be congruent to 0 mod b^d, and thus $w = s$. In the other case, if $m_1 \leq m_2$ the exponent for b will be lower than d so $w \neq s$. When P_1 decrypts the message and gets w, it performs a PET with the encryption of s to see if the two are equal. If the result is 1, then $w = s$ and P_2 outputs True because $m_1 > m_2$. When the result is not 1 P_2 outputs False because $m_1 \leq m_2$.

3.7 Extension Techniques

When evaluating an integer comparison using Figure 3.1 one of the main limitations is the range of $0 \ldots (d-2)$. If the message that is intended to be sent lies outside of the range provided by d, there are a few ways to alter the protocol to fit the new specifications. The most basic solution is to expand the parameters such that they can include the message within the message space. For this expansion, if a party wanted to send a message e the order of g can be changed from b^d to $b^{d'}$ where $d < e < d'$. Though this allows for the message to be sent within one encryption, there is a hindrance in performance because the public key has a size of $O(d)$ and the efficiency of solving the discrete logarithm of g is directly correlated to the size of d. Instead of increasing the size of a single message, an alternative solution is to break the message into multiple batch encryptions that can be done in parallel (dividing the input or 'blocking'). To accomplish this, the inputs can be represented with a base of base $(d-1)$. This comparison will be compromised of several parallel executions of the protocol of Figure 3.1.

To effectively use this approach we need to modify the final steps of the protocol so that it can accommodate multiple batches within the comparison. From the example above, consider the case where we want the comparison over the range of $0 \ldots (e-1)$ for some $e > (d-1)$. To accomplish this, we break down our integer of range e into k separate parallel encryption where $k = \lceil \log_{(d-1)}(e) \rceil$. We can then restructure our integers m_1, m_2 into piecewise encryptions with a base of $(d-1)$ as follows:

$$m_1 = \alpha_{k-1}(d-1)^{k-1} + \alpha_{k-2}(d-1)^{k-2} + \cdots + \alpha_1(d-1) + \alpha_0$$

and

$$m_2 = \beta_{k-1}(d-1)^{k-1} + \beta_{k-2}(d-1)^{k-2} + \cdots + \beta_1(d-1) + \beta_0$$

where $\alpha_i, \beta_i \in \{0 \dots d-2\}$.

From here we are able to turn the single comparison into multiple comparisons that produce a value of 1 if $\alpha_i > \beta_i$. Though this would seem as simple as looking to see if one of the comparisons produces a value of 1, we must remember that if the result of the first comparison says that $\alpha < \beta$, all subsequent results are void and not important. To conclude that $\alpha > \beta$ from a block that is not the initial block, you must also prove that all predecessors resulted in $\alpha = \beta$. The expansion of the following k Boolean expressions is as shown below, to prove that $m_1 > m_2$ exactly *one* of the expressions will be result in true:

$$(\alpha_{k-1} > \beta_{k-1})$$

or

$$(\alpha_{k-1} = \beta_{k-1}) \wedge (\alpha_{k-2} > \beta_{k-2})$$

or

$$(\alpha_{k-1} = \beta_{k-1}) \wedge (\alpha_{k-2} = \beta_{k-2}) \wedge (\alpha_{k-2} > \beta_{k-2})$$

or

$$\vdots$$

or

$$(\alpha_{k-1} = \beta_{k-1}) \wedge (\alpha_{k-2} = \beta_{k-2}) \wedge \cdots \wedge (\alpha_0 > \beta_0).$$

If $m_1 \leq m_2$, all of these expressions will be false. We can now apply this fact to securely evaluate $(m_1 > m_2)$ by essentially running k parallel instances of the protocol, and replacing individual plaintext equality tests with the Boolean tests listed above. After running the parallel computation messages are re randomized and shuffled before being sent back to the decrypting party. This extended protocol is given in Figure 3.2.

Secure Comparison

Party $\mathbf{P}_1$		**Party $\mathbf{P}_2$**
Private Input:		**Private Input:**
$\alpha_0 \dots \alpha_{k-1}$		$\beta_1 \dots \beta_{k-1}$
Cr_1 private key x		Cr_2 private key sk'
For $i \in \{0 \dots k-1\}$		
$c_i \leftarrow \mathsf{Enc}(b^{\alpha_i})$		
	$\xrightarrow{c_0 \dots c_{k-1}}$	
		For $i \in \{0 \dots k-1\}$
		$s_i \leftarrow_{\$} \{0 \dots b^d - 1\}$ s.t.
		$s_i \not\equiv 0 \bmod b$
		$c'_i \leftarrow (c_i)^{b^{(d-\beta_i-1)}} \mathsf{Enc}(s)$
		$D_{k-1} \leftarrow \mathsf{Enc}'(s_{k-1})$
		$\vdots$
		$D_0 \leftarrow \mathsf{Enc}'(\beta_{k-1} \| \beta_{k-2} \| \dots \| s_0)$
	$\xleftarrow{c'_0 \dots c'_{k-1}, D_0 \dots D_{k-1}}$	
For $i \in \{0 \dots k-1\}$		
$w_i \leftarrow \mathsf{Dec}(c'_i)$		
$D'_{k-1} \leftarrow \mathsf{PET}(D_{k-1}, w_{k-1})$		
$\vdots$		
$D_0 \leftarrow \mathsf{PET}(D_0, \alpha_{k-1} \| \alpha_{k-2} \| \dots \| w_0)$		
	$\xrightarrow{D'_1 \dots D'_{k-1}}$	
		If any $\mathsf{Dec}'_{sk'}(D'_i) = 1$
		Output *True*
		Output *False* otherwise.

Figure 3.2: Secure integer comparison protocol for large ranges using blocking

Chapter 4

Protocol Security

4.1 Simulation Security Basics

When discussing the security of a system, simulation can be used to describe a comparison of what happens in the "real world" and what happens in an "ideal world". The concepts of "real" vs. "ideal" are key in the understanding of the proofs that we claim in this section. First, one can describe the "real world" as the world in which two parties are comparing information through the use of our protocol. To compute the overall comparison of the integer, messages will flow to and from each party and together they learn the final output of the protocol. In contrast, the "ideal world" scenario that one can look at is like connecting with an all knowing oracle instead of another party. This would be similar to being able to do the calculations with just one party. With these two terms explained, the overall definition of simulation security becomes a lot more clear. We will flesh out the "real" and "ideal" scenarios below, where the inputs are (x, y) and the security parameter is 1^k:

Real: A protocol can be broken into the computations that each side does throughout the protocol. With this we can split protocol π into two parts: $\pi = \pi_A, \pi_B$. Let's say that for this instance, we are P_B who uses π_B to interact with P_A. When π_B is finished, there will be an output that it can see, which will be called c. On the other end of the protocol π_A will also result in an output d. The output for the "real world" scenario is the pair (c, d) that is generated. **Ideal:** Use a simulator S to come up with an input x that fits within the security parameters (1^k) for the protocol. Calculate the ideal functionality of $c = f_A(x, y)$ and $d = f_B(x, y)$. Where c and d are the final outputs of the simulation. The output for the "ideal world" scenario is the pair (c, d) that is generated.

If these two results are impossible to tell from one another, we have a simulation

secure protocol.

Definition Simulation Security: *A protocol can be described as simulation secure if For all PPT machines A there exists a PPT machine S such that for all x, y the ensembles $\{Real(\pi, y, A, 1^k)\}_k$ and $\{Ideal(f, y, S, 1^k)\}_k$ are computationally indistinguishable [3].*

The notion of this security definition is that a party in both atmospheres would learn a similar amount of information. By looking at the ideal world scenario it is clear that it is impossible to learn anything from the input, because the input was just randomly made up. This is the purpose of the simulation security proof; since someone who receives nothing cannot learn anything about the plaintext (which is the "ideal world" perspective), a scenario where the adversary receives a ciphertext should only be able to determine the same amount. The ciphertexts should not give any hints as to what the plaintexts beneath hold. We can say that an encryption scheme is secure if all of the information that is known at the end can be directly determined through prior knowledge. The protocol does not leak information.

To create the simulations for the following sections, there are three key components (Described in Lindell's solo work [41], as well as his work with Katz [37]) that a simulator must be able to accomplish:

1. Generate views for the other party that are indistinguishable from the real view;
2. Extract the inputs used by the adversary during the execution
3. Make the view generated consistent with the output that is based on the adversary's input

4.2 Our Simulation Environment

Our setting of private sorting requires a function F_{sort} that runs between P_A and P_B with their private integers x and y. In this function P_A has the input (x, b, d, k, g, u) and uses P_B (that knows s_k and y) to output the resultant of a comparison between x and y, the two encrypted values to be sorted. The output of the function is a binary value, where a message with 1 means that value x is larger than value y and any other message states the opposite. The ideal world model of the protocol is shown in Figure 4.1. In this model, the simulator interacts with the party through the ideal functionality of the system, and uses this functionality to replicate what the protocol would do for the other party . In Figure 4.1, the simulator interacts with the ideal functionality through protocol Π_{sort}, which makes the output of the function look like the real world protocol to party P_A.

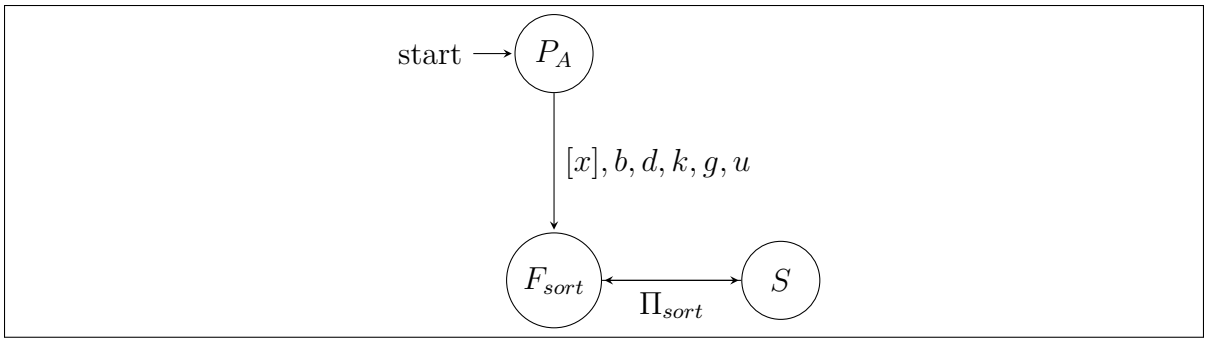

Figure 4.1: Ideal functionality of our protocol for P_A

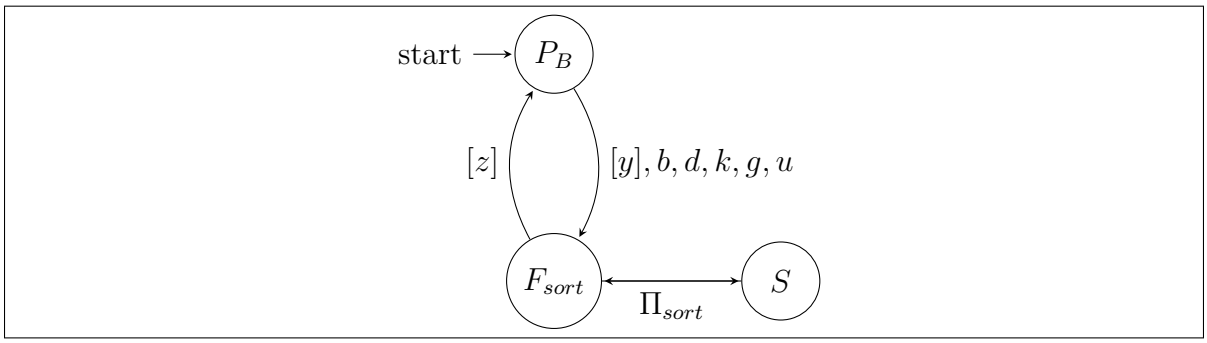

Figure 4.2: Ideal functionality of our protocol for P_B

Similarly, there exists an ideal world scenario for P_B in which it interacts with the ideal functionality and a simulator and receives the output of the protocol z. In Figure 4.2 there is a template for what the ideal functionality looks like for party P_B.

4.3 Simulation Proof

For the proof of our security model, we look at the security of the protocol with respect to a semi-honest (passive) adversary in a two-party setting. This style of adversary goes along with the correct path of the protocol, but tries to learn additional information based on the transcript and messages exchanged throughout the protocol. As the title of this chapter suggests, the proof that will be demonstrated is a simulation based proof. This type of proof gives security under sequential composition, but not concurrent composition (like the proof discussed by Damgaard in [20]). To prove this we are going to show that the manifest created by the protocol is computationally indistinguishable from a simulated view of the protocol that is using randomized inputs [51]. This would suggest that no information is leaked through the protocol itself. Next we define the semi-honest notion

of simulation security:

Parties P_A and P_B interact in a protocol Π which computes the function of the protocol given the expected inputs and produces the expected outputs. Let F be a function defining the ideal functionality of the protocol Π, taking a pair of inputs $(\mathsf{in}_A, \mathsf{in}_B)$ to a pair of outputs $(\mathsf{out}_A, \mathsf{out}_B)$. The *view* of participant P_i (where $i = A, B$) will be denoted by $\mathbf{VIEW}^{\Pi}_{P_i}(\mathsf{in}_A, \mathsf{in}_B)$ and is defined as the information P_i observes and produces throughout the protocol. Let Sim_i be a simulator that takes in the inputs of party P_i and the ideal functionality of the protocol F and produces a transcript of the protocol. With this setup, we now give the definition of simulation security of a protocol.

Definition Secure Against Passive Adversaries: A protocol Π is *secure against passive adversaries* from the point of view of P_i (for $i = A, B$) if a probabilistic polynomial time simulator Sim_i exists for each party such that $\mathsf{Sim}_i(\mathsf{in}_i, F(\mathsf{in}_A, \mathsf{in}_B))$ is computationally indistinguishable from $(\mathbf{VIEW}^{\Pi}_{P_i}(\mathsf{in}_A, \mathsf{in}_B), \mathsf{out}_i)$. Recall the discussion of computational indistinguishably in Section 2.2.5.

In a two party setting, we can say that the protocol Π is *secure against passive adversaries* if it is secure from both the point of view of P_A and the point of view of P_B.

For the remainder of this section we prove that the comparison protocol of Figure 3.1, which we will reference as Π, is secure against passive adversaries. To do this we prove the security through the eyes of P_A and through P_B.

In our case, the ideal functionality F is a function with the inputs (m_1, m_2) and output α (a binary indicator which results in *True* if $m_1 > m_2$ and *False* otherwise. It is clear that F defines the functionality of the protocol Π. When Π terminates, P_B receives output of F. Let $\mathbf{OUTPUT}^{\Pi}(m_1, m_2)$ be the output received by P_B.

Lemma 4.3.1 *The protocol* Π *is secure against passive adversaries from the point of view of* P_A*.*

Proof In order to show that P_A does not learn anything about m_2 we will construct a valid simulator Sim_2 for P_B with the property that

$$\mathsf{Sim}_2(m_2, F(m_1, m_2)) \stackrel{c}{\equiv} \mathbf{VIEW}^{\Pi}_{P_i}(m_1, m_2).$$

Here, we write $\stackrel{c}{\equiv}$ for the relation of computational indistinguishability. The simulator Sim_2 is given the input values for P_B and is able to simulate P_B's view by sampling random values r, s and C' as shown in Figure 4.3. The use of semantically secure encryption for message D prevents party P_A from being able to distinguish between real and simulated

plaintext. From this stage, we have to prove that the remainder of the messages are indistinguishable from messages that would be sent in a simulation setting. Let us first look at the value r that is generated by P_B. To do this, we must define what the values of r can be. The first random variable chosen by Sim_2 is the r value for the message space; this should be randomly sampled through the entirety of the message space with an equal probability for each integer in the set $\mathbb{Z}_{b^d}$. Next we need to find the random variable used in the randomiser space. Let $\mathcal{R} \subset \mathbb{Z}_{b^d}$ denote the set of all values r for which $r \not\equiv 0 \bmod b$. This restriction is necessary because we are working alongside a group of prime power order b, anything congruent to 0 mod b within the randomiser space would be cancelled out. Let $s \leftarrow_{\$} \mathcal{R}$. When P_A receives the message C' it is able to decrypt to find a plaintext v but, because of the structures of the sets in which our randomisers were picked from P_A is unable to tell if the message was formed by the process of $v = b^{d+m_1-m_2-1} + s$ or if a simulator was used and the value was $v = s$. The results for both of these processes should be uniform in their distribution over the set of $\mathcal{R}$, which we just defined for the simulator space.

Next we must prove that using the function $v = b^{d+m_1-m_2-1} + s$ not only limits results to answers within the set of $\mathcal{R}$, but also results in a uniform distribution across the set of $\mathcal{R}$. The key to the first part of this proof is that,

$$(b^{d+m_1-m_2-1} + s) \bmod b^d \in \mathcal{R} \text{ if } (b^{d+m_1-m_2-1} + s \bmod b^d) \bmod b \neq 0.$$

Looking at the second half of this equation, we can see that there is a duplicate modulus when dealing with the s value (it uses mod$b$ both inside and outside the brackets). By reducing this we get

$$(b^{d+m_1-m_2-1} + s) \bmod b \neq 0.$$

This can be further reduced, since the first section is working in base b and is under the modulus of b. This reduction leaves us with $s \bmod b \neq 0$ which is part of the definition of s. This proves that $(b^{d+m_1-m_2-1} + s) \in \mathcal{R}$. Lastly we look at the distribution of this function over the set of $\mathcal{R}$. Since we know that s is uniform over the set of $\mathcal{R}$ and it is the resultant of the reduction mod$b$, $b^{d+m_1-m_2-1} + s$ is uniform as well. Seeing that both the real world and ideal world scenarios result in answers that are computationally indistinguishable, the system is secure against passive adversaries from the point of P_A. ∎

Lemma 4.3.2 *The protocol* Π *is secure against passive adversaries from the point of view of* P_B.

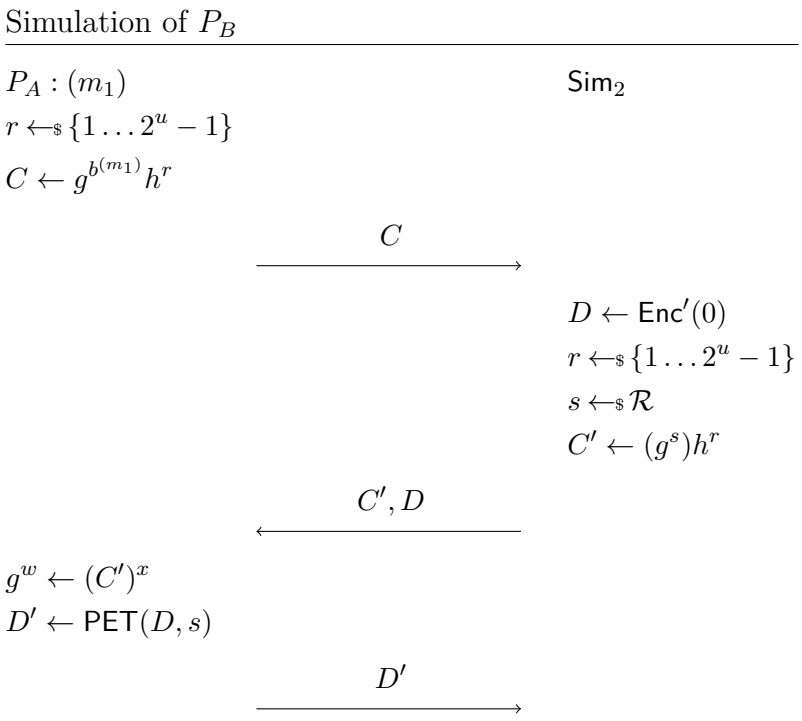

Figure 4.3: Simulation of P_B

Proof Now we construct a simulator Sim_1 with the property that

$$\mathsf{Sim}_1(m_1, F(m_1, m_2)) \stackrel{c}{\equiv} (\mathbf{VIEW}^{\Pi}_{P_i}(m_1, m_2), \mathbf{OUTPUT}^{\Pi}(m_1, m_2)).$$

This simulator is presented in Figure 4.4. Since P_B learns the output of the protocol, in order for Sim_1 to produce a view indistinguishable from a real execution, we must provide it access to input messages m_1 and m_2 so that it can correctly simulate the result anticipated by P_B. The term *anticipated* is used here because the distribution of the resultant (i.e. if $(m_1 > m_2)$) relies on the knowledge of m_2 and where it lies within the field of acceptable inputs. An example of this is the case in which $m_2 = d - 2$, in this case, m_2 is the largest possible number to be compared, as such P_B would expect to output *False* for all possible values of m_1. The use of this information is only in the final message D' that is sent to P_B to show the end result. With the knowledge of m_2, Sim_1 does not need to know the private key for decryption as it just encrypts values based on what the value of m_2 is. The two messages C and D' that are sent from Sim_1 to P_B must share the same uniformity as the real world for Sim_1 to be computationally indistinguishable from a real world party (P_A). First, we look at the message C that is sent as the initial message. Since Sim_1 chooses a random value $C \in \mathbb{Z}^*_n$, and P_B does not have the secret key for the

encryption, C is indistinguishable from a valid ciphertext from the view of P_B. This is an example of the Small RSA Subgroup Decision Assumption presented in Section 3.3. P_B then homomorphically applies its message share and returns C', D to Sim_1.

From here, Sim_1 looks at the probability of the resultant of $(m_1 > m_2)$, knowing the input of m_2 and the likelihood that a random number is larger or smaller than m_2. Sim_1 provides P_B with the encryption of 1 if $(m_1 > m_2)$. Otherwise it provides P_B the encryption of a uniform value in the message space, $\mathcal{M}_{\mathsf{Enc}'}$ of Enc'. This makes P_B's view indistinguishable from a real-world execution of the protocol. ∎

Simulation of P_A

Sim_1		$P_B : (m_2)$
$C \leftarrow_{\$} \mathbb{Z}_n^*$		
	$\xrightarrow{\quad C \quad}$	
		$r \leftarrow_{\$} \{1 \dots 2^u - 1\}$
		$s \leftarrow_{\$} \{1 \dots b^d - 1\}$ s.t $s \not\equiv 0 \bmod b$
		$C' \leftarrow (C)^{b^{d-m_2-1}} g^s h^r$
		$D \leftarrow \mathsf{Enc}'(g^s)$
	$\xleftarrow{\quad C', D \quad}$	
If $(m_1 > m_2)$		
$D' \leftarrow \mathsf{Enc}'(1)$		
Else		
$r \leftarrow \mathcal{M}_{\mathsf{Enc}'}$		
$D' \leftarrow \mathsf{Enc}'(s)$		
	$\xrightarrow{\quad D' \quad}$	
		$z \leftarrow \mathsf{Dec}'_{sk'}(D')$
		Output *True* if $z = 1$
		Output *False* otherwise.

Figure 4.4: Simulation of P_A

Putting together Theorems 4.3.1 and 4.3.2, we obtain:

Theorem 4.3.3 *Protocol* Π *is secure against passive adversaries.* ∎

Chapter 5

Implementation

Within this chapter we talk about an implementation of the protocol (from Figure 3.1) that was done to evaluate the performance of our protocol vs. other protocols that are currently used in practice. We also look into the structure of our message space vs. the structure of the DGK message space and discuss the benefits of each structure. Working in a mathematical field, the approach for the implementation of the protocol was mainly done using functional programming. Separate mathematical functions such as the encryption and decryption functions were isolated from the main program and called as functions. This allowed for clear concise code when viewing the functions that relate directly to the manipulation of the plaintext/ciphertext in the protocol functions. An example of this would be the function that is called for the encryption of the first message c in Figure 3.1. With the ability to pre-process the generation of the generators and the finite field that the protocol is working in, the function itself is just raising the base to the appropriate power and calling the encryption function on that specific message.

5.1 Environment

When looking into different possible environments for the implementation of this protocol, there were many factors that had to be considered. First and foremost, the implementation had to be done in a language that allowed for complex mathematical functions to be calculated efficiently. These calculations were mainly using the cyclic algebraic group structures mentioned in Section 2.2.5. When looking to find environments that would work well with our specifications, we also considered ease of use and speed for writing up the protocol. With one of our focusses being ease of implementation, it felt natural to lean towards a strain of python as it is very fast to program and there are a lot of packages that deal with mathematics [49].

In the end the protocol was implemented in SageMath which is an open source library for python that works by combining the works of SciPy, NumPy, SymPy as well as many more mathematical packages all into a single working environment [58]. This string of python is typically used for research purposes and it included all of the relevant mathematical functions that were needed for our protocol and encryption scheme (which was presented in similar works [42]). The particular environment used was sage-windows, which is a windows based form of sage that runs the terminal through a notebook run on a local server. This structure allowed work to be done on many separate computers, and was efficient when implemented with a repository.

5.2 Coding Concepts

There were a few important concepts with regards to the construction of the cryptosystem that were studied before going through with the implementation of the system. The first of which, was the key generation of the system. When generating the parameters for the public key, it was important to consider the fact that p and q were both constructed using the message space, the randomiser space and the padding space. To do this unique prime numbers had to be found for each space, and specifically the pseudoprimes for the message space needed to be relatively prime to the randomiser primes, in that they share no similar factors. This was important as sharing a factor with the randomiser space would severely weaken the encryption.

Another concept that was important when implementing was using exponentiation for solving the decryption. When decrypting the message, one would receive it in the form $g^m h^r$. We can get rid of the h^r using the private key, but afterwords we need to find the decimal number for an exponential binary number. A simple visual of the concept is shown below:

$$c = g^{110110} \rightarrow c = 54$$

We need to be able to extract the bits from the exponent to get the overall value. To do this, we use squarings, such that we can look at every binary digit one by one and evaluate based on prior knowledge. For this we square the number until we fill all but the first value with 0's. Then we determine that digit and fill all but the first two values with 0's, etc. until all digits are found. Section 5.2 explains how this would be structured using a 4 digit exponent on the input 1101.

Squaring	Breakdown of g $g = g^{x_3x_2x_1x_0}$	Subtracting negatives	Result
$((y^2)^2)^2$	$g^{x_3x_2x_1x_0000} = g^{x_0000}$	$g = g^{x_0000}$	$x_0 = 0$ *if* $g = 1$ *else* $x_0 = 1$
$(y^2)^2$	$g^{x_3x_2x_1x_000} = g^{x_1x_000}$	$g = g^{x_1x_000} \cdot g^{-0x_000}$	$x_1 = 0$ *if* $g = 1$ *else* $x_1 = 1$
y^2	$g^{x_3x_2x_1x_00} = g^{x_2x_1x_00}$	$g = g^{x_2x_1x_00} \cdot g^{-0x_100} \cdot g^{-00x_00}$	$x_2 = 0$ *if* $g = 1$ *else* $x_2 = 1$
y	$g^{x_3x_2x_1x_0} = g^{x_3x_2x_1x_0}$	$g = g^{x_3x_2x_1x_0} \cdot g^{-0x_200} \cdot g^{-00x_10} \cdot g^{-000x_0}$	$x_3 = 0$ *if* $g = 1$ *else* $x_3 = 1$

Table 5.1: Table for extracting bits using squarings

5.3 Algorithms Used

This section is used to present some of the algorithms that were used when implementing the system using SAGE. The algorithms shown in this section are broken into two main groups: algorithms for key generation and algorithms for encryption and decryption. For key generation we will show the creation of all of the derived parameters for the public key (n,g, and h) and private key (x). We will not provide the algorithms used for the protocol itself as they should be interpretable based on the diagrams from Section 3.6.

Algorithm 2 Creation of p from Section 3.5

```
1: procedure findP(u)
2:     while true do
3:         a ← randint(2^(u−1), 2^u)
4:         if a is a prime then
5:             return a
6:         end if
7:     end while
8: end procedure
```

Algorithm 4 Creation of n from Section 3.5

```
1: procedure MakeN(b, d, u)
2:     p_t ← findP(1535)          ▷ 2048 bit message -257 bits for g -256 bits for h
3:     q_t ← findP(1535)
4:     while true do
5:         p_s ← findP(u)
6:         p ← b^d p_s p_t + 1
7:         if p is a prime then
8:             break
9:         end if
10:    end while
11:    while true do
12:        q_s ← findP(u)
13:        q ← b^d q_s q_t + 1
14:        if q is a prime then
15:            break
16:        end if
17:    end while
18:    n ← pq
19:    return n
20: end procedure
```

Algorithm 6 Creation of g from Section 3.4

```
1: procedure MAKEG(b, d, p, q)
2:     while true do
3:         x ← randint(2, p − 2)
4:         y ← pow(x, (p − 1)/b, p)
5:         if y ≠ 1 then
6:             g_p ← pow(x, (p − 1)/(b^d), p
7:             break
8:         end if
9:     end while
10:    while true do
11:        c ← randint(2, q − 2)
12:        d ← pow(c, (q − 1)/b, q)
13:        if d ≠ 1 then
14:            g_q ← pow(c, (q − 1)/(b^d), q
15:            break
16:        end if
17:    end while
18:    g ← crt([g_p, g_q], [p, q])          ▷ Chinese Remainder Theorem
19:    return g
20: end procedure
```

Algorithm 8 Creation of h from Section 3.5

```
1: procedure MAKEH(p_s, q_s, p, q)
2:     while true do
3:         m ← pow(randint(2, p − 2), (p − 1)/p_s, p)
4:         if m ≠ 1 then
5:             h_p ← m
6:             break
7:         end if
8:     end while
9:     while true do
10:        n ← pow(randint(2, q − 2), (q − 1)/q_s, q)
11:        if n ≠ 1 then
12:            h_q ← n
13:            break
14:        end if
15:    end while
16:    g ← crt([h_p, h_q], [p, q])
17:    return h
18: end procedure
```

Algorithm 10 Encryption from Section 3.5

```
1: procedure ENCRYPT(m)
2:     r ← randint(2, 2^u − 1)
3:     c ← (pow(g, m, n) ∗ pow(h, r, n)) mod n
4:     return c
5: end procedure
```

Algorithm 12 Decryption from Section 3.5

```
1: procedure DECRYPT(c)
2:     table ← {}
3:     i ← 0
4:     for i < 257 do                          ▷ Creates table of squarings for comparison
5:         table[i] ← c
6:         c ← pow(2, c, n)
7:         i ← i + 1
8:     end for
9:     ntrack ← deque(257)                     ▷ Tracks 1's to subtract
10:    answer ← deque(257)                     ▷ Holds bits for answer
11:    for i in range (256, −1, −1) do
12:        for j in range len(ntrack) do
13:            ntrack[j] ← ntrack[j] − 1
14:            table[i] ← table[i] · negs[ntrack[j]]   ▷ negs is a table of negative squarings
   of g
15:        end for
16:        if table[i] ≠ 1 then
17:            ntrack ← 256
18:            answer ← 1
19:        else
20:            answer ← 0
21:        end if
22:    end for
23:    result = int(''.join(map(str, answer)), 2)
24:    return result
25: end procedure
```

5.4 Analysis

Within this section, we do a comparison of the performance of our protocol in Figure 3.1 against the secure integer comparison DGK protocol of Damgård, Geisler, and Krøigaard (DGK) [16, 18]. The primary difference between the two being the use of homomorphic operations on bitwise encrypted values, versus encoding the entire value in a single ciphertext. Here we use the term 'cost' with respect to the computational complexity.

The basic form of the encryption is standard across both schemes

$$\mathsf{Enc}(m) = g^m h^r \bmod n.$$

Encryption and Re-randomization Cost.

Although the message space of the DGK cryptosystem has a small prime order, while our message space has a large prime power order, the encryption operations are similar. The main cost in both of these encryptions is the computation of the random factor $h^r \bmod n$, which is sized equivalently in both schemes ($p_s q_s$ in ours, $v_p v_k$ in DGK). Both schemes allow g^m to be precomputed and combined with the randomiser in one modular multiplication, indicating that the encryption of a single ciphertext takes the same amount of time across both schemes. Re-randomization is just the homomorphic addition of 0 which also takes an identical amount of time for both of the schemes.

Decryption Cost. Damgård et al. [18] point out that decryption in their scheme can be performed in a short exponentiation modulo p which is more efficient than using modulo $n = pq$:

$$C^{v_p} = g_p^{m v_p} h^{v_p} = g^{m v_p}.$$

In the DGK protocol, decryption is just used to check if the plaintext is 0. In this case the v_p factor in the exponent of g does not need to be removed. In our encryption scheme we begin by eliminating the random factor identically as in DGK:

$$C^{p_s} = g_p^{m p_s} h^{p_s} = g^{m p_s}.$$

In Figure 3.1 we used the factor x to eliminate the p_s term in the exponent. This can be done more efficiently by computing the discrete logarithm to recover (mp_s), then computing $(mp_s)(p_s^{-1}) \bmod b^d$. Taking the discrete log is efficient for a small base such as $b = 2$, and can be optimized to the cost of about one d-bit modular exponentiation.

Cost of P_2's homomorphic addition. When P_2 receives ciphertext C, it must compute $C^{b^{d-m_2-1}}$ which costs up to d squarings.

Cost of Enc'. Our scheme uses an additional encryption scheme Enc' for the computation of a plaintext equality test. Here we can use an elliptic curve encryption scheme, such ElGamal implemented over an elliptic curve group. The cost of this, however, is marginal relative to the cost of a modular exponentiation in an RSA subgroup. Our experiments

with assembly optimized elliptic curve implementations produced modular exponentiations in around 10μs, whereas an equivalent modular exponentiation in the RSA setting was in around 2ms.

Parameterizations. For the message space, we note the optimal choice for b in terms of key length is 2. In our evaluation we will compare 8-bit messages, i.e., messages in the range $0 \ldots 255$. This implies $d = 255 + 2 = 257$. For the cryptographic parameters we adhere to current NIST[1] minimum recommended guidelines on key lengths require a 3072-bit factoring modulus, and a 256-bit discrete logarithm group.

We note Groth [32] conjectured that since the order of the randomizer space of his cryptosystem is hidden, for performance reasons it may be possible to safely parameterize it to a size smaller than what would typically be required to make the discrete logarithm hard. Coron et al. [14] nonetheless found an attack on this approach essentially in $O(\sqrt{p_s})$ time and $O(\sqrt{p_s})$ space. Although the $O(\sqrt{p_s})$ space requirement makes the attack strictly worse than generic methods for solving a discrete logarithm (and in fact a significant real-world implementation challenge), we argue it would be inadvisable to go below minimum recommendations on discrete logarithm groups sizes. We parameterize the bit length u of p_s and q_s (and corresponding DGK randomizer space) accordingly.

Working at the 128-bit security level, this implies a parameterization of our cryptosystem as follows: $|n| = 3072$, $|p|, |q| = 1536$, $u = |p_s|, |q_s| = 256$, and $|p_t|, |q_t| = 1536 - 256 - \lceil \log_b(b^d) \rceil = 1280 - 257 = p$. For the implementation of Enc' we use elliptic-curve ElGamal over the standard NIST curve `secp256r1`.[2]

For the DGK implementation we use the analogous parameterizations. Using the notation of [18] we set $|n| = 3072$, $|p|, |q| = 1536$, randomizer space $|v_p|, |v_q| = 256$, and message space of order $u = 11$, which is the next largest prime up from $\log_2 256$.

At the 256-bit security level we require a 15360-bit factoring modulus and 512-bit discrete logarithm group. This implies $|n| = 15360$, $|p|, |q| = 7680$, $|p_s|, |q_s| = 512$. We also use elliptic-curve ElGamal over curve `secp521r1`.

Performance comparison. The goal of the implementation was to provide a basis for comparison between the two protocols, and as such the metric of interest is the relative (as opposed to absolute) running times. In each case we made an effort to use optimizations (such as fixed-base exponentiations, working mod$p$ instead of mod$n$, etc.) where possible.

[1] `https://www.keylength.com/en/4/`
[2] `http://www.secg.org/SEC2-Ver-1.0.pdf`

As a simplifying assumption we did not factor in the cost of network transmission, though it would only impact the performance in our favour given the significant difference in the total communication cost.

At the 128-bit security level the computation time of our protocol was approximately 4.3 times faster than DGK. Our protocol transmitted approximately 7 times less data (896 bytes compared to DGK's $6,144$ bytes). At the 256-bit security level the gap widens slightly. Our protocol was 5.1 times faster in computation than DGK and transmitted approximately 7.5 times less data ($4,096$ bytes compared to DGK's $30,720$ bytes). This can be seen in Section 5.4.

With this expanding difference in terms of computation time and data transmission, we can say that for any comparison using a set of integers that are larger than two binary digits in length, our protocol outperforms the standard DGK protocol and would be more efficient in everyday use.

	128 Bit Security		256 Bit Security	
	Computation Time	Data Transmitted	Computation Time	Data Transmitted
DGK Protocol	1x	6144 bytes	1x	30720 bytes
Our Protocol	0.23256x	896 bytes	0.19608x	4096 bytes
Difference	0.76744x	5248 bytes	0.80392x	26624 bytes

Table 5.2: Comparison of DGK protocol performance vs. our performance

Chapter 6

Extension to Geo-spatial Analysis

6.1 Overview

During the initial first months of the thesis, research focussed on the field of geographic information systems (GIS). The initial direction of the thesis was to create a protocol for the geo-spatial analysis of an area and find points of interest that were closest to a given GPS location. Even though this geo-spatial protocol was not worked on after the pivot to secure integer comparison, it is a prime example for where the our protocol would be able to fit into current cryptographic practice. This type of protocol would be useful in a wide array of applications, from helping dispatchers find the closest emergency services for an incident to having corporations find prime locales for specific businesses [35]. This field was very interesting with regards to cryptography as the locations sent and compared should not be known to both parties. The complexity of this field sparked an interest in creating a semantically secure protocol for geo-spatial analysis.

The field of geo-spatial analysis is the heart of GIS (Geographic Information Systems). It acts as a bridge between the raw information that is stored on a device and the many relationships that that data can have with the world around it [21]. For an application to work in this type of field it would have to grapple with a large number of comparisons quickly and would have to have a sorting structure for the locations that were being compared. When examining the field, we were unable to find a tool that we felt was adequate for comparing the encrypted values of such a large dataset while maintaining the security and privacy of the data itself.

Initially we planned to use a protocol to make a set of comparisons with a list of locations and find out which location was closest to the GPS coordinate that was sent. To accomplish this, we had to break the over-arching protocol into three separate stages and find the most effective means for each of these three stages. The three stages that we

broke the comparisons down into were:

1. **Distance Calculations:** The distances from all data sources to the current position were found. For this a Euclidean distance calculation under encryption was needed between the current location and a set location from a pre-recorded list of locations.
2. **Homomorphic Shuffling:** This was used to hide the identities of each of the distances that were compared, meaning that the first location would appear at a different location within the list of distances every time. This stage helped with the secrecy of the locations so that are unknown within the next stage of the protocol.
3. ***k*-Nearest Neighbour Computation:** Sort the list of all distances and list the closest k locations to the current GPS location. This is the bulk of the computation for the protocol as there needs to be a net of comparisons for the proper ordering of the locations to be found.

While attempting to find an ideal process for the "k-nearest neighbour" computation, we discovered that the current comparison protocols in practice were not sufficient for what we wanted for the protocol. The number of encryptions for the process of bitwise decomposition was not good enough (in terms of computational time and cost) in my opinion. It was at this time that the thesis was pivoted to find a faster secure form of integer comparison so that the time spent within the sorting process would be significantly reduced. Even though this geo-spatial protocol was not worked on after the pivot to secure integer comparison, it is a prime example for where the our protocol would be able to fit into current cryptographic practice.

6.2 Math Basics

6.2.1 Euclidean Distance

In geography, the Euclidean distance is the direct distance between two points in a certain number of dimensions (in our case we are looking at two dimensions). First we need to define the location of a point. Let $p = \{x, y\}$, where p is the representation of a point we are using in our distance calculation. In an unencrypted setting, the way to calculate the distance between points a and b would be to use the following formula:

$$D(p_a, p_b) = (x_a - x_b)^2 + (y_a - y_b)^2$$

This approach finds the overall distance of the unencrypted data. For the encrypted approach we will need to compute the expanded version of this equation. It looks as

follows:

$$D(p_a, p_b) = x_a^2 + x_b^2 - 2x_a x_b - 2y_a y_b + y_a^2 + y_b^2$$

6.3 Geo-spatial Protocol

6.3.1 Distance Calculation

By calculating the Euclidean distance under encryption, we need to modify the Euclidean distance equation to fit the homomorphic properties of the encryption scheme that we are using. For this, we must use different operations which translate into the squaring, subtraction and addition used in the plaintext space. This approach is shown in Figure 6.2

6.3.2 Shuffling

This aspect of the protocol focuses on shuffling the order of the list of distances that was created in the first part of the protocol. It allows the comparison to be blinded, so that the result of the comparison is only known to P_A once the reverse shuffling has been done. This is a complex procedure, and because the initial idea for the thesis pivoted, it was not researched in detail. However, it is important to include in the structure of the protocol because it is vital in hiding the resultant from P_B.

6.3.3 Comparison Protocol

For the comparison section of the protocol, we will need to be doing comparisons on the size of the distances that have been calculated in Section 6.3.1. In these comparisons, one party has both of the encrypted inputs and the other has the secret key for decrypting the inputs. In our protocol (described in Section 3.6), it can be seen that each party holds an input for the comparison. This difference means that to use our protocol in this geo-spatial protocol, we will need to extend the protocol so that both parties have one input, and they compute the comparison off those inputs.

We accomplish this by doing the subtraction of the integers under encryption and then blinding the result using a randomized number. By working in a cyclic group of size n, we can abuse the fact that adding two numbers r, x such that $r, x > n/2$ will result in a number within the first half of the total space covered by n. This property can also apply to subtraction wherein subtracting a large number in n from a small number in n results in a number that is in the second half of the space covered by n. This results in a protocol similar to the presented by Baldimtsi et al. [2]. Below in Figure 6.3 is a set of images

Euclidean Distance Calculation

Party $\mathbf{P}_A$		**Party $\mathbf{P}_B$**
Private Input:		**Private Input:**
x_a, y_a, x_b, y_b (coordinates of locations x,y)		private keysk
$r, s, t, u \leftarrow_{\$} \{1 \dots 2^{30}\}$		
$p = \langle [[x_a]]^r, [[y_a]]^s \rangle$		
$p = \langle [[x_a r]], [[y_a s]] \rangle$		
$q = \langle [[x_b]]^t, [[y_b]]^u \rangle$		
$q = \langle [[x_b t]], [[y_b u]] \rangle$		
	$\xrightarrow{p, q}$	
		$\langle x_a r, y_a s \rangle \rightarrow \langle (x_a r)^2, (y_a s)^2 \rangle$
		$\langle x_b t, y_b u \rangle \rightarrow \langle (x_b t)^2, (y_b u)^2 \rangle$
		$p' = \langle [[x_a^2 r^2]], [[y_a^2 s^2]] \rangle$
		$q' = \langle [[x_b^2 t^2]], [[y_b^2 u^2]] \rangle$
		$i = [[-2x_a x_b rt]]$
		$j = [[-2y_a y_b su]]$
	$\xleftarrow{p', q', i, j}$	
$p'' = \langle [[x_a^2 r^2]]^{r^{-2}}, [[y_a^2 s^2]]^{s^{-2}} \rangle = \langle [[x_a^2]], [[y_a^2]] \rangle$		
$q'' = \langle [[x_b^2 t^2]]^{t^{-2}}, [[y_b^2 u^2]]^{u^{-2}} \rangle = \langle [[x_b^2]], [[y_b^2]] \rangle$		
$i' = i^{r^{-1} t^{-1}} = [[-2x_a x_b]]$		
$j' = j^{s^{-1} u^{-1}} = [[-2y_a y_b]]$		
$D = [[x_a]]^2 [[x_b]]^2 [[-2x_a x_b]] [[-2y_a y_b]] [[y_a]]^2 [[y_b]]^2$		
$D = [[x_a^2 + x_b^2 - 2x_a x_b - 2y_a y_b + y_a^2 + y_b^2]]$		
Output D		

Figure 6.1: Euclidean distance sub protocol

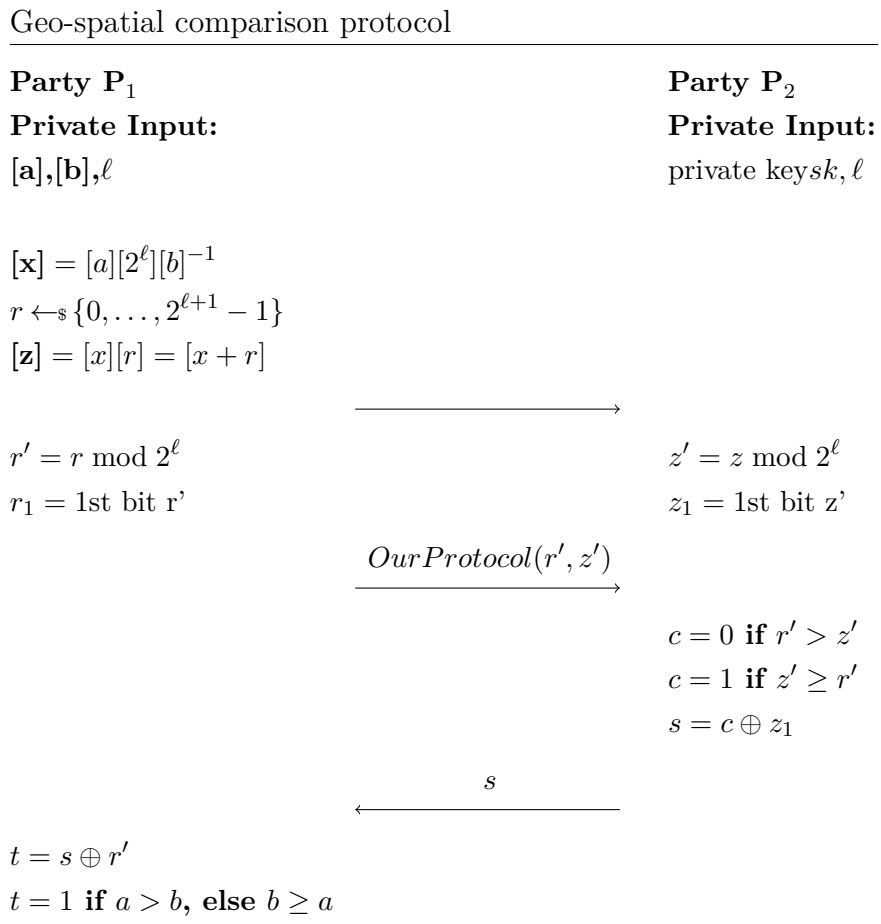

Figure 6.2: Geo-spatial comparison protocol

that can help with the understanding of the properties that are being discussed. By using these properties we are able to tell which integer is larger by looking on which half of n the resultant lies in. The correctness of this is described in more detail in Section 6.3.5.

6.3.4 Privacy Proof

Privacy in our setting means that throughout the process of the protocol Party A only learns the result of the comparison and Party B does not learn anything new throughout the transaction. To prove this, we can look into situations where either Party A is corrupt or where Party B is corrupt.

In the first case, let us assume that Party A is corrupt and is trying to gain information from Party B. Since both of the inputs for the comparison are committed by Party A, the only information that they do not currently possess and which they can retrieve is the decryption of the inputs that they have and the secret key itself. Let us say that instead

of picking a random number from the generation in the protocol, Party A chooses 0, so that they have a better chance at seeing the difference between a and b. In the protocol, the value c' is used to decipher the bigger number in the comparison and its value solely represents the binary result of whether $z' > r'$. No information is provided that allows either party to see the actual difference. This stops Party A from being able to discover the difference. For example, if Party A replaced value b with 0, the end result would state $a > b$ without stating the value of a or the difference between a and b.

In the second case, let us assume that Party B is malicious and wants to know the plain text values of a and b. Party B receives z from Party A and wants to recover r such that they can decipher $a \cdot b^{-1}$. Given that r is chosen uniformly though a large number space, the value z does not reveal information about the difference of a and b. Given a scenario in which $z = 50$ in cyclic group of 200, for every possible difference there is a corresponding random value r' that would allow you to arrive at that area. If $a - b = 1$ then $r = 149$, but that is just as likely as $a - b = 2$ and $r = 148$, which continues for every possible value of $a - b$. The uniform distribution of the random value r is able to perfectly hide the value of $a - b$ because in a group of n, there are n equally likely possibilities for the actual inputs.

6.3.5 Correctness

The protocol must terminate with the correct result in every possible case. Though the number of individual cases will be dependent upon the size of the cyclic groups that are being used in the protocol, there are a base 8 cases. These cases cover the different values of r'_l, z'_l, and c' and show that the resultant of the protocol is correct with respect to these values. Along with the following case descriptions is Figure 6.3, which provides a visual guide for following along. The plotted values x, r, z represent the x, r'_l, z'_l values from the protocol. Recall that our goal is to find the location of x and determine the resultant from that. A simpler table in Section 6.3.5 is also provided for a cleaner analysis.

Note: The values of c' are modulus 2^l which means they just state which number is farther into its respective segment

Case 1. $r'_l = 0$**,** $z'_l = 0$**, and** $c' = 0$**.** In this case the values of r and z are both in the first of the two segments of the group. The value of $c' = 0$ denotes that $r'_l < z'_l$. Since we subtract r from z to get x, we know that this still falls in the first segment of the group. This tells us that $a < b$ which corresponds to the answer 0 that would be given by $(r'_l + z'_l + c') \bmod 2$.

Case 2. $r'_l = 0$**,** $z'_l = 0$**, and** $c' = 1$**.** In this case the values of r and z are both in the first of

the two segments of the group. The value of $c' = 1$ denotes that $r'_l > z'_l$. Since we subtract r from z to get x, we know that this cycles us to the second segment of the group. This tells us that $a > b$ which corresponds to the answer 1 that would be given by $(r'_l + z'_l + c') \bmod 2$.

Case 3. $r'_l = 0$, $z'_l = 1$, **and** $c' = 0$**.** In this case r lies in the first segment and z lies in the second segment of the group. The value of $c' = 0$ denotes that $r'_l < z'_l$. Since we subtract r from z to get x, we know that this keeps us in the same group. This tells us that $a > b$ which corresponds to the answer 1 that would be given by $(r'_l + z'_l + c') \bmod 2$.

Case 4. $r'_l = 0$, $z'_l = 1$, **and** $c' = 1$**.** In this case r lies in the first segment and z lies in the second segment of the group. The value of $c' = 1$ denotes that $r'_l > z'_l$. Since we subtract r from z to get x, we know that this brings us down to the first segment of the group. This tells us that $a < b$ which corresponds to the answer 0 that would be given by $(r'_l + z'_l + c') \bmod 2$.

Case 5. $r'_l = 1$, $z'_l = 0$, **and** $c' = 0$**.** In this case r lies in the second segment and z lies in the first segment of the group. The value of $c' = 0$ denotes that $r'_l < z'_l$. Since we subtract r from z to get x, we know that this cycles us to the second segment of the group. This tells us that $a > b$ which corresponds to the answer 1 that would be given by $(r'_l + z'_l + c') \bmod 2$.

Case 6. $r'_l = 1$, $z'_l = 0$, **and** $c' = 1$**.** In this case r lies in the second segment and z lies in the first segment of the group. The value of $c' = 1$ denotes that $r'_l > z'_l$. Since we subtract r from z to get x, we know that this cycles us to down to the first segment of the group. This tells us that $a < b$ which corresponds to the answer 0 that would be given by $(r'_l + z'_l + c') \bmod 2$.

Case 7. $r'_l = 1$, $z'_l = 1$, **and** $c' = 0$**.** In this case the values of r and z are both in the second of the two segments of the group. The value of $c' = 0$ denotes that $r'_l < z'_l$. Since we subtract r from z to get x, we know that this moves us to the first segment of the group. This tells us that $a < b$ which corresponds to the answer 0 that would be given by $(r'_l + z'_l + c') \bmod 2$.

Case 8. $r'_l = 1$, $z'_l = 1$, **and** $c' = 1$**.** In this case the values of r and z are both in the second of the two segments of the group. The value of $c' = 1$ denotes that $r'_l > z'_l$. Since we subtract r from z to get x, we know that this cycles us through the first segment and back to the second segment of the group. This tells us that $a > b$ which corresponds to the answer 1 that would be given by $(r'_l + z'_l + c') \bmod 2$.

6.4 Future Work

The pivot in the thesis and lack of time to return prevented us from implementing the protocols listed above. Though these protocols should intertwine correctly, they have not been tested and this could be an area of future work regarding our protocol. Another area of potential future work is adapting our protocol for the circumstances that this setting presents. Our protocol could also undergo an adaptation for the case when one

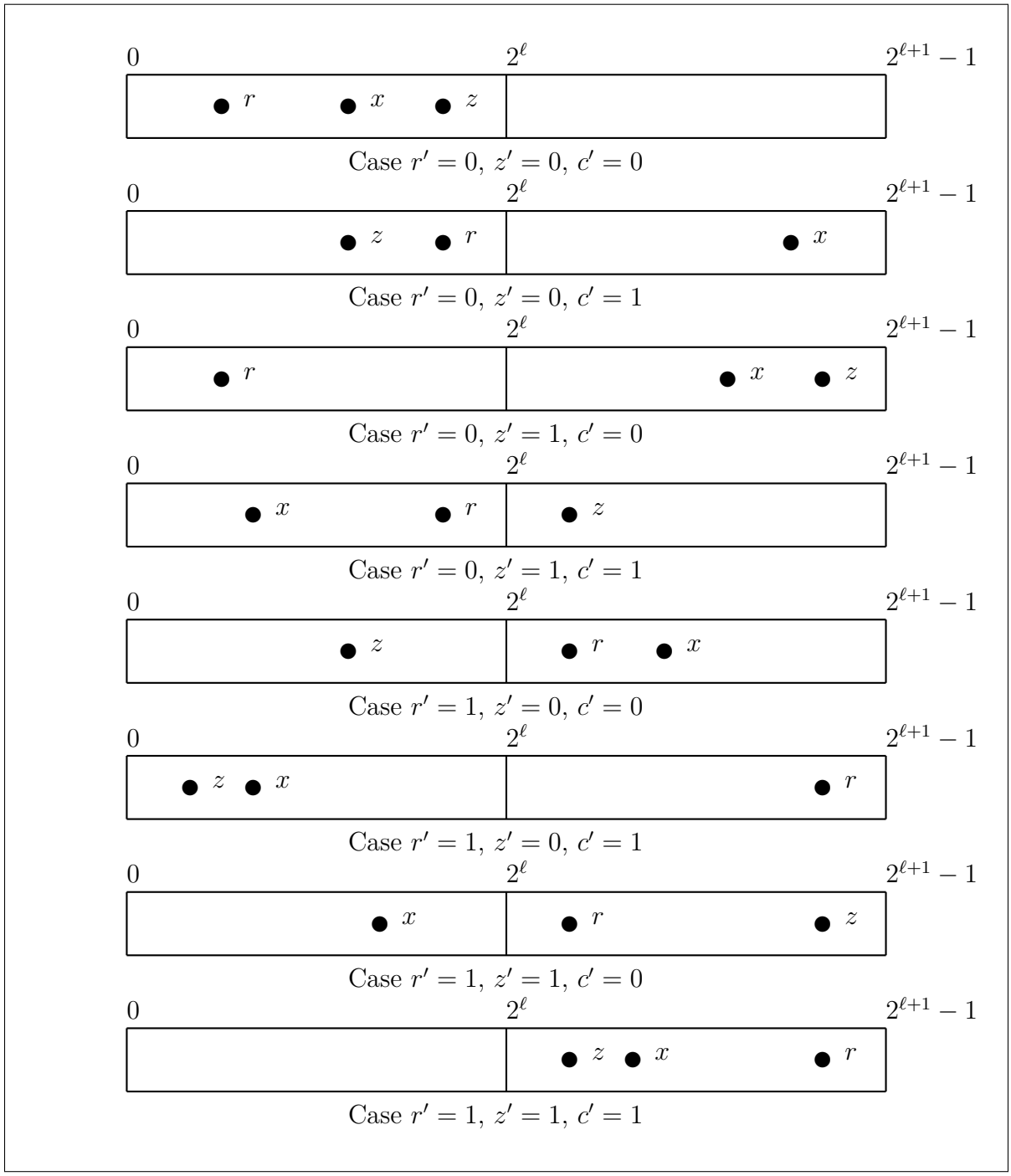

Figure 6.3: All variations described in geo-spatial correctness

c' $(r' > z')$	r'	z'	Result
0	0	0	$b \geq a$
0	0	1	$a > b$
0	1	0	$a > b$
0	1	1	$b \geq a$
1	0	0	$a > b$
1	0	1	$b \geq a$
1	1	0	$b \geq a$
1	1	1	$a > b$

Table 6.1: Table for variations in geo-spatial correctness

party has both of the inputs which would simplify the geo-spatial wrapper protocol.

Chapter 7

Conclusion and Future Work

This chapter summarizes the thesis of this paper and discusses the contributions that can potentially be used in the field. It also explores the limitations of the current work and the potential areas where future work can be done regarding our cryptosystem. Though an example use of the threshold homomorphic property was presented, their are still undiscovered uses for this type of property. This chapter is split into three main sections:

1. Section 7.1 provides a brief summary of the thesis restating the important aspects of the research.

2. Section 7.2 presents a discussion on the contributions of the cryptosystem and talks about the problems that were encountered while designing and implementing it.

3. Section 7.3 talks about potential future uses for our cryptosystem and talks about areas in which more work can be done using the cryptosystem.

7.1 Summary

This thesis reviewed some of the public key cryptosystems that are currently used in practice and described the main functionalities of their encryption schemes as:

1. $\mathsf{Gen}(\lambda) \rightarrow (p_k, s_k)$

2. $\mathsf{Enc}(m, p_k) \rightarrow (c)$

3. $\mathsf{Dec}(c, s_k) \rightarrow (m)$

It discussed the creation of our encryption system and broke down the encryption into the same three basic functions as the systems currently in practice. The unique homomorphic

properties of the system were discussed and the hardness assumptions securing the encryption scheme that was used were explained. Alongside the explanation of the protocol used for secure integer comparison, there was also a description for the expansion of the protocol to include integers beyond the scope of the security parameters for the protocol. This expansion technique allowed for linear growth (in number of encryptions) which is sufficient when comparing it to other schemes in use. A simulation security proof was provided to explain the security of the protocol and explain the steps put into place to prevent information leakage. The implementation through the use of SAGE, was described and the algorithms for the encryption decryption and key generation functions were shown. Core concepts for these algorithms were also reviewed to allow an easier understanding. Finally, an example use of our cryptosystem was discussed in terms of a geo-spatial analysis protocol that was studied during the thesis. This protocol was explained in it's stages and showed an instance in which using the secure integer comparison from our protocol would be useful. Future works for the implementation of the cryptosystem into other fields was also briefly discussed.

7.2 Discussion

The main contributions that are made towards secure computation throughout this paper are the development of our cryptosystem, the discovery of the threshold homomorphic property, and the implementation of this property through a new protocol for secure integer comparison.

The research and use of prime power groups allow for an exponential addition under encryption that creates a unique threshold function. This function allows for anyone using the scheme to compare multiple messages together to see if they fit a certain mathematical criteria. For example the threshold function that is discussed in this paper is:

$$f(m_1, m_2) = \begin{cases} 0 & m_1 + m_2 \geq t \\ m_1 + m_2 & \text{otherwise.} \end{cases}$$

The versatility of this threshold function is something that we do not believe we have fully realized. However, by implementing a new protocol displaying one of the possibilities we have gained more understanding as to some of the applications this property can be used for.

When looking at some of the problems that were encountered during this research, one obvious indicator was the pivot from the geo-spatial analysis protocol to the secure

integer computation protocol. The scale of the thesis was reduced when the lack of a sufficient way to securely compute integer comparisons was discovered. The intention for this change was to fill the absence of an efficient protocol for future researchers.

7.3 Future Work

While this thesis was able to explore the cryptosystem that was developed, there are still an abundance of interesting paths that this research could be extended upon. In Chapter 6 there was discussion about the extension of our protocol into the larger geo-spacial protocol. Improving on the cohesiveness of the two protocols and developing/implementing a full protocol for the purpose of geo-spatial analysis is one area which would be interesting to invest future works in.

Within our description of the homomorphic threshold property, we discussed the implication of the one way security, in that information could be leaked without a second cryptosystem. While this was not something that we were looking to use in our protocol, it could be interesting to find research areas in which this one sided security would be more valued and try to implement the system in that area of study.

There are a number of other fields in which secure computation is currently being used, such as database linkage and online auctions. While we have only looked at the threshold property in terms of addition and subtraction, there could be more variances of what this type of property would allow. More research into the field of prime power groups could lead to new discoveries with what you are able to do in the exponent space.

7.4 Conclusion

The wealth of information available online is growing at a staggering pace and to compensate for having to process this information new breakthroughs in computation are of the utmost importance. By making improvements in the field of secure computation, we can reduce the strain on servers that need to process this information. The continued research going into this field should help improve the speed at which people will be able to jointly compute information.

Bibliography

[1] Reinhold Baer et al. Classes of finite groups and their properties. *Illinois Journal of mathematics*, 1(2):115–187, 1957.

[2] Foteini Baldimtsi and Olga Ohrimenko. Sorting and searching behind the curtain. In *International Conference on Financial Cryptography and Data Security*, pages 127–146. Springer, 2015.

[3] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. *SIAM Journal on Computing*, 33(4):783–818, 2004.

[4] 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. ACM, 1990.

[5] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 92–111. Springer, 1994.

[6] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, volume 6632, pages 169–188. Springer, 2011.

[7] Ian F Blake and Vladimir Kolesnikov. Conditional encrypted mapping and comparing encrypted numbers. In *International Conference on Financial Cryptography and Data Security*, pages 206–220. Springer, 2006.

[8] Christian Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 120–127. ACM, 1999.

[9] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *Advances in Cryptology Crypto 2000*, pages 93–111. Springer, 2000.

[10] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

[11] Ran Canetti, U Friege, Oded Goldreich, and Moni Naor. Adaptively secure multiparty computation. 1996.

[12] Rhys Carlton, Aleksander Essex, and Krzysztof Kapulkin. Threshold homomorphic properties of prime power subgroups with applications to secure integer comparison. In submission, Western University, Canada, 2017.

[13] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 577–594. Springer, 2010.

[14] Jean-Sébastien Coron, Antoine Joux, Avradip Mandal, David Naccache, and Mehdi Tibouchi. *Cryptanalysis of the RSA Subgroup Assumption from TCC 2005*, pages 147–155. 2011.

[15] Geoffroy Couteau. Efficient secure comparison protocols. *IACR Cryptology ePrint Archive*, 2016:544, 2016.

[16] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Information Security and Privacy: 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007. Proceedings*, pages 416–430, 2007.

[17] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1(1):22–31, 2008.

[18] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. A correction to "efficient and secure comparison for online auctions". *Int. J. Appl. Cryptol.*, 1(4):323–324, 2009.

[19] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, volume 1992, pages 119–136. Springer, 2001.

[20] Ivan Damgård and Jesper Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. *Advances in Cryptology-CRYPTO 2003*, pages 247–264, 2003.

[21] Michael John De Smith, Michael F Goodchild, and Paul Longley. *Geospatial analysis: a comprehensive guide to principles, techniques and software tools*. Troubador Publishing Ltd, 2007.

[22] David Steven Dummit and Richard M Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

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

[24] Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 554–563. ACM, 1994.

[25] Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Cryptographers' Track at the RSA Conference*, pages 457–471. Springer, 2001.

[26] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007(1):013801, 2007.

[27] Martin Franz, Björn Deiseroth, Kay Hamacher, Somesh Jha, Stefan Katzenbeisser, and Heike Schröder. Towards secure bioinformatics services (short paper). In *International Conference on Financial Cryptography and Data Security*, pages 276–283. Springer, 2011.

[28] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography*, volume 7, pages 330–342. Springer, 2007.

[29] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[30] Oded Goldreich. A note on computational indistinguishability. *Information Processing Letters*, 34(6):277–281, 1990.

[31] 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, pages 365–377, 1982.

[32] Jens Groth. Cryptography in subgroups of z_n^*. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings*, pages 50–65, 2005.

[33] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[34] Ioannis Ioannidis and Ananth Grama. An efficient protocol for yao's millionaires' problem. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 6–pp. IEEE, 2003.

[35] Bin Jiang and Xiaobai Yao. *Geospatial analysis and modelling of urban structure and dynamics*, volume 99. Springer Science & Business Media, 2010.

[36] Kristján Valur Jónsson, Gunnar Kreitz, and Misbah Uddin. Secure multi-party sorting and applications. *IACR Cryptology ePrint Archive*, 2011:122, 2011.

[37] Jonathan Katz and Yehuda Lindell. Handling expected polynomial-time strategies in simulation-based security proofs. In *Theory of Cryptography Conference*, pages 128–149. Springer, 2005.

[38] Mehmet Kuzu, Mohammad Saiful Islam, and Murat Kantarcioglu. Efficient similarity search over encrypted data. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 1156–1167. IEEE, 2012.

[39] Xing-Xin Li, You-Wen Zhu, and Jian Wang. Efficient encrypted data comparison through a hybrid method. *Journal of Information Science & Engineering*, 33(4), 2017.

[40] Hsiao-Ying Lin, Wen-Guey Tzeng, et al. An efficient solution to the millionaires' problem based on homomorphic encryption. In *ACNS*, volume 5, pages 456–466. Springer, 2005.

[41] Yehuda Lindell. How to simulate it-a tutorial on the simulation proof technique. *IACR Cryptology ePrint Archive*, 2016:46, 2016.

[42] Quanquan Ma. *The LTV Homomorphic Encryption Scheme and Implementation in Sage*. PhD thesis, Worcester Polytechnic Institute, 2013.

[43] James L Massey. Cryptography: Fundamentals and applications. In *Copies of transparencies, Advanced Technology Seminars*, volume 109, page 119, 1993.

[44] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[45] Silvio Micali and Phillip Rogaway. Secure computation. In *Annual International Cryptology Conference*, pages 392–404. Springer, 1991.

[46] Majid Nateghizad, Zekeriya Erkin, and Reginald L Lagendijk. An efficient privacy-preserving comparison protocol in smart metering systems. *EURASIP Journal on Information Security*, 2016(1):11, 2016.

[47] National Institute of Standards and Technology (NIST). Recommended elliptic curves for federal government use, 1999.

[48] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT 1999*, pages 223–238. Springer-Verlag, 1999.

[49] Fernando Perez, Brian E Granger, and John D Hunter. Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2):13–21, 2011.

[50] David Pointcheval. New public key cryptosystems based on the dependent-rsa problems. In *Eurocrypt*, volume 99, pages 239–254. Springer, 1999.

[51] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Eurocrypt*, volume 96, pages 387–398. Springer, 1996.

[52] Minghua Qu. Sec 2: Recommended elliptic curve domain parameters. 1999.

[53] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[54] Ron Rothblum. Homomorphic encryption: From private-key to public-key. In *TCC*, volume 6597, pages 219–234. Springer, 2011.

[55] Berry Schoenmakers and Pim Tuyls. *Practical Two-Party Computation Based on the Conditional Gate*, pages 119–136. Springer Berlin Heidelberg, 2004.

[56] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[57] Claude E Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4):656–715, 1949.

[58] William Stein and David Joyner. Sage: System for algebra and geometry experimentation. *ACM SIGSAM Bulletin*, 39(2):61–64, 2005.

[59] Minh Van Nguyen. Number theory and the rsa public key cryptosystem. *sage*, 5:7, 2008.

[60] Thijs Veugen. Comparing encrypted data. *Multimedia Signal Processing Group, Delft University of Technology, The Netherlands, and TNO Information and Communication Technology, Delft, The Netherlands, Tech. Rep*, 2011.

[61] Yodai Watanabe, Junji Shikata, and Hideki Imai. Equivalence between semantic security and indistinguishability against chosen ciphertext attacks. In *International Workshop on Public Key Cryptography*, pages 71–84. Springer, 2003.

[62] Can Xiang, Chunming Tang, Yunlu Cai, and Qiuxia Xu. Privacy-preserving face recognition with outsourced computation. *Soft Computing*, 20(9):3735–3744, 2016.

[63] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[64] Xun Yi, Russell Paulet, and Elisa Bertino. *Homomorphic encryption and applications*, volume 3. Springer, 2014.

Curriculum Vitae

Name:	Rhys Carlton
Post-Secondary Education and Degrees:	Western University 2011 - 2015 B. Eng University of Western Ontario London, ON
Honours and Awards:	Dean's List 2011-2015
Related Work Experience:	Teaching Assistant The University of Western Ontario 2015 - 2017

Publications:

Rhys Carlton, Aleksander Essex, and Krzysztof Kapulkin. Threshold Homomorphic Properties of Prime Power Subgroups with Applications to Secure Integer Comparison. In submission.