Homomorphic Encryption in Algebraic settings

Jack Aiston

Thesis submitted for the degree of
Doctor of Philosophy

School of Mathematics & Statistics
Newcastle University
Newcastle upon Tyne
United Kingdom

2019

Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council, Centre for Doctoral Training in Cloud Computing for Big Data [grant number EP/L015358/1].
First and foremost I'd like to thank my family: Sharon, Malcolm and Olivia Aiston for supporting me all the way through my time at University. I never would have been able to achieve half of what I have without them. The other major factor in my research was my excellent supervisor Andrew Duncan. I really appreciate the support he has provided while at the same time allowing me to dictate my own direction for my research.
Outside of my studies I'd like to thank Naomi Hannaford, Tom Lowe and Kieran Peel for helping me make the most of my PhD experience as a whole.
Finally thank you to Paul Watson and Darren Wilkinson for giving me this opportunity in the CDT. Also, all the people who I have worked with including but not limited to: Matt Forshaw, Oonagh McGee, Tom Cooper, Peter Michalak, Lauren Roberts, Hugo Firth, David Robertson, Cameron Trotter and Tom Owen.

Abstract

Cryptography methods have been around for a long time to protect sensitive data. With data sets becoming increasingly large we wish to not only store sensitive data in public clouds but in fact, analyse and compute there too. The idea behind homomorphic encryption is that encryption preserves the structure and allows us to perform the same operations on ciphertext as we would on the plaintext. A lot of the work so far restricts the operations that can be performed correctly on ciphertexts. The goal of this thesis is to explore methods for encryption which should greatly increase the amount of analysis and computation that can be performed on ciphertexts.

First of all, we will consider the implications of quantum computers on cryptography. There has already been research conducted into quantum-resistant encryption methods. The particular method we will be interested in is still classical. We are assuming these schemes are going to be used in a post-quantum world anyway, we look at how we can use the quantum properties to improve the cryptosystem. More specifically, we aim to remove a restriction that naturally comes with the scheme restricting how many operations we can perform on ciphertexts.

Secondly, we propose a key exchange protocol that works in a polynomial ideal setting. We do this so that the key can be used for a homomorphic cryptography protocol. The advantage of using key exchange over a public key system is that a large proportion of the process needs to be carried out only once instead of needing a more complicated encryption function to use for each piece of data. Polynomial rings are an appropriate choice of structure for this particular type of scheme as they allow us to do everything we need. We will examine how we can perform computation correctly on ciphertexts and address some of the potential weaknesses of such a process.

Finally after establishing a fully homomorphic encryption system we will take a more in-depth look at complexity. Measuring the complexity of mathematical problems is, of course, crucial in cryptography, but the choice of measure is something we need to consider seriously. In the final chapter we will look at generic complexity as its gives us a good feel for how difficult the typical instances of a problem are to solve.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Despite the cloud providing the incredible potential for the future of computation – extremely fast processing power and seemingly unlimited storage to name just a couple – the move toward cloud computing faces challenges that must be addressed before it is fully adopted. The problem that concerns a lot of potential users is that others may also have access to the information that is sent to the cloud [Kulkarni *et al.* (2012)]. This is especially concerning for those dealing with sensitive data. If security concerns are particularly high, for example when it comes to medical or financial data, encryption methods must be used to put minds at ease that the data won't fall into the wrong hands.

NIST (National Institute of Standards and Technology)-recommended cryptographic algorithms have been approved after receiving intense security analysis [Barker *et al.* (2012)]. More so, this is an ongoing process where each algorithm is constantly being examined. Three classes of approved algorithms are used: Hash functions; symmetric-key algorithms; asymmetric-key algorithms.

In 2017 NIST began the process of trying to select new cryptographic algorithms to augment already widely accepted schemes [Alagic *et al.* (2019)]. At the end of that year, the first round of candidates were announced and the public was encouraged to comment on them. In January the first round ended and second round began with 26 of the candidates being invited to add any updates to the schemes. The 3rd round is set to begin 2020/2021. One of the main aims of introducing these new schemes is to deal with the challenges of modern and near future-security.

A new generation of security issues has begun with the rapid adoption of the internet of things (IOT). There are more connected devices than people on the planet. We are currently playing catch up trying to secure these devices. The problem is only getting worse as the rate at which IOT is being adopted increases. It is said that there is only a short window of time to deal with the problem before it gets almost irreversibly out of control.

In more standard computer network settings, security is a well studied and implemented

Security strength		Through 2030	2031 and beyond
≤ 80	Applying	Disallowed	
	processing	Legacy-use	
112	Applying	Acceptable	Disallowed
	Processing		Legacy use
128	Applying/Processing	Acceptable	Acceptable
192		Acceptable	Acceptable
256		Acceptable	Acceptable

Table 1.1: Security-strength time frames

topic. We need to use all that knowledge and experience to secure all of our new devices. Just because most devices in the IOT landscape are weak and perform a limited number of tasks, doesn't mean that there can't be large negative repercussions from malicious people who can get access. One of the more famous examples of this is the Mirai botnet attack in 2016 [Antonakakis *et al.* (2017)].
As tempting as it may be to use all the best settings to achieve maximum security, we have standards on what is considered secure. Trying to use larger parameters will just cause a strain on computation and bandwidth, something we want to avoid in general but a precious resource in particular for IOT. Table 1.1 gives an outline of the security strengths (in bits) that will be acceptable to use over the next couple of decades. The number one goal of cryptography has always been to ensure readable versions of sensitive or private data do not get into the wrong hands, even if it is potentially accessible. It was suggested in the 70s however that it may be possible to do more than just safely store data; in fact, useful operations could be performed on encrypted data [Rivest *et al.* (1978)]. This concept is known as homomorphic cryptography. Figure 1.1 provides a visual summary of what the ultimate goal of this area of research is. The vertical arrows represent cryptographic methods as they have always been, a method to secure information that may be stored in an insecure place. The horizontal arrows represent the methods of data analysis we currently use. The bottom arrow is the version we have been using longer, data analysis on some local device. The top arrow represents the move to cloud where the same work can potentially be carried out much faster. Should a practical solution for homomorphic cryptography be found, the need for the lower arrow could be removed. In many IOT settings this offline analysis is not even possible as the devices holding the data are too weak to carry out the required functions. Since the idea was presented, homomorphic encryption has been through a few evolutions [Acar *et al.* (2018)]. The first colleciton of homomorphic encryption methods are known as partial homomorphic encryption. These protocols are homomorphic with respect to a single operation. Examples of this are RSA (after Rivest, Shamir and Adleman) where $E(m_1) * E(m_2) = E(m_1 * m_2)$, Goldwasser-Micali where $E(m_1) * E(m_2) = E(m_1 \oplus m_2)$ and Paillier where $E(m_1) * E(m_2) = E(m_1 + m_2)$ [Yi *et al.*

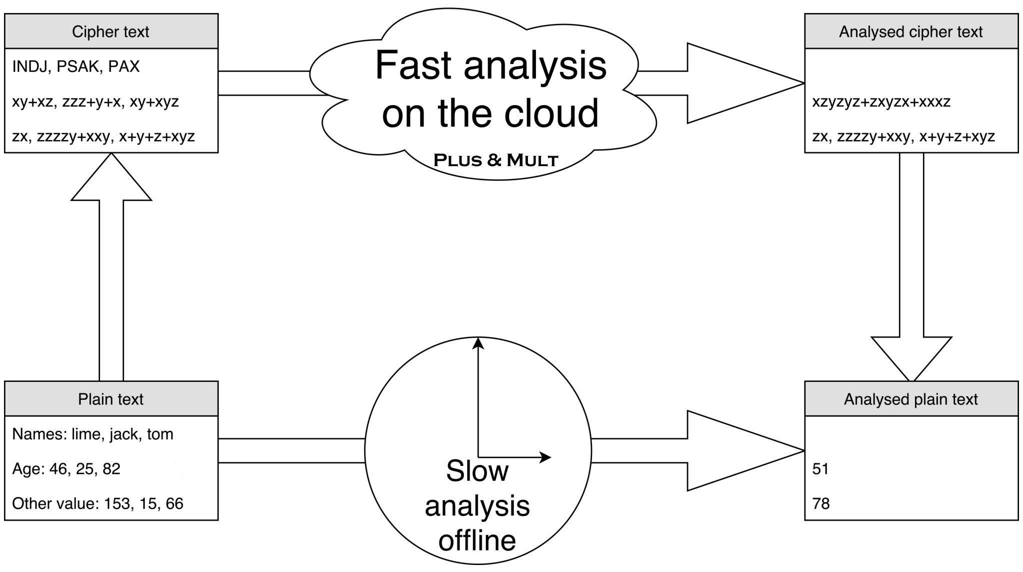


Figure 1.1: Summary of data analysis

(2014)]. While an interesting step in the right direction, no meaningful computation can be performed using these properties alone.
Somewhat homomorphic cryptosystems are a class of cryptosystems that have encryption functions that are only approximately homomorphic. The problem here is that the errors in the approximations accumulate so that only finitely many operations are possible without the data becoming irretrievably corrupted. One of the advantages of these systems is that some allow both addition and multiplication to be performed on ciphertexts. In [Naehrig *et al.* (2011)] a somewhat homomorphic cryptosystem that takes advantage of the Ring Learning with Errors problem is presented.
The next important advance was the development of fully homomorphic encryption systems. An encryption scheme is called fully homomorphic if we can perform as many operations of our choice on ciphertexts as we need. In the last 10 years, a few of these methods have been proposed [Acar *et al.* (2018)]. The first of these methods is Gentry's lattice-based scheme. To begin with, Gentry started with a somewhat homomorphic encryption scheme and developed a method to deal with the noise that comes with such schemes. This method came in two steps, first was known as squashing that reduces the degree of the ciphertexts to a point that the second step can handle. The second step is known as bootstrapping which safely re-encrypts the data, resetting the error to the desired amount. Moving forward with the idea of a bootstrapping method, a new fully homomorphic encryption method was suggested, but this time set in the integers [Van Dijk *et al.* (2010)]. The beauty of this method is that simply being integers is much more conceptually simple. The third method is based on the ring learning with errors problem. The setting for this scheme is in $R_q \equiv \mathbb{Z}_q[x]/(f(x))$, the ring of polynomials modulo

some polynomial f with coefficients in $\mathbb{Z}_q$. This scheme, discussed in [Lyubashevsky *et al.* (2010)], has drawn a lot of attention due to two major factors. First, the learning with errors problem is thought to be very difficult to solve even in a post-quantum world. Secondly, ciphertexts are relatively small - a very important aspect of considering how quickly ciphertexts can grow when multiplication is performed.
A new threat to the world of security is quantum computing; which is quickly being accepted as a reality, not only in most of our lives but potentially in the next decade [Easttom (2017)]. In conventional computing, information is stored as a sequence of bits, each of which is a 0 or 1. Quantum computing in a similar sense deals with information that is stored as a sequence of qubits, each of which is 0, 1, or a quantum superposition of the two. When it comes to physically observe a qubit, a measurement is performed, and if in a superposition, the qubit will collapse down to either a 0 or 1. Although there is an infinite number of combinations of superposition for a qubit, an important restriction of quantum computing is that it is physically impossible to store a qubit in a superposition. This means that there is no increase in efficient storage compared to classical computing. One of the key areas where quantum computing does outperform classical is its speedup of various algorithms. One of the earliest examples of this is the Deutsch-Jozsa algorithm [Deutsch & Jozsa (1992)]. The algorithm solves the problem of determining if a function that outputs a single bit is constant or balanced (outputs a 0 for half its inputs and a 1 for the other half). In the classical version, a number, exponential in the size of the input, total queries to the function are required to solve the problem. However, the quantum Deutsch-Jozsa algorithm only requires a single query. This example of an exponential speedup is the motivation for quantum attacks in cryptography.
While considered secure from all known conventional attacks, integer factorization, discrete log, and elliptic curve methods each have been subject to efficient quantum attacks, for example the well known Shor's algorithm [Shor (1994)] for solving the discrete logarithm problem. It is uncertain when access to a quantum computer will be available but, as already stated, many believe it to be in our lifetime. As such alternative "quantum-resistant" methods need to be kept in mind and ready to go as soon as possible. Fortunately, while old methods may become obsolete, this emerging area of computing has already proven to bring with it new possibilities. Random numbers are required in many security methods. Unfortunately in classical computing, we aren't able to generate genuine random numbers. We, therefore, have to rely on pseudorandom numbers which are generated from some algorithm. On the other hand, a qubit can be set up in a superposition of 0 or 1 where, after performing a measurement, there is an even chance of observing the 0 or the 1. Assuming no one has tampered with this qubit, this measurement will be truly random. Security initiatives in quantum computing therefore no longer need to worry about an attacker having malicious access to the pseudorandom number generator. There is still cause for concern with issues such that the qubits may still be tampered with, however, research

has gone into testing for such malicious activity [Colbeck & Kent (2011)].

One of the most high profile cryptographic methods to emerge from quantum computing so far is the BB84 protocol [Shor & Preskill (2000)]. The symmetric key algorithms mentioned earlier, require two or more parties to have knowledge of a shared key that should be unknown to anyone else. Key exchange methods already exist in classical computing that makes this possible, however, many of the more well-used versions rely on the discrete log and other problems that we have already mentioned will become insecure in quantum settings. Alternatively, the BB84 protocol performs the key exchange and bases its security on the fundamental principles of quantum mechanics. The reason the scheme works is due to the fact that taking a measurement of a qubit requires a choice of basis in which to measure in. This is the secret that the parties looking to exchange keys don't tell anyone. An eavesdropper can't know what the choice of basis was used so an attempt to read the key has a non zero probability of altering the key in a noticeable way.

In chapter 3 we will have a look at an example of an advantageous change that can be made to an encryption scheme in a quantum setting. Lattice-based methods are based on a different difficult problem and are one of the front-runner candidates for cryptosystems in a quantum world. In particular, the NTRUEncrypt public key cryptosystem has accumulated a lot of interest since its initial development towards the end of the 90's [Pecen *et al.* (2014)], in particular two NTRU schemes have made it to the second round of NIST post-quantum competition earlier [Alagic *et al.* (2019)].

Another example of a lattice-based protocol is the GGH (Goldreich-Goldwasser-Halevi) encryption scheme [Goldreich *et al.* (1997)]. This is an example of the somewhat homomorphic encryption methods where the number of operations (in this case addition) that can be performed correctly is restricted by an error component. Although many schemes have these error terms, this one, in particular, has an error that can be easily represented by a random walk [Spitzer (2013)]. Because of this, after a given number of additions have been performed on a ciphertext, we can create a distribution of the potential accumulative error. Most importantly, there is a finite number of options each of which we know the probability of.

There are many variations of the addition circuit on a quantum computer. We will consider the circuit in [Takahashi *et al.* (2009)], which is very similar to that of the classical version, in section 3.2.1 for adding together ciphertexts. The major difference is that the quantum version will preserve superpositions. For example, suppose we have qubits represent a superposition of values 2 and 4 and this is added to a superposition of values 8 and 11. The circuit will output a superposition of (2+8), (2+11), (4+8), and (4+11). Updating the GGH system in such a way that a ciphertext is in a superposition of all potential classical ciphertexts means that we know the exact form of a sum even without performing a measurement. The rest of section 3.2 contains original content which works towards the main aim of this chapter, to look at quantum methods to correct this error

back to a standard amount for a fresh ciphertext.
The updates for the error correction in this chapter also bring with them new issues that put the security of the protocol at risk. In section 3.3 we will look at methods to deal with these issues. Fortunately, the problem isn't a full break of the cryptosystem, nevertheless, it is still, of course, important to be aware of different attacker goals [Stinson (2005)]. Two approaches will be used for protection. The first comes from classical computing and is based on work in [Watson (2012)], where we temporarily use private clouds at the start to obscure information that would otherwise put individual data at risk when processing. The second method is quantum and takes inspiration from the quantum key exchange discussed earlier. We aim to use the fact that if an eavesdropper tries to look at particular information, a fundamental change will occur to the qubits involved. This may not stop the eavesdropper but will give us a tool to see if there has been malicious activity.
In section 3.4 we will take a look at how some basic statistical functions were implemented in a ring learning with errors setting [Naehrig *et al.* (2011)] and study how plausible it would be to do the same in this scheme.
In chapter 4 we aim to build a fully homomorphic encryption scheme. Rings are an appropriate choice of structure for this particular type of scheme due to the plus and multiplication operations available [Kahrobaei *et al.* (2019)]. They can be built up into more meaningful functions that would enable data analysis. In particular we will be interested in polynomial rings, much like in [Rai (2004)].
The work in [Kahrobaei *et al.* (2019)] uses rings to build a fully homomorphic encryption scheme using rings and ideals. Much like a lot of the lattice-based methods, encryption is performed by multiplying a plaintext by a secret key followed by adding a random element. In this scheme in particular a message, m, is encrypted by multiplying by a random private idempotent, r, of a private idea followed by adding a random element, i, of that private ideal i.e. of the form $mr + i$. Any issue we discuss later is how to form a correct homomorphism. This scheme achieves an additive homomorphism by firstly using the distributive property of multiplying the plaintext by the same random idempotent $m_1r + m_2r = (m_1 + m_2)r$. Secondly the only requirement for the random element added during the encryption is that is an element of the private ideal, a property that will be maintained by adding two ideal elements together. The multiplicative homomorphism is achieved by taking advantage of those two points alongside the fact that r was an idempotent, namely $m_1r \cdot m_2r = m_1m_2r^2 = m_1m_2r$.
Testing for ideal membership has been used as a difficult problem forming the basis of cryptosystems proposed in rings [Albrecht *et al.* (2016)]. These protocols are at risk of being broken due to the existence of Gröbner bases. While it is thought to be difficult to establish whether a ring element belongs to an ideal, based on the basis elements of the ideal, a Gröbner basis is constructed in such a way that the problem is easy. Therefore, the security of these categories of schemes is reliant on not being able to find a Gröbner

basis. In commutative polynomial rings, covered in section 4.1, it is known that any ideal basis has a corresponding Gröbner basis and can be found using Buchberger's algorithm [Buchberger & Winkler (1998)]. Commutative schemes base their security on the fact that some Gröbner bases take an infeasible length of time to find. We will, however, be looking at a non-commutative scheme in section 4.2, the advantage being some bases for ideals have provably no Gröbner basis. This will be done by looking at examples of where Mora's algorithm [Mora (1985)] does not terminate. These initial examples will come from [Rai (2004)]. A lot of work in these Polly Cracker systems is based on Gröbner bases that are infeasible to calculate but theoretically finite. Work with infinite Gröbner bases isn't as prevalent with Rai's work typically being what is referred to when talking about infinite Gröbner bases, for example in [Bulygin (2005)] and [Cortés *et al.* (2007)].
In section 4.3 we will use an ideal whose basis can't be converted into a Gröbner basis. From this we will build a key exchange protocol based on the Polly Cracker scheme that is covered in [Rai (2004)]. We wish to keep any encryption method as computationally inexpensive as possible so the majority of the work will be done by the key exchange (if data is being streamed for example we can't afford time-consuming encryption methods). The accompanying encryption function that will depend on the generated secret key is conceptually very simple. Although the simplicity of it is an excellent feature, two more advantages come with this specific choice of function; the first of which is the homomorphic property. Addition is preserved under the encryption function regardless of which polynomial space we work in. As for multiplication, we discuss how that can also be preserved provided we are a little more careful in our choice of polynomial space. The remainder of this section will focus on our efforts to strengthen the scheme against a potential attack.
Braid groups are another structure that have been studied for their cryptographic potential [Flores & Kahrobaei (2017)] and are the justification of the second advantage of our choice of encryption function which we will look at in more detail in section 4.4. Hecke algebras are connected to braid groups and having both addition and multiplication operations make it a good candidate for the choice of polynomial space to work in. This section will show how we can upgrade our scheme. With this particular setup, the encryption function will output a polynomial that has similar properties to instances of the conjugacy problem from braid groups. This will hopefully give our method an extra level of security from other potential attacks.
Testing of the work in this chapter was done using the functional programming language Haskell. Using a language like this is helpful as programs are built up as a series of functions, a more natural way to express the methods we use than an object-oriented language like Java. This approach is of great importance as working with polynomials is much slower than working with integers like a lot of current protocols. In particular, the famous MapReduce method [Dean & Ghemawat (2008)] can be utilised to help speed up operations over large amounts of ciphertexts. In this work, polynomials are represented as

lists and the two MapReduce functions each perform fundamental changes to those lists.

- map - $(k_1, v_1) \to \text{list}(k_2, v_2)$
- reduce - $(k_2,\text{list}(v_2)) \to \text{list}(v_2)$

(k and v represent a key-value pair).
Finally, in chapter 5, we will look at the complexity of finding partial Gröbner bases. Although we will see that there are certain ideal bases that have an infinite number of polynomials in their Gröbner basis, it is only important to find the correct finite subset. We, therefore, will look at the complexity of finding this subset as opposed to finding a complete Gröbner basis.
When it comes to measuring complexity we need to be careful which measure we use. In section 5.1 we will use the quicksort algorithm [Hoare (1961)] as an example to understand the various measures. While something like worst-case complexity will tell us if a particular hard problem does have hard instances, it isn't much use if there is only one or a very small number of examples. It is therefore tempting to use something like average-case complexity which, as the name suggests, will give an indication of on average how hard a type of problem is. The issue here is the fact that in statistics it is well known that the mean value is rather sensitive to outliers. Just one example of an extremely hard instance of a problem may cause the average to come across as still hard, as demonstrated in figure 1.2. As such we will look at generic complexity as our measure. This will discount outlying difficult problems and give us a better idea of the complexity of the majority of instances.
In order to gain confidence in the security of the protocol from the previous chapter, we want to show the complexity of the ideal membership problem is at least exponential. The algorithm for finding Gröbner basis, known as Mora's algorithm, is an iterative method made up of two main steps during each iteration. The first step is finding new candidates for the basis and adding them to a queue. The second step involves testing the polynomial next in the queue to see if it would be redundant to add to what will become the Gröbner basis.
Section 5.2 onwards will cover our approach to finding the generic complexity of the problem we based the protocol in chapter 4 on. In particular in section 5.2 we will outline the theorem for Gröbner bases in general. In sections 5.3 and 5.4 we will apply the theorem we established in 5.2 to the Gröbner basis we will be using throughout chapter 4.
It is very difficult to model the polynomials needed for the basis, we therefore will begin by focusing on one observation that can be made about the first step of Mora's algorithm. Following this, we will make an assumption about the second step however, we will discuss three different approaches that provide evidence that our assumption may well hold. The observation we make will put a lower bound on the number of polynomials that is still exponential. While we will be covering a particular example for this observation, we will make it clear how the proof can be applied to other examples and, in particular, which

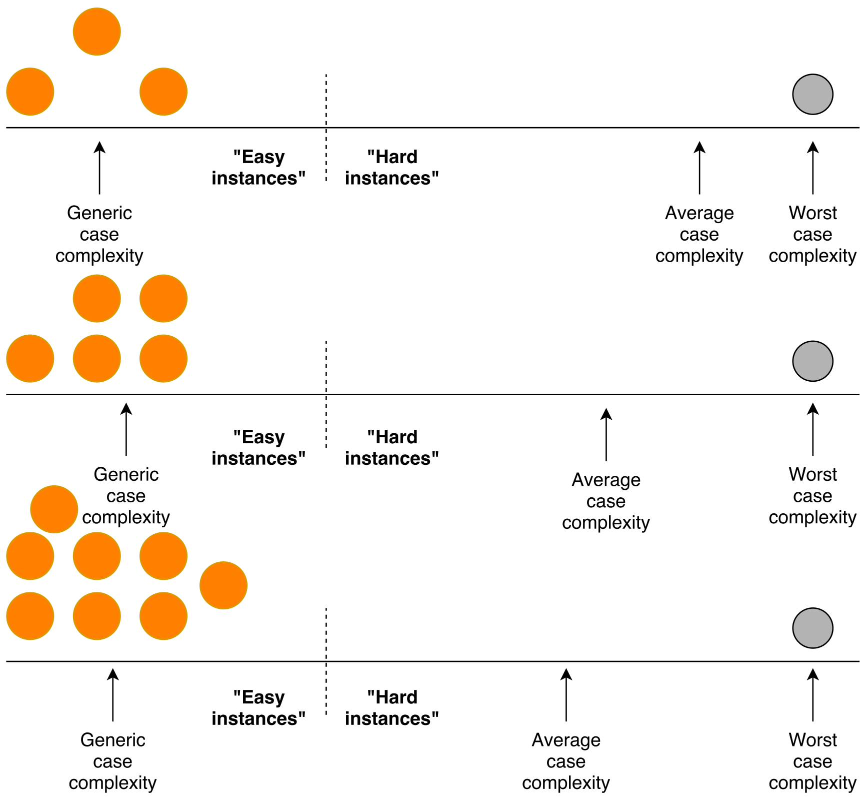


Figure 1.2: Comparison of measures of complexity

properties need to be there for the proof to hold. As for the assumption in step 2, we claim that a certain proportion of polynomials in the queue will be added to the basis. We try to back this claim up with the 3 following approaches:

- Simulations in Haskell,
- Re-ordering the queue of polynomials each time to better understand the structure,
- Classify the polynomials into one of 3 types and consider how the distribution of these 3 classes changes over time.

Chapter 2

Background material

2.1 Algebraic structures

Definition. A *binary operation* on a set S is a function $* : S \times S \to S$ that sends the ordered pair (a, b) to $(a * b) \in S$.

By imposing axioms on a set equipped with a binary operation we get the following definitions.

Definition. A *monoid* is a set M, with a binary operation $*$ such that the following conditions hold.

- Associativity: $m_1 * (m_2 * m_3) = (m_1 * m_2) * m_3$ for all $m_1, m_2, m_3 \in M$.
- Identity: There exists an element $e \in M$ such that for all $m \in M$, we have $m * e = e * m = m$. The element e is called the *identity* of M.

Associativity is an important and useful property to have when it comes to evaluating expressions on parallel machines. Figure 2.1 shows how the parse tree for an expression can be reduced by taking advantage of associativity [Kuck (1977)]. In the diagram, originally the first step adds together a and b, the second step adds c and the third step adds d. However, because c and d don't depend on a and b, two additions can be performed at the same time. Then, by the associative property, the result of these sums can be added together to get the same result as before one step earlier.
The *free monoid* on a set is the monoid whose elements are words made up of zero or more elements from that set. The binary operation on this particular monoid is a concatenation of words, with the empty word made up of zero elements, denoted e, as the identity.

Definition. A *group* is a monoid G, with identity e and with the following further property.

- Inverse: For each element $g \in G$, there exists an element $g^{-1} \in G$ such that $g^{-1} * g = e = g * g^{-1}$. The element g^{-1} is called the *inverse* of g.

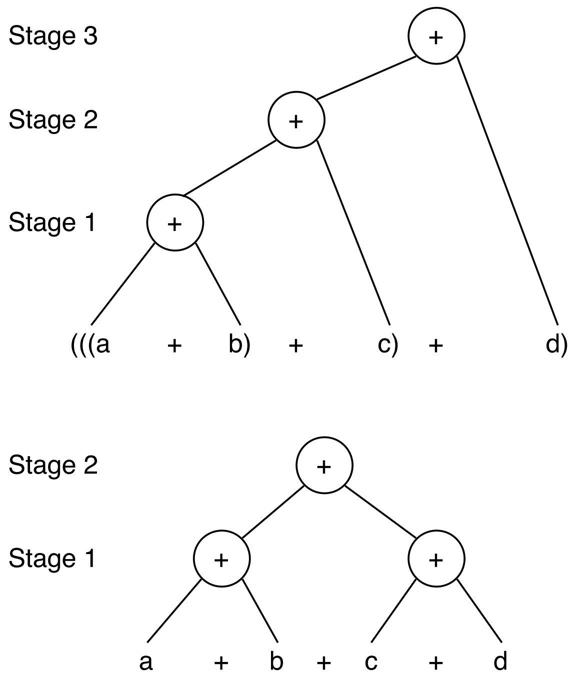


Figure 2.1: Tree reduction

Furthermore, we say that a group is abelian if for all $g_1, g_2 \in G$ we have that $g_1 * g_2 = g_2 * g_1$.

Definition. A *ring* is a set R with 2 binary operations $+$ and $\times$, which we call addition and multiplication, with an identity element 0 for addition alongside the following axioms.

- R is an abelian group with respect to addition.
- $(r_1 \times r_2) \times r_3 = r_1 \times (r_2 \times r_3)$ for all $r_1, r_2, r_3 \in R$.
- Distributivity: $r_1 \times (r_2 + r_3) = (r_1 \times r_2) + (r_1 \times r_3)$ and $(r_1 + r_2) \times r_3 = (r_1 \times r_3) + (r_2 \times r_3)$.
- Unit: There is a unique element 1 such that $1 \neq 0$ and $1 \times r = r = r \times 1$ for all $r \in R$.

Furthermore we say a *commutative ring* has all the above properties alongside

- $r_1 \times r_2 = r_2 \times r_1$.

The group structure brings with it one of the most important problems in modern cryptography.

Definition. *Discrete logarithm problem.* Given a group G, let $\langle g \rangle$ be the cyclic subgroup generated by $g \in G$ and $a \in \langle g \rangle$, find an integer x such that

$$g^x = a \tag{2.1}$$

2.2 Commutative and non-commutative polynomial rings

2.2.1 Monomials and orderings

Definition. The set of *monomials* in the variables $\{x_1, x_2, \dots, x_n\}$ is defined in the 2 following ways:

- Commutative: $\{x_1^{\beta_1} x_2^{\beta_2} \cdots x_n^{\beta_n} | \beta_i \in \mathbb{N} \cup \{0\}\}$,
- Non-Commutative: $\{x_{i_1} x_{i_2} \cdots x_{i_k} | i_1, i_2, \dots, i_k \in \{1, 2, \dots, n\}, k \in \mathbb{N} \cup \{0\}\}$, namely elements of the free monoid on the set $\{x_1, x_2, \dots, x_n\}$,

where the standard conventions of $x_i^1 = x_i$, $x_i^a x_i^b = x_i^{a+b}$ apply.

For testing purposes, implementation of polynomial rings were created in Haskell. The differences between the commutative and non-commutative versions become very clear here. The commutative monomials are described with just a list of integers representing the exponents, whereas the non-commutative monomials require lists of tuples made up of the variable and a list of its positions in the monomial.
Example The following are examples of how the monomial $x_1 x_3^2 x_2 x_1 \in \{x_1, x_2, x_3, x_4\}$ in the 2 implementations of monomials in Haskell:

- Commutative: $[2, 1, 2, 0]$,
- Non-commutative: $[(x_1, [1, 5]), (x_2, [4]), (x_3, [2, 3])]$.

Note here that in the commutative case we have collected all the variables with the same index together by moving the x_1 at the end of the monomial to the front. Also in the non-commutative version, we have not included an x_4 in the definition at all as it doesn't appear in the monomial, something we can't do with the commutative case.
The problem we will be dealing with in a later chapter requires us to have an ordering on our collection of monomials. While there are many orderings to choose from. The following definitions will be a common theme.

Definition. A set X is *totally ordered* under $\leq$ if the following properties hold for any $x_i, x_j, x_k \in X$:

- Antisymmetry: $x_i \leq x_j$ and $x_j \leq x_i \Rightarrow x_i = x_j$,
- Transitivity: $x_i \leq x_j$ and $x_j \leq x_k \Rightarrow x_i \leq x_k$,
- Totality: $x_i \leq x_j$ or $x_j \leq x_i$.

Definition. An *admissible order* is a total order on the set of monomials satisfying the following property:

$$x_i \leq x_j \Rightarrow x_i x_k \leq x_j x_k, \text{ for all monomials } x_i, x_j, x_k. \tag{2.2}$$

Often these orders are required to be well-orders as well. If we have finite variables, then the conditions are:

- The order is a total order
- For any monomial x_i, we have $1 \leq x_i$.

For the following definitions assume that the 2 monomials, $m_1, m_2 \in \{x_1, x_2, \ldots, x_n\}$, being compared are represented in the form

$$m_1 = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}, m_2 = x_1^{\beta_1} x_2^{\beta_2} \cdots x_n^{\beta_n}, \tag{2.3}$$

for the commutative case. The non-commutative case will be of the form

$$m_1 = x_{i_1} x_{i_2} \cdots x_{i_k}, m_2 = x_{j_1} x_{j_2} \cdots x_{j_{k'}}. \tag{2.4}$$

For each definition we will assume the order on the variables is $x_1 > x_2 > \cdots > x_n$.

Definition. Under the *lexicographical ordering*, $m_1 < m_2$ if

- (Commutative) $\exists i \leq n$ with $\alpha_i < \beta_i$, and then $\alpha_j = \beta_j$ for $1 \leq j < i$,
- (Non-commutative) Either $\exists l' \leq min\{k, k'\}$ with $\alpha_{i'_l} < \beta_{j'_l}$ and then $\alpha_{i_l} = \beta_{j_l}$ for $1 \leq l < l'$, or $k < k'$, and then $\alpha_{i_l} = \beta_{j_l}$ for $1 \leq l \leq k$.

The commutative definition is admissible whereas the non-commutative definition is not admissible (e.g. let $x_1 > x_2$, then $x_1 < x_1x_2$ but $x_1^2 > x_1x_2x_1$).

Definition. Under the *degree lexicographical ordering*, $m_1 < m_2$ if

- (Commutative) $deg(m_1) = \sum_{i=1}^n \alpha_i < deg(m_2) = \sum_{i=1}^n \beta_i$ or $deg(m_1) = deg(m_2)$ and $m_1 < m_2$ under the lexicographical ordering,
- (Non-commutative) $deg(m_1) = k < deg(m_2) = k'$ or $deg(m_1) = deg(m_2)$ and $m_1 < m_2$ under the lexicographical ordering.

Both of these definitions are admissible.

Definition. Under the *degree reverse lexicographical ordering*, $m_1 < m_2$ if

- (Commutative) $deg(m_1) < deg(m_2)$ or $deg(m_1) = deg(m_2)$ and $\alpha_i > \beta_i$ with all following exponents are equal ($\alpha_j = \beta_j$, $i < j \leq n$),
- (Non-commutative) $deg(m_1) < deg(m_2)$ or $deg(m_1) = deg(m_2)$ and working from right to left of both words, the first letter where m_1 and m_2 differ, say $i_1$and i_2 respectively, is such that $i_1 > i_2$.

Neither of these definitions are admissible.

2.2.2 Ideals

Definition. If I is a subset of a commutative ring $(R, +, \cdot)$, it is called an *ideal* of R if:

- $(I, +)$ is a subgroup of $(R, +)$
- $\forall x \in I, \forall r \in R : rx \in I$

Definition. If I is a subset of a non-commutative ring $(R, +, \cdot)$, it is called a *left (right) ideal* of R if:

- $(I, +)$ is a subgroup of $(R, +)$
- $\forall x \in I, \forall r \in R : rx \in I (xr \in I)$

Definition. If I is a subset of a non-commutative ring $(R, +, \cdot)$, it is called a *two sided ideal* of R if:

- $(I, +)$ is a subgroup of $(R, +)$
- $\forall x \in I, \forall r_1, r_2 \in R : r_1 x r_2 \in I$

Definition. Given a two sided ideal I of a ring R, an *equivalence relation* $\sim$ on R, for $a, b \in R$, is defined as

$$a \sim b \text{ if and only if } a - b \in I. \tag{2.5}$$

Definition. Given a ring R, an ideal I in R and $a \in R$, the *equivalence class* of a in R is

$$[a] = a + I = \{a + r | r \in I\}. \tag{2.6}$$

Definition. Given an ideal I in R, the set of all equivalence classes called the *quotient ring* denoted R/I. It in itself is a ring with addition and multiplication defined as

$$\begin{gathered}(a + I) + (b + I) = (a + b) + I, \\ (a + I)(b + I) = (ab) + I,\end{gathered} \tag{2.7}$$

where $a, b \in R$.

2.2.3 Leading terms

Polynomials will be a key structure used in our encryption methods and thus it is important to understand some of the key properties of them. Let K be a field, let M be a monoid. The *monoid ring* of M over K is the set $K\langle M\rangle$ of all elements of the form $\sum_{w \in M} c_w w$ with $c_w \in K$ and $c_w \neq 0$ for only finitely many $w \in M$, together with the addition $+$ defined by

$$\sum_{w \in M} c_w w + \sum_{w \in M} c'_w w = \sum_{w \in M} (c_w + c'_w) w \tag{2.8}$$

and the multiplication $\cdot$ defined by

$$\sum_{u\in M} c_u u \cdot \sum_{v\in M} c_v v = \sum_{w\in M} (\sum_{uv=w} c_u c_v) w. \tag{2.9}$$

Given a set X, the monoid ring of $\langle X\rangle$ over K is called the *non-commutative polynomial ring* [Xiu (2012)]. Furthermore, we define the *commutative polynomial ring* as the quotient ring of the non-commutative polynomial ring $K\langle X\rangle/I$ where $I = \langle uv = vu \ \forall u, v \in X\rangle$.

A lot of the algorithms described later are interested with the various leading parts of polynomial. The key components are given here [Adams *et al.* (1994)].

Definition. Suppose 2 polynomials, the first commutative, the second non-commutative, of the form

$$\begin{aligned} f &= a_1 x_1^{\beta_{1,1}} x_2^{\beta_{1,2}} \cdots x_n^{\beta_{1,n}} + \cdots + a_m x_1^{\beta_{m,1}} x_2^{\beta_{m,2}} \ldots x_n^{\beta_{m,n}}, \\ g &= a_1 x_{i_{1,1}} x_{i_{2,1}} \cdots x_{i_{k_1,1}} + \cdots + a_m x_{i_{1,m}} x_{i_{2,m}} \cdots x_{i_{k_m,m}}, \end{aligned} \tag{2.10}$$

under some order, have there terms expressed in descending order. We define

- $\mathrm{LM}(f) = x_1^{\beta_{1,1}} x_2^{\beta_{1,2}} \cdots x_n^{\beta_{1,n}}$, the *leading monomial* of f,
  $\mathrm{LM}(g) = x_{i_{1,1}} x_{i_{2,1}} \cdots x_{i_{k_1,1}}$, the *leading monomial* of g,
- $\mathrm{LC}(f) = \mathrm{LC}(g) = a_1$, the *leading coefficient* of f and g,
- $\mathrm{LT}(f) = a_1 x_1^{\beta_{1,1}} x_2^{\beta_{1,2}} \ldots x_n^{\beta_{1,n}}$, the *leading term* of f,
  $\mathrm{LM}(g) = a_1 x_{i_{1,1}} x_{i_{2,1}} \cdots x_{i_{k_1,1}}$, the *leading term* of g.

2.2.4 Building Hecke algebras

In chapter 4, we will be interested in a particular algebraic structure, namely the Hecke algebra. We will give the details of the algebra here and use this to improve on an encryption scheme later.

Definition. A *Coxeter system* is a pair (W,S), where $S = \{s_1, s_2, \ldots, s_r\}$ are generators of a *Coxeter group W* defined by the presentation

$$\langle s_1, s_2, \ldots, s_r | (s_i s_j)^{m_{i,j}} = 1\rangle, \tag{2.11}$$

where $m_{i,i} = 1$, $\forall i \in \{1, \ldots, r\}$, $m_{i,j} \geq 2$ for $i \neq j$, and can equal ∞.

The Bruhat ordering is a partial order on the elements of a Coxeter group.

Definition. If (W, S) is a Coxeter with generators S, then the *(strong) Bruhat order* is the partial order on the group W, defined by $u \leq v$ if some substring of some reduced word for v is a reduced word for u. A reduced word for an element of W is a minimal length expression of the element as a product of elements of S.

Another important group for this section is the Braid group which has a standard presentation defined as follows.

Definition. The *braid group* B_n, for any $n \geq 2$ is defined by the following presentation [Artin (1947)]

$$\left\langle \sigma_1, ..., \sigma_{n-1} \;\middle|\; \begin{array}{c} \sigma_i\sigma_j = \sigma_j\sigma_i, \text{ for } |i-j| \geq 2 \\ \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}, \text{ for } 1 \leq i \leq n-1 \end{array} \right\rangle \tag{2.12}$$

Several problems arising from presentations of groups have been suggested as the premise for cryptosystems. One in particular of interest is the conjugacy search problem.

Definition. (*Conjugacy search problem*) Given conjugate elements $u, w \in B_n$, find $v \in B_n$ such that

$$w = v^{-1}uv. \tag{2.13}$$

There have been protocols already suggested based on conjugacy search problems in braid groups. For example, a scheme based on the Diffie Hellman conjugacy problem has been presented, where the hardness of the brute force attack is proportional to $exp\{\frac{1}{2}pn\log(n)\}$, where the size of a plaintext is $pn\log(n)$ bits [Ko *et al.* (2000)]. For this particular scheme though, polynomial time attacks exist such as in [Cheon & Jun (2003)]. Although schemes based around these conjugacy problems have been broken, it is still thought to be an open problem if the conjugacy search problem can be solved in general for braid groups.
This problem leads to another similar problem which will help to justify the choice of encryption function in chapter 4.

Definition. (*Multiple conjugacy search problem*) Given m pairs of conjugate elements $(u_1, w_1), \ldots, (u_m, w_m) \in B_n$ which are all conjugated by the same element. Find $v \in B_n$ such that

$$w_i = v^{-1}u_iv, \forall i \in \{1, \ldots, m\}. \tag{2.14}$$

From a Coxeter group W we can form a *Hecke algebra* [Bump (2010)]. This is an algebra over the ring of Laurent polynomials $\mathbb{Z}[q, q^{-1}]$. This algebra is denoted $\mathcal{H}_q(W)$. Much like the Coxeter group, given the Coxeter system (W, S), where $|S| = r$ then the Hecke algebra has generators $T_1, T_2, \ldots, T_r$. For a generator T_i, the i represents the Coxeter generator s_i from the Coxeter group W. If $w = s_{i_1}s_{i_2}\cdots s_{i_n}$, where $s_{i_j} \in S^{\pm 1}$, is a reduced word in W then, we define $T_w = T_{i_1}T_{i_2}\cdots T_{i_n}$. The generators are subject to the relation

$$(T_iT_j)^{m_{i,j}} = T_iT_jT_i\cdots = T_jT_iT_j\cdots = (T_jT_i)^{m_{i,j}}, \tag{2.15}$$

where there are $m_{i,j}$ copies of both T_i and T_j on each side. The relation $s_i^2 = 1$ in the

Coxeter group is replaced with the *quadratic relation*

$$T_i^2 = (q-1)T_i + q. \tag{2.16}$$

If $w = w_1 w_2$ in W and $length(w) = length(w_1) + length(w_2)$ then $T_w = T_{w_1} T_{w_2}$.

2.3 Symmetric and Asymmetric encryption techniques

It is common practice to make public the encryption method used on data. If Alice and Bob wish to encrypt messages in such a way that the other can decrypt, they can agree on the scheme used in an open channel. The key that is used in the encryption function, however, must be kept secret otherwise any ciphertext produced will be immediately insecure. A *key exchange* protocol is used to allow Alice and Bob to communicate a shared secret key despite using an open channel.
Figure 2.2 gives an overview of how key exchanges are performed. Alice and Bob each begin by choosing a secret piece of information P_A and P_B respectively. They combine their information with some publicly known information P_k and send each other the results i.e. Bob receives $P_k(P_A)$ and Alice receives $P_k(P_B)$. Finally, they combine their secret key with the information they have received, resulting in the shared key $P_k(P_A, P_B)$. It should be difficult for an eavesdropper to calculate $P_k(P_A, P_B)$ based on $P_k(P_A)$ and $P_k(P_B)$, which can be seen by the public.

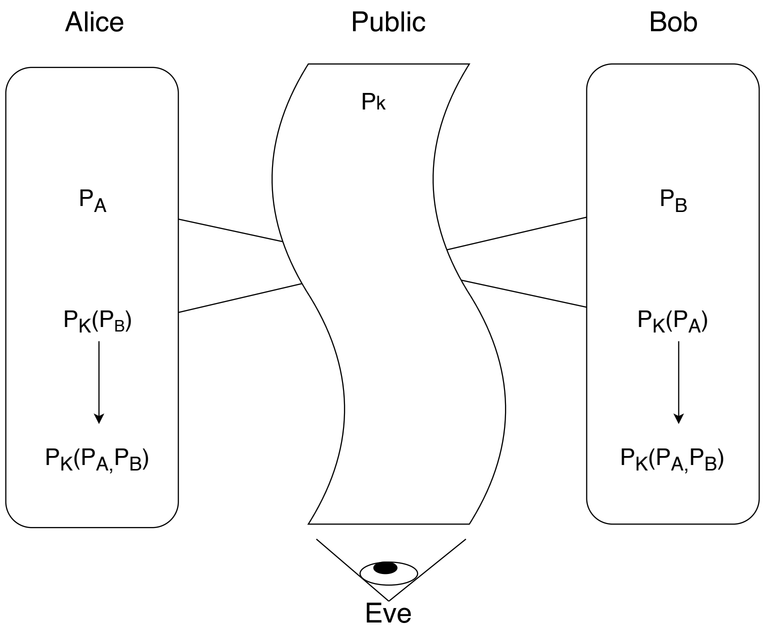


Figure 2.2: Key exchange process

A well-known example is the Diffie-Hellman key exchange. It is performed with the following method:

1. Alice and Bob publicly agree on two prime numbers g and p, where g is a primitive root of p (every value from 1 to $p-1$ is congruent to a power of g mod p).

2. Alice chooses in secret a value a and from that calculates $A = g^a \bmod(p)$. She then sends A over the public channel to Bob. Bob does the same with b and $B = g^b \bmod(p)$.

3. Alice can now calculate $B^a \bmod(p) = g^{ba} \bmod(p)$ which is equal to $A^b \bmod(p) = g^{ab} \bmod(p)$ that Bob can calculate.

It is thought to be difficult for an eavesdropper to find $g^{ab} \bmod(p)$ given the public information g, p, $g^a \bmod(p)$ and $g^b \bmod(p)$.
The term *symmetric* used to describe the previous method refers to the fact that Alice and Bob have the same information at the end of the process. They are therefore able to encrypt and decrypt messages under the same scheme. *Asymmetric* schemes differ in that one party may be able to encrypt and decrypt but, one or more other parties are only able to encrypt. Figure 2.3 gives an example of multiple users encrypting their data and sending it to the service. The service is able to decrypt the information but if the users see any of the ciphertexts, they should not be able to decrypt it.

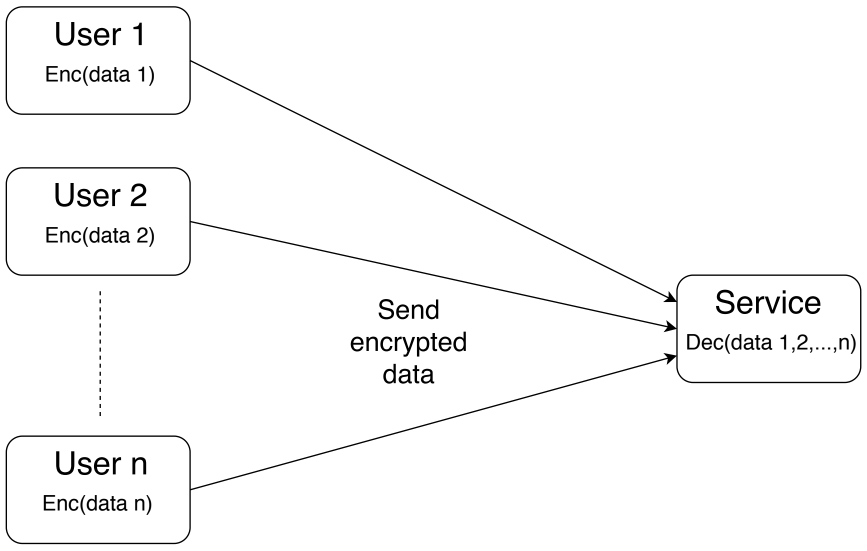


Figure 2.3: Multiple user public key encryption

The *RSA encryption* method (named after its creators Rivest, Shamir, Adleman), is a well known and widely used example of an asymmetric encryption scheme. The public key is a modulus n and an exponent e both positive integers. The modulus $n = pq$ where p and q are primes. The ciphertext c is generated from a plaintext m by the encryption function E defined as

$$c = E(m) = m^e (mod\ n). \tag{2.17}$$

Here n is the product of two primes p and q, both of which are kept secret. We choose a value of e that satisfies both $1 < e < \lambda(n)$ and $gcd(e, \lambda(n)) = 1$ where $\lambda(n)$ is the smallest positive integer m such that

$$a^m \equiv 1\ (mod\ n), \tag{2.18}$$

for every integer a from 1 to n that is co-prime to n. The function is implemented efficiently using the method of repeated squaring.
The decryption function, D, on a ciphertext c is defined by

$$m = D(c) = c^d (mod\ n), \tag{2.19}$$

where $ed \equiv 1(mod\ \lambda(pq))$.
Without knowledge of the private key, an attacker needs to be able to factorise n to break the cryptosystem. If an attacker can find p and q such that $n = pq$ then they can easily calculate $\lambda(n) = (p-1)(q-1)$. With knowledge of $\lambda(n)$, it is easy to find d using the Euclidean algorithm. Currently, in classical computing, it is believed that prime factorisation does not have an efficient algorithm to solve it, thus RSA is considered secure.

2.4 Homomorphic encryption

Definition. Let (G, $*$) and (H, $\circ$) be groups. A function $f : G \mapsto H$ is a *homomorphism* if $\forall a, b \in G$,

$$f(a * b) = f(a) \circ f(b). \tag{2.20}$$

Furthermore, we say that we have an isomorphism if the homomorphism is bijective i.e. it is injective (1 to 1) and surjective (onto). If we have an isomorphism $f : G \mapsto H$, then we write $G \cong H$.

A lot of work has been carried out in group theoretic homomorphic encryption. For example, the homomorphic property in RSA is satisfied by using the commutativity of integer multiplication.

$$\begin{aligned} E(m_1) \cdot E(m_2) &= m_1^e \cdot m_2^e (mod\ n) \\ &= (m_1 \cdot m_2)^e (mod\ n) \\ &= E(m_1 \cdot m_2). \end{aligned} \tag{2.21}$$

While the homomorphic property of RSA does appear to be a useful property of an already well respected encryption system, it is worth thinking about real world restrictions of such a property would be. To see this consider the following function

Definition. Given $c = m^e (mod\ n)$

$$HALF(c) = \begin{cases} 0, & \text{if } 0 \leq m < n/2 \\ 1, & \text{if } n/2 \leq m < n-1. \end{cases} \tag{2.22}$$

The reason there is interest in a definition such as this is that it can be used to show that the homomorphic property of RSA can be seen as a double edged sword. The homomorphic property of the RSA means that if we have access to an oracle for $HALF()$ then, given $c = me$ and e we can compute the following:

$$\begin{aligned} HALF(m^e) = 0 &\Leftrightarrow m \in [0, \frac{n}{2}) \\ HALF(2m^e) = 0 &\Leftrightarrow m \in [0, \frac{n}{4}) \cup [\frac{n}{2}, \frac{3n}{4}) \\ HALF(4m^e) = 0 &\Leftrightarrow m \in [0, \frac{n}{8}) \cup [\frac{n}{4}, \frac{3n}{8}) \cup [\frac{n}{2}, \frac{5n}{8}) \cup [\frac{3n}{4}, \frac{7n}{8}) \\ &\vdots \end{aligned} \tag{2.23}$$

Using such a binary search approach means that it is possible to establish the value of m in polynomial time, by computing values of $HALF(2^i me)$ for various i. Fortunately, there is currently no well known polynomial time algorithm to implement HALF, but the point here is that there are lots of potential ways for information about messages to be leaked and it would only take one to be established in reality for there to be a major security concern.

Alongside RSA, other encryption methods are homomorphic under a single operation. For example, the Paillier encryption method is homomorphic under addition. A more desirable property for an encryption method to have is a ring homomorphism.

Definition. Given two rings, R and S, a *ring homomorphism* is a function $f : R \to S$ such that

- $f(a + b) = f(a) + f(b) \forall a, b \in R.$
- $f(ab) = f(a)f(b) \forall a, b, \in R.$
- $f(1_R) = 1_s$, the multiplicative identity in each ring.

2.4.1 Fully homomorphic encryption

The ultimate goal of homomorphic cryptography is to be able to perform any operation on encrypted data homomorphically. This is known as *Fully homomorphic encryption.* Ideally, these operations on ciphertext would be performed with as little extra effort as it would have been to perform on the original ciphertext. In order to achieve this, we would

like to have a collection of operations that can be built up into any given operation.
Gentry proposed in his paper [Gentry (2009)], an encryption method that would allow the evaluation of arbitrary circuits. His protocol allowed evaluation of addition and multiplication gates on encrypted data in a ring. The reason this is so important is that we are able to construct Boolean algebra. The values in this algebra may be identified with integer arithmetic modulo 2. Here addition plays the Boolean role of XOR and multiplication plays the Boolean role of AND. Computers perform all operations through a series of logic gates, where each gate performs a Boolean operation. Therefore, any operation we wish to perform on any data will be an application of the Boolean algebra. This is why finding a ring theoretic encryption method is of such value.

2.5 Implementation considerations

2.5.1 Classical computation

Different types of attacks

Not every potential attacker will be looking for a full break of a cryptosystem. Even without knowledge of the secret key, there may still be important information to be learned from various aspects of the protocol.
A *chosen-plaintext attack* is a type of attack that tries to gain information about an encryption scheme assuming a given arbitrary plaintext, they can find the corresponding ciphertext.
A *known-plaintext attack* is a type of attack that tries to gain information using the fact that the attacker knows the plaintext version of a particular ciphertext. This is a weaker attack than the previous one as a chosen-plaintext attack doesn't have to wait for a specific ciphertext to appear naturally.
A *chosen-ciphertext attack* is a type of attack that tries to gain information by having the attacker learn the decryption of a chosen ciphertext.

Optimal asymmetric encryption padding

Definition. A *trapdoor function* is a function that is easy to compute but the inverse is believed to be difficult to compute.

RSA is thought to be a trapdoor function. Bellare and Rogaway put forward a scheme in the 90's which could convert any trapdoor function into an encryption scheme [Manger (2001)]. RSA led the way with this method resulting in the RSA-OAEP scheme, this provides more security advantages than just the standard RSA scheme. We will discuss the scheme here and use f to represent the trapdoor function, so one could easily substitute RSA in here.

Setup

Here we will assume that the data is in binary format. Our trapdoor function $f : \{0,1\}^k \to \{0,1\}^k$, has an inverse g. We also require two other parameters k_0 and k_1 which are sufficiently large but satisfy $k_0 + k_1 < k$. Now although our trapdoor function acts on k bits, the scheme itself encrypts messages of the form $m = \{0,1\}^n$, where $n = k - k_0 - k_1$. Two arbitrary functions are also incorporated

$$H : \{0,1\}^{n+k_1} \to \{0,1\}^{k_0} \text{ and } G : \{0,1\}^{k_0} \to \{0,1\}^{n+k_1} \tag{2.24}$$

Encryption

If we wish to encrypt a plaintext m, we begin by randomly generating $r \in \{0,1\}^{k_0}$. We then proceed to calculate

$$\begin{aligned} s &= G(r) \oplus (m||0^{k_1}) \in \{0,1\}^{n+k_1} \\ t &= H(s) \oplus r \in \{0,1\}^{k_0} \\ w &= s||t \in \{0,1\}^k \\ c &= f(w) \in \{0,1\}^k \text{ is the ciphertext} \end{aligned} \tag{2.25}$$

where $||$ represents concatenation of bits.

Decryption

$$\begin{aligned} w &= g(c) \in \{0,1\}^k \\ s &= w[0,\dots,n+k_1-1] \in \{0,1\}^{n+k_1} \\ t &= w[n+k_1,\dots,k] \in \{0,1\}^{k_0} \\ r &= H(s) \oplus t \in \{0,1\}^{k_0} \\ z &= G(r) \oplus s \in \{0,1\}^{n+k_1} \\ m &= z[0,\dots,n-1] \in \{0,1\}^n \text{ is the plaintext} \\ y &= z[n,\dots,n+k_1-1] \in \{0,1\}^{k_1} \end{aligned} \tag{2.26}$$

where the values in [ ] represent the bit positions of the variable in front of the brackets. These added steps provide us with an added level of security. Whilst the trapdoor function should prevent outsiders from finding the plain text information, there may be a goal of simply tampering the data. This could be done if the adversary just had access to the ciphertext data by changing some of the 1's to 0's and vice versa. In classic RSA, when decrypted this, in theory, would just output a different value where anyone without knowledge of the plaintext would simply accept this as being the truth. However, in RSA-OAEP alongside the plaintext that is returned, we also get back y. Now we expect this to be a string of k_1 0's so if that is not what we observe then we know that the ciphertext has been changed in some way.

Computation limitations

In almost any scenario, but in particular, for IOT, we want to keep computational costs and bandwidth use as low as we can. As such it is highly recommended that symmetric key algorithms are used for encrypting data as asymmetric schemes have relatively heavy computational costs. As the number of entities in a system grows, it becomes exceedingly difficult to distribute shared secret keys manually. Therefore we need support from automated key-establishment schemes. This is where Asymmetric encryption schemes shine, as demonstrated in figure 2.4. The public part of a key pair generated in this scheme can be used by another party wishing to establish a secret key with the key pair owner. The other party encrypts their choice of shared secret key using the public key and the key pair owner is the only who can retrieve that information with their private key. One of the NIST-approved asymmetric schemes is RSA. Any RSA key-pair generated will have a modulus n, with a length of either 2048 or 3072 bits, where n is the product of two primes. The public exponent, e, will be an odd integer lying in the range $65537 \leq e < 2^{256}$.

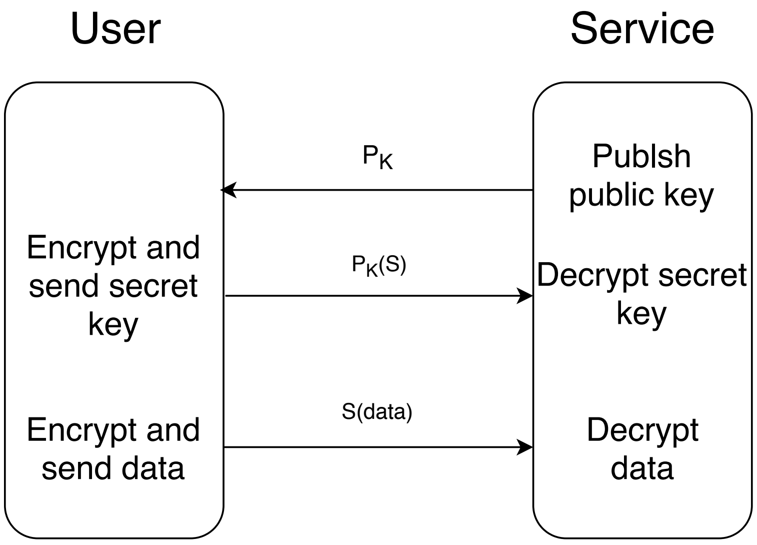


Figure 2.4: Mixing public and private key methods

The following two algorithms are the NIST-approved choices for encryption.
The *Advanced Encryption Standard* (AES) algorithm is a symmetric block cipher that can process blocks of 128 bits with use of cipher keys of length either 128, 192 or 256 bits. Any implementation of AES must support at least one of those 3 key lengths.
Triple Data Encryption Algorithm (TDEA) encrypts and decrypts data in 64-bit blocks using 56-bit keys. Although there are 2 variants of these methods, the use of two-key TDEA is no longer approved (three-key TDEA being the approved alternative).

2.5.2 Quantum computation

A qubit can be in one of two states $|0\rangle$ and $|1\rangle$ or it can be in a *superposition* of states represented by the linear combination

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \text{ where } \alpha, \beta \in \mathbb{C}. \tag{2.27}$$

The values α and β represent probabilities of what we will observe when we measure the qubit. We get a value of 0 with probability $|\alpha|^2$ and a value of 1 with probability $|\beta|^2$, thus giving us the condition $|\alpha|^2 + |\beta|^2 = 1$. The states $|0\rangle$ and $|1\rangle$ are known as the *computational basis states.*
A qubit in the state

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{2.28}$$

when measured, has a 50/50 change of being a 0 or a 1.
Suppose we have 2 qubits. A 2 qubit system has 4 computational basis states denoted $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ and a superposition of these states can be written as

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2.29}$$

where the coefficients are normalized in the same way as done with one qubit. An important 2 qubit state is the *Bell state* or *EPR pair*

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.30}$$

The interesting property about this state is that if we measure the first qubit, we get the value of 0 with a probability of a half (same as 1) but from this point the complete state has been decided, since the first qubit can only be followed by the same value.

There is a quantum version of the NOT gate in classical computing. In classical computing, this sends 0 to 1 and vice versa. We treat this the same in the quantum world, just the coefficients for 0 and 1 states are swapped. We define the matrix X to be the quantum *NOT* gate

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.31}$$

which is multiplied to our quantum state $\alpha|0\rangle + \beta|1\rangle$ using the normal matrix and vector multiplication by representing our state as

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{2.32}$$

Surprisingly the only requirement on quantum gates is that the matrix representation must be unitary. A matrix U is *unitary* if $U^\dagger U = I$ (the dagger notation represents the adjoint of the matrix, namely the complex conjugate of the transpose). Two other important gates are

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.33}$$

known as the Z gate, alongside the *Hadamard* gate

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.34}$$

Finally, two gates that will be of particular importance in this work will be the CNOT gate and the Toffoli gate. Firstly the *CNOT* gate is defined as

$$CNOT \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.35}$$

The first qubit is fixed and acts as a control variable. If it is a 1 the second qubit swaps its value (0 becomes 1 or 1 becomes 0).
The *Toffoli* gate is defined as

$$\mathsf{Toffoli} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.36}$$

Similar to the CNOT gate, the first two qubits are fixed and act as control variables, the third qubit is swapped if both the control qubits are equal to 1. As you can see here, the matrix transformations used can get large quickly. In the quantum world, there are various proofs that as few as 2 quantum gates is enough to perform any calculation up to a particular desired accuracy [Nielsen & Chuang (2010)].

Chapter 3

Quantum cryptography and Post-quantum cryptography

In this chapter, we will consider how quantum computing can be used to change what cryptographic tools we have at our disposal. The first section will focus on existing work that has looked at a post-quantum world. This is compiled of work by; Shor, on attacking the widely used RSA, work by Bennett and Brassard on using quantum technology to form a key exchange and work by Goldreich Goldwasser and Halevi on a cryptosystem that is more resistant to quantum attacks than something like RSA.
Section 2 will begin with an important quantum circuit by Takahashi. This will enable the remainder of the original work in the section which looks to improve the homomorphic properties of the quantum-resistant method previously mentioned.
Sections 3 and 4 are built off the foundations set in section 2 and aim to improve the method to be more practical.

3.1 Effects of quantum methods on cryptography

In this section, we will look at some of the ways we already know quantum computers will change cryptography. Firstly, we will see how a widely used problem becomes insecure with the aid of quantum algorithms. After which, we will see how other algorithms, both quantum and classical, will have a major interest in them. This interest may stem from the opportunity to use these new computers (in the case of quantum algorithms), or the need to replace insecure algorithms (in the case of classical algorithms).

3.1.1 Classical methods are vulnerable to quantum attacks

The discrete logarithm problem has been utilised widely within modern public-key cryptography as the basis of security [McCurley (1990)]. While it has become the standard for many real-world security systems, there are limitations. Firstly, while encryption methods such as RSA, that exist in the ring of integers containing $(\mathbb{Z}, \cdot)$, are homomorphic with respect to multiplication, the addition operation won't perform correctly. We see that in almost every choice of e and n that

$$E(m_1 + m_2) = (m_1 + m_2)^e (\text{mod n}) \neq m_1^e + m_2^e \ (\text{mod n}) = E(m_1) + E(m_2). \tag{3.1}$$

This is likely to be the case with similar cryptosystems that come from abelian groups which leads our search to other algebraic structures. In this work we will consider vector spaces and rings for fully homomorphic encryption, however, [Ostrovsky & Skeith (2008)] proves that constructing an FHE over a ring with identity is equivalent to constructing a group homomorphic encryption over a finite non-abelian simple group. The second issue, however, is the major issue we need to address when considering quantum computers. This issue is the assumption that decryption is hard. RSA belongs to a collection of cryptosystems that base their security on the discrete logarithm problem, which we defined in chapter 2.
While the discrete logarithm problem isn't the only difficult problem forming a basis for these systems, there exists polynomial time algorithms that convert to problems such as factoring.
The development of Shor's algorithm has put cryptosystems that utilize problems like the discrete logarithm in jeopardy [Shor (1994)]. This is due to the efficiency of modular exponentiation using repeated squaring and the quantum Fourier transform, the latter of which, as the name suggests, may be performed using quantum algorithms.
This exploitable weakness and the possibility of quantum computers becoming available in our life leads us to look for so-called *quantum resistant* cryptosystems.
While Shor's algorithm may be one of the most prolific quantum attacks, there is evidence of other weaknesses in classical methods from quantum attacks. In classical computing, we consider an oracle that is able to obtain important details about a private key for an attacker and try to understand how a scheme may deal with the attacker using that information. However, In the quantum world an attacker may be able to use a quantum oracle i.e. they can query the oracle with a superposition of classical queries [Boneh *et al.* (2011)].

3.1.2 A key exchange protocol

Unlike traditional key exchange protocols that base their security on a problem thought to be hard, fundamental properties of quantum mechanics are used to keep data safe in quantum cryptography. In this section, we will look at a very early example from Bennett and Brassard [BENNETT (1984)].
Although it is typical to use the standard basis for working with qubits, there is nothing to stop Alice and Bob using another basis for communicating a secret key. In fact, the key

exchange protocol described here requires the use of 2 bases. The main requirement of these basis elements is that they are conjugate to one another. Although the terminology differs from that here, [Bennett *et al.* (1992)] describes the important property of conjugate bases as "any measurement of a single photon's rectilinear (0 vs 90 degrees) polarization randomizes its diagonal (45 vs 135 degrees) polarization, and vice versa".
Recall that in our standard basis, any qubit is represented as

$$\alpha|0\rangle + \beta|1\rangle, \tag{3.2}$$

where $|\alpha|^2 + |\beta|^2 = 1$. Upon measurement of this qubit we have a probability of $|\alpha|^2$ or $|\beta|^2$ of outputting a 0 or 1 respectively. Now consider an alternative basis, which we will call the *diagonal basis*, where any qubit can be represented as

$$\alpha'(\frac{|0\rangle + |1\rangle}{\sqrt{2}}) + \beta'(\frac{|0\rangle - |1\rangle}{\sqrt{2}}). \tag{3.3}$$

As with the standard basis, here we have a probability of $|\alpha'|^2$ or $|\beta'|^2$ of a measurement outputting the basis element $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ or $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Now suppose we have one of the diagonal basis elements but try to measure it using the standard basis. Expressed as a sum of the standard basis elements we have

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \text{ and } \frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}|0\rangle + \frac{-1}{\sqrt{2}}|1\rangle. \tag{3.4}$$

Notice that in both cases we have an equal probability of observing a 0 or a 1. We can do the same with standard basis elements measured in the diagonal basis.

$$|0\rangle = \frac{1}{\sqrt{2}}(\frac{|0\rangle + 1\rangle}{\sqrt{2}}) + \frac{1}{\sqrt{2}}(\frac{|0\rangle - |1\rangle}{\sqrt{2}}) \text{ and } |1\rangle = \frac{1}{\sqrt{2}}(\frac{|0\rangle + 1\rangle}{\sqrt{2}}) + \frac{-1}{\sqrt{2}}(\frac{|0\rangle - |1\rangle}{\sqrt{2}}). \tag{3.5}$$

Once again, in both cases, we have an equal probability of observing a 0 or 1. We call bases with this property conjugate. We will adopt the notation here that the standard basis elements $|0\rangle$ and $|1\rangle$ are written as the vectors $[1, 0]^t$ and $[0, 1]^t$ respectively. Also, the diagonal basis elements $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ are written as $[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^t$ and $[\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]^t$ respectively. $[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^t$ can be thought of as the 0 for this basis and $[\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]^t$ the 1.
Alice and Bob publicly agree to use the same two conjugate bases. Alice then chooses a sequence of 0's and 1's she wished to transmit. Qubits used to represent these values are sent to Bob over a quantum channel. Each 0 is encoded as either $[1, 0]^t$ or $[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^t$. Each 1 is encoded as either $[0, 1]^t$ or $[\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]^t$.
For each qubit received, Bob chooses at random, one of the two conjugate bases and takes a measurement. Alice does not publicly reveal anything about her choice of basis at this point. If Bob's choice of basis matches Alice's, then when he measures he will observe the value that Alice intended. If opposite bases are chosen, then as we saw above, Bob will

observe the 0 or 1 element from his basis with equal probability (i.e. he has a 50% chance of observing the value Alice intended).

Once all transmission has finished, Alice and Bob can publicly discuss which bases they used for each qubit. For a qubit where they have a match, they can confirm that have communicated successfully a 0 or 1. The qubits where they disagree are discarded. An example of this process is given in the following table. In the table, a $+$ is used to represent the standard basis whereas a $\times$ is used to represent the diagonal basis.

Alice message	0	1	0	1	0	1	0	1
Alice basis choice	$+$	$+$	$\times$	$\times$	$+$	$\times$	$+$	$\times$
Alice sends	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\end{bmatrix}$	$\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{-1}{\sqrt{2}}\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{-1}{\sqrt{2}}\end{bmatrix}$	$\begin{bmatrix}0\\1\end{bmatrix}$	$\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}$
Bob basis choice	$+$	$\times$	$+$	$\times$	$+$	$\times$	$\times$	$+$
Bob measurement	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{-1}{\sqrt{2}}\end{bmatrix}$	$\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{-1}{\sqrt{2}}\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$
Shared key?	0			1	0	1		

The question remains, what about outsiders trying to eavesdrop on this information exchange? Suppose an attacker tries to interfere before the qubits reach Bob. Much like Bob, the attacker must choose one of the two bases to measure in. Assume here that Bob and Alice are using the same basis as anything else would be thrown away and therefore useless to the attack anyway. If the attacker also happens to choose the same basis, then the qubit will have been measured and sent to Bob, whose measurement will essentially be a redundant step but it will be the intended value. If the non-matching basis is chosen, a measurement has a 50/50 chance of still giving the correct value. This means that if an attack takes place, the is only a 25% chance of each value in the shared secret sequence being incorrect.

As an example, suppose Alice sends a 0 in the standard basis i.e. $[1,0]^t$. With 50% probability, Eve also chooses to measure in the standard basis and so the exchange looks like this

$$\begin{aligned}&\text{Alice sends } [1,0]^t \to \text{Eve's measurement } [1,0]^t\\&\text{Eve sends } [1,0]^t \to \text{Bob's measurement } [1,0]^t.\end{aligned} \tag{3.6}$$

The other scenario is that Eve chooses to measure Alice's message using the diagonal basis. Eve, therefore, has two possible measurements. After forwarding the message on to Bob, who is using the standard basis, he has also has two potential measurements. The four potential scenarios are:

1.
$$\begin{aligned}&\text{Alice sends } [1,0]^t \to \text{Eve's measurement } [\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}]^t\\&\text{Eve sends } [\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}]^t \to \text{Bob's measurement } [1,0]^t.\end{aligned} \tag{3.7}$$

2.
$$\begin{aligned} &\text{Alice sends } [1,0]^t \rightarrow \text{Eve's measurement } [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^t \\ &\text{Eve sends } [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^t \rightarrow \text{Bob's measurement } [0,1]^t. \end{aligned} \tag{3.8}$$

3.
$$\begin{aligned} &\text{Alice sends } [1,0]^t \rightarrow \text{Eve's measurement } [\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]^t \\ &\text{Eve sends } [\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]^t \rightarrow \text{Bob's measurement } [1,0]^t. \end{aligned} \tag{3.9}$$

4.
$$\begin{aligned} &\text{Alice sends } [1,0]^t \rightarrow \text{Eve's measurement } [\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]^t \\ &\text{Eve sends } [\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]^t \rightarrow \text{Bob's measurement } [0,1]^t. \end{aligned} \tag{3.10}$$

Of these four possible outcomes, only two of them would give Bob reason to believe that the qubits have been tampered with.
For Bob and Alice to check if any eavesdropping has occurred, they will need to publicly compare the values they have at certain positions to see if any don't match, which will then need to be discarded. If there are differences then there has been some kind of attack and the process must be restarted. The problem here, however, comes to down the possibility that the values they choose may have been seen by an attacker but just so happened to remain correct. Choosing more and more values to compare will help decrease this probability but it would always be impossible to say for certain there has been no attack.

3.1.3 A post-quantum somewhat homomorphic encryption system

In this section, we look at what is known as a *somewhat homomorphic* encryption system (SHE). This class of cryptosystem allows encrypted data to be processed but each operation introduces some error. The number of additions that can be performed on the encrypted data, with correct decryption, is bounded as the error will grow too large to deal with. The scheme is work in this particular section is a brief introduction to the scheme presented in [Goldreich *et al.* (1997)]. Here we will study a problem that is thought to be difficult to solve even with quantum computers, followed by an encryption scheme built off that problem. Later in the chapter, we will look at an error correcting method to extend the number of additions that can be performed, We will do this by creating a superposition of all possible encryptions of a plaintext and manipulating this appropriately.

Closest vector problem

To study this problem, we first we must establish some important definitions. A *lattice* is a subset of $\mathbb{R}^d$ which we will denote as L, with basis $x_1, x_2, \ldots, x_d$, as the following

$$L = \{\sum_{i=1}^{d} a_i x_i | a_i \in \mathbb{Z}\}. \tag{3.11}$$

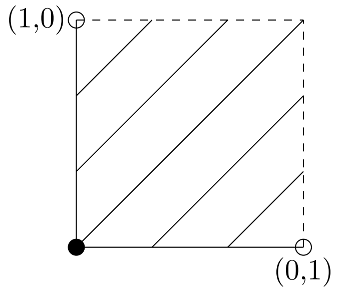
For a vector $v \in \mathbb{R}^d$, $||\mathbf{v}|| = \sqrt{v_1^2 + v_2^2 + \cdots + v_d^2}$ is the Euclidean norm. We will also need the following *dist* function

$$dist(x, L) = min\{||x - y|| \ : \ y \in L\}. \tag{3.12}$$

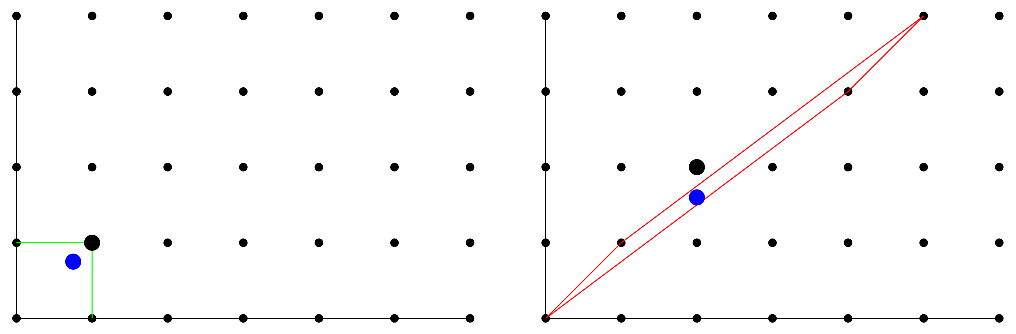
Our next step is to define an object that will allow us to further our understanding of lattices. If we take a lattice $\mathit{\Lambda} \in \mathbb{R}^d$ which has a basis $x_1, x_2, ..., x_d$. The set defined by:

$$\mathit{\Pi} = \{\sum_{i=1}^{d} \alpha_i x_i \mid 0 \leq \alpha_i < 1 \mid i = 1, ..., d\}, \tag{3.13}$$

is the *fundamental parallelepiped* of the lattice $\mathit{\Lambda}$ [Barvinok (2002)]. Once again for us to visualize this we will consider a lattice in $\mathbb{R}^2$ with a basis $(0, 1)$ and $(1, 0)$. The parallelepiped will take the following form:



The intervals from $(0, 0)$ to $(1, 0)$ and $(0, 0)$ to $(0, 1)$ are both contained in the parallelepiped whereas the intervals from $(1, 0)$ to $(1, 1)$ and $(0, 1)$ to $(1, 1)$ are not. Also, the point $(0, 0)$ is within the parallelepiped whereas the points $(1, 0)$, $(0, 1)$ and $(1, 1)$ are not. Thus, we can translate the parallelepiped one unit in the direction of one of our basis vectors and still have an empty intersection. At the same time the union of all these translations of the basis vectors, $z_1(0, 1) + z_2(1, 0)$, where $z_1, z_2 \in \mathbb{Z}$, will be equal to $\mathbb{R}^2$. This gives a partition of $\mathbb{R}^2$.
An important concept in this section is the idea of what makes a good or bad basis for a lattice. A good basis is typically used as the private key for a system and the bad basis used for the public key. The difference between the two bases is the difficulty in solving



Integer lattice with basis $(0, 1), (1, 0)$ Integer lattice with basis $(1, 1), (4, 3)$

Figure 3.1: Example good and bad basis

the closest vector problem. One way to explain the difference is through the use of the following two diagrams [Silverman (2006)], where the blue point we want to find the close lattice point.

Definition. *Closest vector problem*: Given a basis B of a lattice $L = L(B) \subset \mathbb{R}^n$ and a vector $\mathbf{x} \in \mathbb{R}^n$ (i.e. not necessarily an element of L), find $\mathbf{u} \in L$ such that

$$||\mathbf{x} - \mathbf{u}|| = dist(\mathbf{x}, L). \tag{3.14}$$

Given a lattice with a fixed basis and a point in the space we are working in, perhaps there exists a parallelpiped which contains that point.

Definition. Given a lattice L generated by an n dimensional basis B, the *Hadamard ratio* [Hoffstein *et al.* (2008)] of the basis is defined as

$$H(B) = (\frac{|det(B)|}{||\mathbf{b}_1||_2 \cdot ||\mathbf{b}_2||_2 \cdots ||\mathbf{b}_n||_2})^{1/n}. \tag{3.15}$$

Here $det(B)$ represents the determinant of the matrix formed from the basis vectors of B. Also $||\mathbf{b}_i||_2$ represents the Euclidean norm of the basis vector $\mathbf{b}_i$. The range of values for this ratio is $0 < H(B) \leq 1$.
A *good basis* is one whose Hadamard ratio is close to 1. Consequently a *bad basis* is one who Hadamard ratio is close to 0.

In figure 3.1, the left diagram represents a good basis. Here we can see that the closest lattice point to our given point will be one of the corner points on the closure of the fundamental parallelepiped. This makes solving the closest vector problem possible in a reasonable amount of time. The right diagram represents a bad basis, where the closest point on the parallelepiped is not the closest point on the lattice. Clearly, in two dimensions we can spot the closest vector point, however, in higher dimensions, algorithms that solve this problem are exponential in complexity [Becker *et al.* (2013)] given a bad basis.

The following encryption, named GGH (after its inventors Goldreich, Goldwasser and Halevi), system is based on the closest vector problem [Goldreich *et al.* (1997)].

Setup

The *GGH encryption* system takes advantage of the closest vector problem where the private key is a good basis R of a lattice (the letter R chosen to represent the fact that this basis is more reduced than the public basis). The basis vectors $\mathbf{r}_1, \ldots, \mathbf{r}_n$ of this basis are represented as the matrix

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n,1} & r_{n,2} & \cdots & r_{n,n} \end{bmatrix}, \tag{3.16}$$

where $r_{i,j}$ is the j^{th} element of $\mathbf{r}_i$.
The public key is a bad basis B for the same lattice, alongside the dimension n and a security parameter $\sigma \in \mathbb{Z}$. σ should be chosen to be small enough that the ciphertext remains closer to the intended lattice point than any other lattice point. The basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ of this basis are represented as the matrix

$$B = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{bmatrix}, \tag{3.17}$$

where $b_{i,j}$ is the j^{th} element of $\mathbf{b}_i$.
It has been suggested that an appropriate method to create a bad basis is to use the Hermite Normal Form of the good basis [Micciancio (2001)], which is a upper triangular matrix with all entries greater than or equal to 0. The diagonal elements in particular are strictly greater than all other values in the same row. Micciancio recommends [Micciancio & Warinschi (2001)] as an efficient way to generate the Hermite Normal Form.

Encryption

Express a message as a vector $\mathbf{m} \in \mathbb{Z}^n$, then the ciphertext is computed as

$$\mathbf{c} = B\mathbf{m} + \mathbf{e}, \tag{3.18}$$

where $\mathbf{e} \in \{-\sigma, \sigma\}^n$ (i.e. the set of 2 elements σ and -σ) is generated randomly each time a ciphertext is sent.

Decryption

The ciphertext $\mathbf{c}$ is decrypted with the following formula

$$\mathbf{m} = B^{-1}R\lfloor R^{-1}\mathbf{c}\rceil, \tag{3.19}$$

where $\lfloor . \rceil$ rounds each element of the vector inside to the nearest integer. To see why this method works, lets consider how the private key is utilized and what is left before the inverse of the public key is used.

$$\begin{aligned} R\lfloor R^{-1}\mathbf{c}\rceil &= R\lfloor R^{-1}(B\mathbf{m} + \mathbf{e})\rceil \\ &= R(R^{-1}B\mathbf{m} + \lfloor R^{-1}\mathbf{e}\rceil) \\ &= B\mathbf{m} + R\lfloor R^{-1}\mathbf{e}\rceil. \end{aligned} \tag{3.20}$$

From this final form, it is clear that if $\lfloor R^{-1}\mathbf{e}\rceil = \mathbf{0}$, a correct decryption will occur. This is why the choice of σ is so crucial; increasing its size increases the chances of $\lfloor R^{-1}\mathbf{e}\rceil \neq \mathbf{0}$. For the purposes of this chapter we will always assume we have an appropriate choice of σ, but for a more in depth look as to what that entails, refer to [Goldreich *et al.* (1997),van de Pol (2011)].
Here the main concern is choosing a candidate value of $\mathbf{e}$ that is both not so"small" that it is possible for someone to crack the decryption but at the same time not so "large" that when we try to decrypt the message we end up with the wrong plain text.

3.2 Performing addition in an SHE while maintaining error size

In this section, we will begin by looking at how addition can be performed on a quantum computer, which preserves superpositions, using an example circuit from [Takahashi *et al.* (2009)]. The remainder of the section covers a new method we introduce to adapt the GGH encryption system, allowing us to theoretically perform an infinite number of additions without risk of incorrect decryption.

3.2.1 Addition on a quantum circuit

Before new content, we need to understand the following work done by Takahashi et al. We wish to be able to add quantum ciphertexts and therefore need a method to perform addition on qubits. To perform the addition we will use the circuit discussed in [Takahashi *et al.* (2009)] described just below. The total number of qubits input to this circuit is $2n+1$.

n is the maximum number of qubits of the two values we wish to add (if the other value has fewer qubits then we pad with zeroes). The qubits $A_0, \dots, A_{n-1}$ are binary representation the first value $B_0, \dots, B_{n-1}$ are the binary representation of the second value. The final qubit A_n is there to act as a carry bit. While this circuit will output the same number of qubits, only $n+1$ of them will represent the desired output.

1. For $i = 1, \cdots, n-1$:
   Apply a CNOT gate to each pair of memory locations B_i and A_i where A_i is the control qubit.

2. For $i = n-1, \cdots, 1$:
   Apply a CNOT gate to each pair of memory locations A_i and A_{i+1} where A_i is the control qubit.

3. For $i = 0, \cdots, n-1$:
   Apply a Toffoli gate to each tuple of memory locations B_i, A_i and A_{i+1}, where B_i and A_i are the control qubits.

4. For $i = n-1, \cdots, 1$:
   Apply a CNOT gate to each pair of memory locations B_i and A_i where A_i is the control qubit, followed by applying a Toffoli gate to each tuple of memory locations B_{i-1}, A_{i-1} and A_i, where B_{i-1} and A_{i-1} are the control qubits.

5. For $i = 1, \cdots, n-2$:
   Apply a CNOT gate to each pair of memory locations A_i and A_{i+1} where A_i is the control qubit.

6. For $i = 0, \cdots, n-1$:
   Apply a CNOT gate to each pair of memory locations B_i and A_i where A_i is the control qubit.

We will use $\diamond$ to denote this qubit addition operation.

The two gates of interest in this algorithm are the CNOT gate and the Toffoli gate. The CNOT gate maps

$$|10\rangle \mapsto |11\rangle \tag{3.21}$$

and

$$|11\rangle \mapsto |10\rangle. \tag{3.22}$$

The other basis vectors are fixed.

The Toffoli gate maps

$$|110\rangle \mapsto |111\rangle \tag{3.23}$$

and

$$|111\rangle \mapsto |110\rangle. \tag{3.24}$$

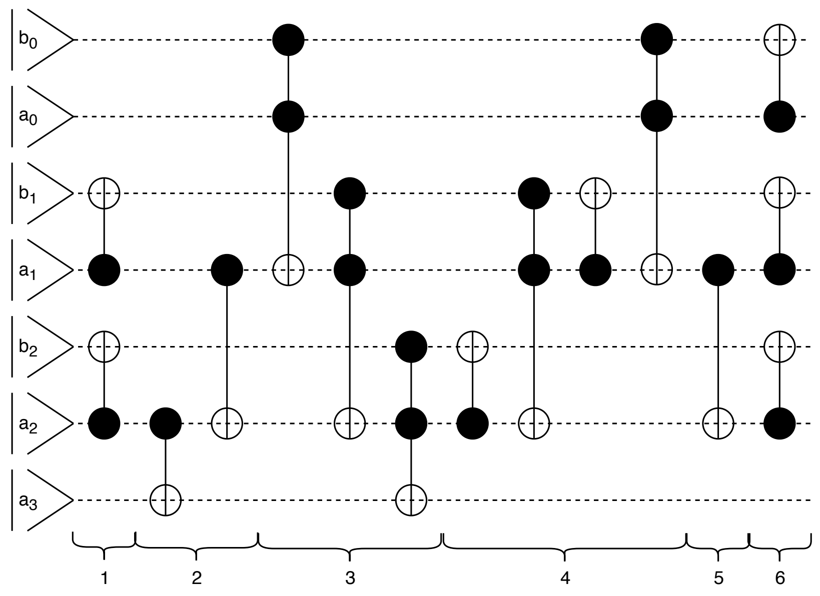


Figure 3.2: Quantum addition for two 3 qubit values

All other basis vectors are fixed.

3.2.2 The problem with Somewhat homomorphic encryption

We look at what is known as a somewhat homomorphic encryption system. This class of cryptosystem allows encrypted data to be processed but each operation introduces some error. Only a finite number of operations can be performed on the encrypted data correctly as the error will grow too large to deal with.
Figure 3.3 demonstrates this problem within the GGH cryptosystem. Due to the distributive nature of matrix multiplication, only the additional error part of the encryption formula prevents an exact homomorphism. The figure shows the possible accumulative error of a ciphertext formed from each number of additions. The red dashed line represents a hypothetical bound for guaranteed correct decryption, while the black line represents no error. A good ciphertext should remain between these bounds, however, it is clear to see that as computation goes on, the probability of an acceptable amount of error decreases.

We are thinking of GGH as a post-quantum cryptopsystem, however, it in itself is not a quantum cryptosystem. The current version has does not depend on quantum properties. In section 3.2.3 and onward we will adapt GGH to become a quantum cryptosysyem by creating a superposition of all possible encryptions of a plaintext. From there we will implement an error-correcting method to extend the number of additions that can be performed. To keep the structure of the ciphertexts as simple as possible, we will choose σ to be of the form 2^k, $k \in \mathbb{N}$.

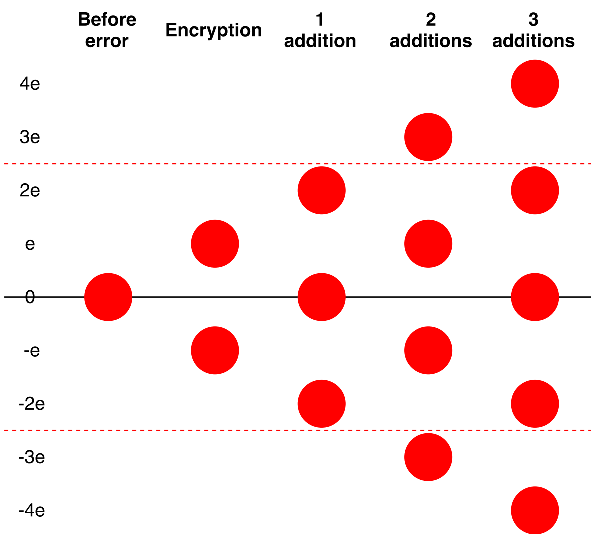


Figure 3.3: Evolution of error

3.2.3 Setting up a quantum variant of GGH

The remainder of the content in this chapter is new work contributed by the author. Our aim is to have a superposition of all potential ciphertexts. We start with 2 plaintext messages $\mathbf{m}_1 = [m_{1,1}, m_{1,2}, \ldots, m_{1,n}]^t$ and $\mathbf{m}_2 = [m_{2,1}, m_{2,2}, \ldots, m_{2,n}]^t$. To begin with consider the classical values after having only calculated the matrix multiplied on the left, i.e. $B\mathbf{m}$, where B is the bad basis defined in equation (3.15). We now need to add the error term on. We will do this in two stages. Instead of immediately adding $\mathbf{e}$ with entries in $\{-2^k, 2^k\}$, we will express the error as

$$\mathbf{e} = -2^k[1, \ldots, 1]^t + \mathbf{e}_1, \tag{3.25}$$

where $\mathbf{e}_1$ has entries in $\{0, 2^{k+1}\}$.

We subtract the vector of 2^k's to begin with for ease of calculation. Suppose we have an integer c, the binary representation of this number is $b_0 b_1 \cdots b_k$, where $c = \sum_{i=0}^{k} b_i 2^i$ and $b_i \in \{0, 1\}$. The important observation here is that the difference in the binary representations 0 and 2^k only differ in the $k + 1^{st}$ bit. For example, $8 = 2^3$ and its binary representation contains a 1 in the 4th position. All other bits are 0. This is why we subtracted 2^k after performing the basis multiplication. Now, in the classical world, we can choose a vector of length n, where each element of the vector comes from $\{0, 2^{k+1}\}$, instead of $\{-2^k, 2^k\}$. Testing of this work was done with Microsoft's $liqui|>$ software [Wecker & Svore (2014)], this step was taken to make the quantum operation a little easier.

In the first stage of adding the error we subtract 2^k from each value thus giving a valid encryption. This gives us 2 ciphertexts of the form

$$\mathbf{c}_1^- = B\mathbf{m}_1 - \mathbf{2^k} = \begin{bmatrix} B_{1,1}m_{2,1} + B_{1,2}m_{2,2} + \cdots + B_{1,n}m_{2,n} - 2^k \\ B_{2,1}m_{2,1} + B_{2,2}m_{2,2} + \cdots + B_{2,n}m_{2,n} - 2^k \\ \vdots \\ B_{n,1}m_{2,1} + B_{n,2}m_{2,2} + \cdots + B_{n,n}m_{2,n} - 2^k \end{bmatrix} \tag{3.26}$$

and

$$\mathbf{c}_2^- = B\mathbf{m}_2 - \mathbf{2^k} = \begin{bmatrix} B_{1,1}m_{2,1} + B_{1,2}m_{2,2} + \cdots + B_{1,n}m_{2,n} - 2^k \\ B_{2,1}m_{2,1} + B_{2,2}m_{2,2} + \cdots + B_{2,n}m_{2,n} - 2^k \\ \vdots \\ B_{n,1}m_{2,1} + B_{n,2}m_{2,2} + \cdots + B_{n,n}m_{2,n} - 2^k \end{bmatrix} \tag{3.27}$$

where $\mathbf{2^k}$ represents a vector of all 2^k's.
Our goal now is to have a ciphertext that is in a superposition of all possible variations of the encryption. Now consider the binary representation of the numbers in our ciphertext vector at this point, say the h-th element of $\mathbf{c}_1^-$ and $\mathbf{c}_2^-$ is $b_{h,1,0}b_{h,1,1}\dots b_{h,1,k},\dots$ and $b_{h,2,0}b_{h,2,1}\dots b_{h,2,k},\dots$ respectively, where $b_{h,i,j} \in \{0,1\}$, $h \in [1,n]$. If we treat those bits as qubits then performing the addition

$$\begin{aligned} &\begin{bmatrix} |b_{1,i,0}\rangle|b_{1,i,1}\rangle\dots|b_{1,i,k}\rangle\dots \\ |b_{2,i,0}\rangle|b_{2,i,1}\rangle\dots|b_{2,i,k}\rangle\dots \\ \vdots \\ |b_{n,i,0}\rangle|b_{n,i,1}\rangle\dots|b_{n,i,k}\rangle\dots \end{bmatrix} + \begin{bmatrix} |0\rangle|0\rangle\dots\frac{|0\rangle+|1\rangle}{\sqrt{2}} \\ |0\rangle|0\rangle\dots\frac{|0\rangle+|1\rangle}{\sqrt{2}} \\ \vdots \\ |0\rangle|0\rangle\dots\frac{|0\rangle+|1\rangle}{\sqrt{2}} \end{bmatrix} \\ &= \begin{bmatrix} |b_{1,i,0}b_{1,i,1}\dots b_{1,i,k}\dots\rangle \diamond |00\dots\rangle\frac{|0\rangle+|1\rangle}{\sqrt{2}}\dots \\ |b_{2,i,0}b_{2,i,1}\dots b_{2,i,k}\dots\rangle \diamond |00\dots\rangle\frac{|0\rangle+|1\rangle}{\sqrt{2}}\dots \\ \vdots \\ |b_{n,i,0}b_{n,i,1}\dots b_{n,i,k}\dots\rangle \diamond |00\dots\rangle\frac{|0\rangle+|1\rangle}{\sqrt{2}}\dots \end{bmatrix} \end{aligned} \tag{3.28}$$

results in a superposition of all possible ciphertexts in the classical version. Here the superposition is appearing at the same qubit position as the $b_{h,i,k}$ bit.
Due to the component wise property of vector addition, we will treat our vectors as if they are length 1. We can therefore tidy up the notation of our bits from $b_{h,i,j}$ to $b_{i,j}$. Furthermore, because we are now dealing with vectors of length 1, we will drop the use of the vector entirely whilst looking at the addition method.
Instead of dealing with cumbersome binary notation we may use the following notation for a ciphertext $\mathbf{c}$

$$\mathbf{c} = \frac{|B\mathbf{m} \diamond (-\sigma)\rangle}{\sqrt{2}} + \frac{|B\mathbf{m} \diamond \sigma\rangle}{\sqrt{2}} \tag{3.29}$$

3.2.4 Preventing an attacker from reversing the quantum gates used for encryption

We stated in the previous section that any attempt to measure of a quantum ciphertext would result in it collapsing down to the classical version. This approach therefore would suggest that the quantum version is no easier to break than the classical version. Quantum gates, however, are reversible and therefore provides reason to believe that if an encryption method is public, an attacker could simply perform the gates in reverse order to undo the encryption method. Let us consider an example using the quantum circuit from 3.2.

Example. Let $a_0a_1a_2$ be the binary representation of a 3 bit plaintext, ready to add the superposition error term say

$$b_0b_1b_2 = |0\rangle\frac{|0\rangle + |1\rangle}{\sqrt{2}}|0\rangle. \tag{3.30}$$

The 7 qubits passed into the circuit therefore are of the form

$$\frac{|0a_20a_10a_00\rangle}{\sqrt{2}} + \frac{|0a_20a_11a_00\rangle}{\sqrt{2}}. \tag{3.31}$$

This is the input to step 1 of the process.
Step 1 transformation outputs:

$$\frac{|0a_2a_2a_1a_1a_00\rangle}{\sqrt{2}} + \frac{|0a_2a_2a_1(1-a_1)a_00\rangle}{\sqrt{2}}. \tag{3.32}$$

This is the input to step 2 of the process.
Step 2 transformation outputs:

$$\frac{|a_2(a_2 \oplus a_1)a_2a_1a_1a_00\rangle}{\sqrt{2}} + \frac{|a_2(a_2 \oplus a_1)a_2a_1(1-a_1)a_00\rangle}{\sqrt{2}}. \tag{3.33}$$

This is the input to step 3 of the process.
Step 3 transformation outputs:

$$\frac{|0a_2a_2a_1a_1a_00\rangle}{\sqrt{2}} + \frac{|(a_2 \wedge a_1)(a_2 \oplus a_1)a_2a_1(1-a_1)a_00\rangle}{\sqrt{2}}. \tag{3.34}$$

This is the input to step 4 of the process.
Note the effect of the Toffoli gate on $|a_2a_1b_1\rangle$, with a_2 being the target qubit. We have the transformation $\frac{(a_2\oplus a_1)a_1a_1}{\sqrt{2}} + \frac{(a_2\oplus a_1)a_1(1-a_1)}{\sqrt{2}} \to \frac{a_2a_1a_1}{\sqrt{2}} + \frac{(a_2\oplus a_1)a_1(1-a_1)}{\sqrt{2}}$. The first term in the superposition comes from the fact $a_2 \oplus a_1 \oplus a_1 = a_2$. The target in the second term of the superposition will never change as a_1 and $1 - a_1$ cannot both equal 0. Also when a_3 is the target, the control values a_2 and $a_2 \oplus a_1$ can't be be both equal to 1 if $a_1 = 1$.

Step 4 transformation outputs:

$$\frac{|0(a_2 \oplus a_1)0a_10a_00\rangle}{\sqrt{2}} + \frac{|(a_2 \wedge a_1)(a_2 \oplus a_1)a_1a_11a_00\rangle}{\sqrt{2}}. \tag{3.35}$$

This is the input to step 5 of the process.
Step 5 transformation outputs:

$$\frac{|0a_20a_10a_00\rangle}{\sqrt{2}} \frac{|(a_2 \wedge a_1)a_2a_1a_11a_00\rangle}{\sqrt{2}}. \tag{3.36}$$

This is the input to step 6 of the process.
Step 6 transformation outputs:

$$\frac{|0a_2a_2a_1a_1a_0a_0\rangle}{\sqrt{2}} + \frac{|(a_2 \wedge a_1)a_2(a_2 \oplus a_1)a_1(1-a_1)a_0a_0\rangle}{\sqrt{2}}. \tag{3.37}$$

The ciphertext is therefore

$$a_3b_2b_1b_0 = \frac{|0a_2a_1a_0\rangle}{\sqrt{2}} + \frac{|(a_2 \wedge a_1)(a_2 \oplus a_1)(1-a_1)a_0\rangle}{\sqrt{2}}. \tag{3.38}$$

Although the qubits a_2, a_1 and a_0 are not needed for further computation, the encryption circuit can only be reversed if the inputs are the values originally output. Therefore, although we already have our ciphertext, there is one final step before moving data into a public cloud. Recall that the X gate, defined in 2.5.2, swaps the probability of measuring a 1 or 0 in a single qubit. Performing a random number of X gates on the a_2, a_1 and a_0 qubits will produce a ciphertext that will output an incorrect plaintext if an eavesdropper tries to reverse the encryption gates. Figure 3.4 shows the offline process where r_0, r_1 and r_2 are positive integers chosen at random and kept secret. See appendix for a simple example of a circuit encrypting 2 values then adding them together.

3.2.5 Performing addition on the quantum ciphertexts and reducing the error increase

Let $c_1 = \frac{|B\mathbf{m}_1 \diamond (-\sigma)\rangle}{\sqrt{2}} + \frac{|B\mathbf{m}_1 \diamond \sigma\rangle}{\sqrt{2}}$ and $c_2 = \frac{|B\mathbf{m}_2 \diamond (-\sigma)\rangle}{\sqrt{2}} + \frac{|B\mathbf{m}_2 \diamond \sigma\rangle}{\sqrt{2}}$. In the classical version of the algorithm, the error will be either $\sigma \in \mathbb{Z}$ or $-\sigma$. As we add other ciphertexts the error will increase by σ or decrease by σ with equal probability. Figure 3.3 shows the potential error values for each number of ciphertext additions. In particular after performing one addition the error will have either canceled out, or doubled in size i.e. will have the form

$$\frac{|B\mathbf{m} \diamond (-2\sigma)\rangle}{2} + \frac{|B\mathbf{m}\rangle}{\sqrt{2}} + \frac{|B\mathbf{m} \diamond 2\sigma\rangle}{2}, \tag{3.39}$$

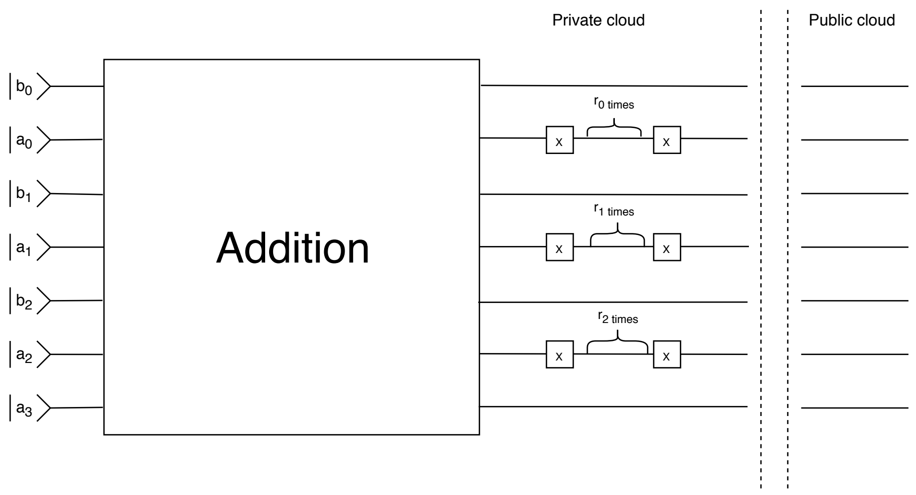


Figure 3.4: Full offline process

where $\mathbf{m} = \mathbf{m}_1 + \mathbf{m}_2$.

If the goal is to correct the error back to a standard ciphertext, the easiest place to start would be adding or subtracting σ to each superposition of ciphertexts. This new superposition will be of the form

$$\frac{|B\mathbf{m} \diamond (-\sigma)\rangle}{2} + \frac{|B\mathbf{m} \diamond \sigma\rangle}{\sqrt{2}} + \frac{|B\mathbf{m} \diamond 3\sigma\rangle}{2} \text{ or } \frac{|B\mathbf{m} \diamond (-3\sigma)\rangle}{2} + \frac{|B\mathbf{m} \diamond (-\sigma)\rangle}{\sqrt{2}} + \frac{|B\mathbf{m} \diamond \sigma\rangle}{2}. \tag{3.40}$$

The problem that remains is to find the appropriate gate that forms the map

$$\frac{|B\mathbf{m} \diamond (-3\sigma)\rangle}{2} + \frac{|B\mathbf{m} \diamond \sigma\rangle}{2} \mapsto \frac{|B\mathbf{m} \diamond \sigma\rangle}{\sqrt{2}} \tag{3.41}$$

or

$$\frac{|B\mathbf{m} \diamond 3\sigma\rangle}{2} + \frac{|B\mathbf{m} \diamond (-\sigma)\rangle}{2} \mapsto \frac{|B\mathbf{m} \diamond (-\sigma)\rangle}{\sqrt{2}}. \tag{3.42}$$

If the error is of the form 2^k, then the qubits we will look at to perform this transformation in binary representation are positions $k+1$ and $k+2$. Because of the way these superpositions are created, the qubits in the 2 positions of interest can be one of 4 forms. To assist in working out which form we are looking at and correctly perform error correction, upon encryption we introduce a new variable we call $CTRL_3$. This variable is a vector of length n where

$$B\mathbf{m} = \begin{bmatrix} b_{1,0}b_{1,1}b_{1,2}\cdots \\ b_{2,0}b_{2,1}b_{2,2}\cdots \\ \vdots \\ b_{n,0}b_{n,1}b_{n,2}\cdots \end{bmatrix} \Rightarrow CTRL_3 = \begin{bmatrix} b_{1,k}b_{1,(k+1)}b_{1,(k+2)} \\ b_{2,k}b_{2,(k+1)}b_{2,(k+2)} \\ \vdots \\ b_{n,k}b_{n,(k+1)}b_{n,(k+2)} \end{bmatrix}, \tag{3.43}$$

i.e. the i^{th} element of the $CTRL_3$ vector is a substring of the binary representation of the i^{th} element of $B\mathbf{m}$. When we add two ciphertexts together, the $CTRL_3$ vectors are added together and each element is reduced modulo 8. Denote row i of $CTRL_3$ by $CTRL_3(i)$. Note that each row is independent of all of the other rows. Using this value, we can establish some information about the form of the superposition, namely the properties in table 3.1.

$CTRL_3(i)$ value	1st superposition term	2nd superposition term	3rd superposition term
2 or 3	$\frac{\lvert\cdots 00\cdots\rangle}{2}$	$\frac{\lvert\cdots 10\cdots\rangle}{\sqrt{2}}$	$\frac{\lvert\cdots 01\cdots\rangle}{2}$
4 or 5	$\frac{\lvert\cdots 10\cdots\rangle}{2}$	$\frac{\lvert\cdots 01\cdots\rangle}{\sqrt{2}}$	$\frac{\lvert\cdots 11\cdots\rangle}{2}$
6 or 7	$\frac{\lvert\cdots 01\cdots\rangle}{2}$	$\frac{\lvert\cdots 11\cdots\rangle}{\sqrt{2}}$	$\frac{\lvert\cdots 00\cdots\rangle}{2}$
0 or 1	$\frac{\lvert\cdots 11\cdots\rangle}{2}$	$\frac{\lvert\cdots 00\cdots\rangle}{\sqrt{2}}$	$\frac{\lvert\cdots 10\cdots\rangle}{2}$

Table 3.1: Superposition form for each control value

Here we have learned about the $k + 1$ and $k + 2$ positions. These qubits decide which case we consider.
Table 3.2 gives us a few examples of ciphertext superpositions with an error value of 1. We can see that upon measuring the $k+1^{th}$ (in this case 2^{nd}) qubit we are left with either a value in the final column or a superposition of the states in the first 2 columns. If we know that we just have a value in the final column then we have an appropriate error and all is left is to create the superposition of errors again.
We will consider here what the process is, for each case, to correct the error given that measuring the $k+1$ qubit leaves us with values from the first two columns. The first thing to note is that in every superposition, all the qubits before the 2 of interest will be the same amongst all 3 terms so there is no need to correct anything there.

Case 1 - 00,10,01

The simplest case to deal with. We perform the following operation

$$X|0\rangle H(\frac{|0\rangle + |1\rangle}{\sqrt{2}}) = |10\rangle. \tag{3.44}$$

All following qubits in the terms are always equal in this case.

Case 2 - 10,01,11

The process here is similar to case 1 however for the Hadamard gate to collapse the terms into 1 instead of 0, a negation is needed first. The process is the following

$$X|1\rangle HZ(\frac{|0\rangle + |1\rangle}{\sqrt{2}}) = |01\rangle. \tag{3.45}$$

Once again all following qubits in the representation are equal.

Case 3 and 4 - 01,11,00 and 11,00,10

These 2 cases are more tricky to deal with. The issue here is the presence of the 00 at the end and in the middle of the lists. This suggests that in this list, the next multiple of 2^k has been reached, completely changing the binary structure of the following values, not just the $k+1$ and $k+2$ Positions. For example, suppose $\sigma = 1$, we may have a superposition of the form

$$\frac{011011}{2} + \frac{000111}{\sqrt{2}} + \frac{010111}{2}. \tag{3.46}$$

Here the qubits in the $k+3$ position aren't all the same as would they would be in case 1 and 2. Correction here, therefore, becomes an issue, as working out how many of the qubits have changed would most likely end up in performing measurements that would destroy the structure.

An initial solution is to run the addition of the ciphertexts again. Upon the next run of the addition, we can have 1 of 3 outcomes, when we measure the state completely. Firstly, with a 50% probability, we will observe $B\mathbf{m}_1 - \sigma$ (Assuming WLOG that is our middle value). If this is the case, we can just accept that value and ignore the previously obtained value. Should we measure a different value despite being in the same superposition as the last run, we can simply take the average of these 2 values giving us $B\mathbf{m}_1 - \sigma$. The final possibility is we observe the same value in the superposition again, thus learning nothing new and we repeat the process once again. A flowchart of this process is given in figure 3.5.

Lemma 3.1. *Suppose it was possible to clone qubits. Asymptotically, the addition algorithm with error correction can be performed successfully.*

Proof. We will first assume that we have a uniform probability of observing any value for the error case. We have already shown that if the error case is 1 or 2, then correction is possible. So far we have at least a 50% success rate. Suppose WLOG our ciphertext is of the form $\frac{|B\mathbf{m}_1-3\sigma\rangle}{2} + \frac{|B\mathbf{m}_1-\sigma\rangle}{\sqrt{2}} + \frac{|B\mathbf{m}_1+\sigma\rangle}{2}$. When the error case takes the values 3 and 4, if we observe $B\mathbf{m} - \sigma$ when taking a measurement, then once again we're in a successful position. Since this outcome has a 50% chance of happening upon measurement, our overall success probability is 75%.

The other potential outcome results in an iteration of the algorithm. As previously stated, the only "bad" outcome from this iteration is measuring the exact same value, which results in us iterating once again. We assume WLOG, that in the first round of our algorithm we observe $B\mathbf{m}_1 - 3\sigma$. In any of our following iterations, if we observe $B\mathbf{m}_1 - \sigma$ or $B\mathbf{m}_1 + \sigma$ then we succeed. If we consider observing $B\mathbf{m}_1 - 3\sigma$ a failure and observing $B\mathbf{m}_1 - \sigma$ or $B\mathbf{m}_1 + 3\sigma$ a success, then our outcome after t iterations can be modeled with

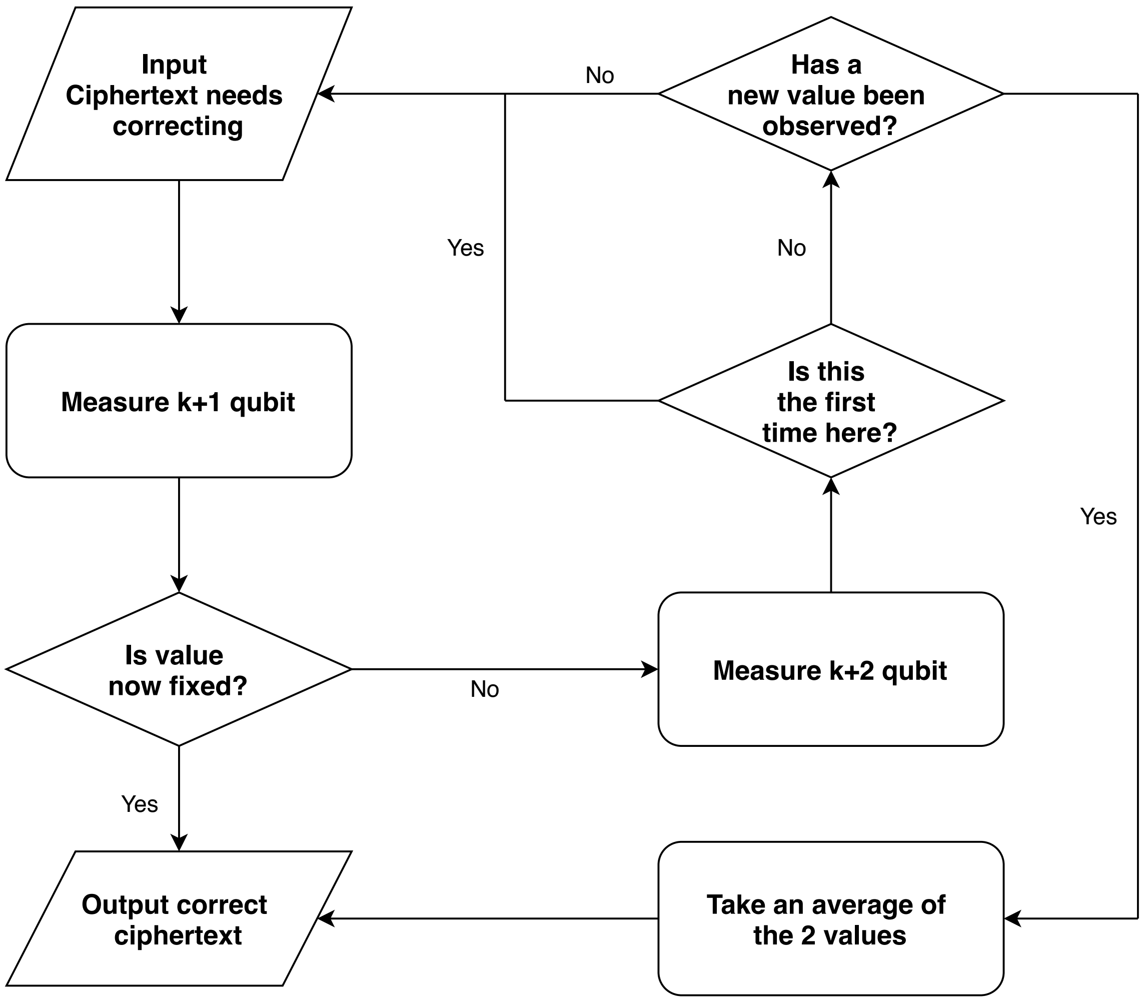

Input
Ciphertext needs correcting
Measure k+1 qubit
Is value now fixed?
Yes
Output correct ciphertext
No
Measure k+2 qubit
Is this the first time here?
Yes
No
Has a new value been observed?
No
Yes
Take an average of the 2 values

Figure 3.5: Iterative method to correct the error

a $Binomial(t-1, 0.75)$ distribution (our first round isn't counted as we need to establish that $B\mathbf{m}_1 - 3\sigma$ is the failure). Our overall success rate is

$$\begin{aligned} 0.75 + 0.25 * Pr(\text{at least 1 success}) &= 0.75 + 0.25 * (1 - Pr(\text{all failures})) \\ &= 0.75 + 0.25 * (1 - 0.25^{t-1}), \end{aligned} \tag{3.47}$$

which clearly approaches 1 as we let our number of iterations tend to infinity. □

Corollary 3.1. *Consider any given row of a vector ciphertext from the GGH encryption method. In order to perform addition using the iterative method, $\frac{4}{3}$ is the expected number of iterations needed by the addition algorithm.*

Proof. As the lemma has shown the asymptotic nature of the method, the expected number of iterations required for each addition is

$$1 \cdot \frac{3}{4} + 2 \cdot (\frac{1}{4} \cdot \frac{3}{4}) + 3(\frac{1}{4^2} \cdot \frac{3}{4}) + 4(\frac{1}{4^3} \cdot \frac{3}{4}) + \cdots = \sum_{n=1}^{\infty} n \cdot \frac{1}{4^{n-1}} \cdot \frac{3}{4}. \tag{3.48}$$

The expectation matching that of a geometric distribution has a resulting value of $\frac{4}{3}$. □

Notice in the lemma the phrase 'suppose it was possible to clone qubits'. Work by Wootters and Zurek proved that it is impossible to create an exact copy of a qubit, a technique extremely common in classical computing.

Theorem 3.1. *No cloning Theorem [Wootters & Zurek (1982)]. There is no unitary operation*

$$|\psi\rangle|0\rangle \mapsto |\psi\rangle|\psi\rangle, \tag{3.49}$$

for any state $|\psi\rangle$.

Although it isn't possible to clone qubits, the same outcome can be achieved by initially creating multiple copies of the same encryption. This iterative method may help to increase the probability of success, however, multiple copies of the ciphertext in a superposition state are cause for concern. This is providing more information for a potential attacker to manipulate. On top of that more computation is required.

$B\mathbf{m}_1 - 3\sigma$	$B\mathbf{m}_1 + \sigma$	$B\mathbf{m}_1 - \sigma$	Error case
7	11	9	4
$\|1110\rangle$	$\|1101\rangle$	$\|1001\rangle$	
9	13	11	1
$\|1001\rangle$	$\|1011\rangle$	$\|1101\rangle$	
11	15	13	2
$\|1101\rangle$	$\|1111\rangle$	$\|1011\rangle$	
13	17	15	3
$\|10110\rangle$	$\|10001\rangle$	$\|11110\rangle$	
15	19	17	4
$\|11110\rangle$	$\|11001\rangle$	$\|10001\rangle$	
17	21	19	1
$\|10001\rangle$	$\|10101\rangle$	$\|11001\rangle$	

Table 3.2: Example list of superpositions

We have discussed in detail how the quantum encryption method and addition error correction is performed. Below we outline the key steps of the process to go through encryption, addition and correction before refining the method.

Encryption of 2 plaintexts

1. Input two plaintexts, $\mathbf{m}_1, \mathbf{m}_2$, that we wish to encrypt and perform addition on.

2. Multiply our two vectors by our public basis, B.
3. For both vectors, create an associated vector $CTRL_3$. The i^{th} row of $CTRL_3$ is a 3 bit value equal to the bits at positions k, $k+1$ and $k+2$ in the original vector.
4. Add the error superpositions $\frac{|-\sigma\rangle+|\sigma\rangle}{\sqrt{2}} = \frac{|-2^k\rangle+|2^k\rangle}{\sqrt{2}}$ to each value.

Online addition

1. Perform the addition algorithm on the two updated ciphertexts. Subtract σ from the result.
2. Measure the qubit at position $k+1$ in the newly obtained value.
3. Add together the $CTRL_3$ variables and subtract 1, modulo 8. Establish if we now have a fixed value, if so go to step 7 if not carry on.
4. If $CTRL_3$ has a value of 6,7,0 or 1 repeat process until 2 different ciphertexts are observed. Take the average of these. Move to step 7.
5. If $CTRL_3$ has a value of 4 or 5 perform a Z gate on the $k+2^{th}$ qubit.
6. Perform the Hadamard gate on the $k+2^{th}$ qubit and an X gate on the $k+1^{th}$ qubit.
7. Add $\frac{|0\rangle+|2^{k+1}\rangle}{\sqrt{2}}$ to the value to create a new superposition.
8. Output new ciphertext.

3.3 Refining the method of addition on the ciphertexts

The addition algorithm described makes a promising start in dealing with the finite number of operations typically allowed with a somewhat homomorphic encryption system. That being said, this quantum variant brings new issues. This section will look to address those problems and how they may be practically dealt with.

3.3.1 Data at rest

The $CTRL_3$ variable is used to ensure the error can be corrected whilst performing additions. Unfortunately, that information also provides a potential attacker with key information. This opens up the ability to perform a chosen-ciphertext attack.

Lemma 3.2. *Knowledge of the $CTRL_3$ variable encrypted with GGH allows for decryption without knowledge of the secret key.*

Proof. When encrypted the superposition ciphertext is of the form

$$\frac{|B\mathbf{m} \diamond (-2^k)\rangle + |B\mathbf{m} \diamond 2^k\rangle}{\sqrt{2}}. \tag{3.50}$$

Suppose WLOG that if an attacker took a measurement they observe $|B\mathbf{m} \diamond 2^k\rangle$. Much like in the classical sense, we know that the superposition this value came from is either the one above or

$$\frac{|B\mathbf{m} \diamond 2^k\rangle + |B\mathbf{m} \diamond 3 \cdot 2^k\rangle}{\sqrt{2}}. \tag{3.51}$$

The $CTRL_3$ is therefore derived from either $B\mathbf{m}$ or $B\mathbf{m} \diamond 2 \cdot 2^k$.
Recall that the mapping from a row of $B\mathbf{m}$ to a row of $CTRL_3$ takes the binary representation $B\mathbf{m}$ is equal to $b_0 b_1 \dots b_k b_{k+1} b_{k+2} \dots$ to the binary representation $b_k b_{k+1} b_{k+2}$. Now consider the inverse of this map. The inverse of $b_k b_{k+1} b_{k+2}$ is the set

$$\begin{aligned}
&[00\dots 0b_k b_{k+1} b_{k+2}000\dots, 10\dots 0b_k b_{k+1} b_{k+2}000\dots, \dots, 11\dots 1b_k b_{k+1} b_{k+2}000\dots]\cup\\
&[00\dots 0b_k b_{k+1} b_{k+2}100\dots, 10\dots 0b_k b_{k+1} b_{k+2}100\dots, \dots, 11\dots 1b_k b_{k+1} b_{k+2}100\dots]\cup\\
&[00\dots 0b_k b_{k+1} b_{k+2}010\dots, 10\dots 0b_k b_{k+1} b_{k+2}010\dots, \dots, 11\dots 1b_k b_{k+1} b_{k+2}010\dots]\cup\\
&[00\dots 0b_k b_{k+1} b_{k+2}110\dots, 10\dots 0b_k b_{k+1} b_{k+2}110\dots, \dots, 11\dots 1b_k b_{k+1} b_{k+2}110\dots]\cup\\
&\vdots
\end{aligned} \tag{3.52}$$

Each of these sets have 2^k elements and the distance between corresponding elements of adjacent elements of each of these sets is 2^{k+3}. Since the difference between $B\mathbf{m}$ and $B\mathbf{m} \diamond 2 \cdot 2^k$ is 2^{k+1}, both values cannot be part of the same inverse sets. There is therefore a one to one mapping between a measured ciphertext and the plaintext given the $CTRL_3$ value. □

Example. Let the error term $\sigma = 1$ and let $n = 1$. Suppose a value is encrypted using the quantum method as a superposition of 11 and 13 i.e.

$$\mathbf{c} = \frac{|1101\rangle + |1011\rangle}{\sqrt{2}}. \tag{3.53}$$

Since the mid point value is 12, $CTRL_3 = 4$. Suppose WLOG, measuring the superposition has output the value 13. An attacker, therefore, knows that $B\mathbf{m}$ is equal to either 12 or 14. Now the inverse of our $CTRL_3$ value is

$$[-4] \cup [4] \cup [12] \cup [20] \cup \dots. \tag{3.54}$$

The only possible value for $B\mathbf{m}$ is 12 therefore an attacker can decrypt by multiplying this by the inverse of the encryption matrix.

When it comes to ensuring an attacker isn't able to use the $CTRL_3$ variable, it is important to consider when the ciphertext is in different states of usage. Firstly data at rest refers to when data is being stored and is typically needed in the future. This is where standard encryption methods are used. Data in use, as its name suggests, is when data is going through some kind of processing.
In classical computing, parity is used to check for unexpected changes in a binary string. The final bit in a string isn't used to represent data, it is selected to ensure the bits in the string sum to an even value (or odd depending on preference).
In order to detect an attempt at manipulating the $CTRL_3$ variable, a combination of parity and the quantum key exchange will be used. To begin with, each of the 3 qubits will be represented in either the standard basis or the diagonal basis. The choice for each qubit is independent of the others. If Alice is the one encrypting the data, then she doesn't tell anyone the choice of basis until an authorised user wants to analyse the data.
For each qubit in the $CTRL_3$ variable, if a potential eavesdropper chooses one of the two bases at random, they will choose the same as Alice with probability 0.5. Even if they do not, measuring with the wrong basis will still give them the correct value with probability 0.5. Therefore the probability of correctly identifying each qubit is 0.75. Given we are we have a vector of length n, with each element containing 3 qubits, an attacker would have a probability of 0.75^{3n} to correctly identify each $CTRL_3$ variable.
A fourth qubit may be added to incorporate parity. This will help in preventing an attackers goal that may be to disrupt the data, not necessarily decrypt. Due to the more fragile nature of qubits, a parity qubit will also provide a check to see if changes have occurred from the natural environment.

Example. Here the control variable is equal to 1 (001 in binary). At random we choose to represent the 3 bits using the standard basis, then diagonal basis, then the standard basis once again. The variable is therefore stored as

$$|0\rangle, \frac{|0\rangle + |1\rangle}{\sqrt{2}}, |1\rangle. \tag{3.55}$$

Assuming an even parity is needed, the parity bit will need to be a 1. It is therefore stored as either $|1\rangle$ or $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$.
We have already established the probability of an attacker finding the correct $CTRL_3$ vector but, now that we are interested in parity, we should check what the probability is of permitted user observing an even parity even if changes have been made. In order to observe the correct parity but wrong values, 2 or 4 of the measured qubits must swap between a 0 and 1. Since we have established that the probability of an attacker observing an incorrect value is 0.25, we must consider a distribution $X \sim Binomial(4, 0.25)$. Therefore, the probability of observing an incorrect value but believing it based on parity is equal to $Pr(X = 2) + Pr(X = 4) \approx 0.2148$.

As an alternative method for detection, an entanglement of qubits with the parity qubit could be used. Entangling the parity with any of the other qubits will result in the parity collapsing if someone tries to measure one of the other entangled qubits. Therefore, if at any point Alice suspects someone may have looked at the data, she can measure the parity bit and there will be a non zero probability that she will observe a worrying value. The data at rest situation may not arise as often in the quantum world due to the process of decoherence [Shor (1995)]. This is the idea that over time, qubits that are being stored will become entangled with their environment and therefore change. An attacker may not be able to trust the value of the $CTRL_3$ variable or corresponding ciphertext the longer it has been stored. Unfortunately, this also prevents error correction.

3.3.2 Data in use - A realistic look

Protecting the $CTRL_3$ variable from attackers while the data is being analysed is more difficult. The authorised user needs to be able to read the variable. Therefore, the choice of bases needs to be made public.
Without an obvious mathematical solution to protecting the data, the use of the technology analysing the data should be more carefully looked at. For example in [Watson (2012)], a cost versus security model for analysing data on a public cloud is discussed. In the paper, it is acknowledged that while the public cloud may have the best potential, a balance between private and public clouds is more realistic. Regardless of which type of cloud is being used, there is a set of simple instructions that make these processes work well. In this example, we shall use the following functions.

- *Split*: The data coming in to be processed is grouped into subsets. Because the data is still in a private environment, the data need not be encrypted at this point e.g.

$$f(x_1, x_2, \dots) = ((x_1, x_{i+1}, x_{2i+1} \dots), (x_2, x_{i+2}, x_{2i+2}), \dots), \tag{3.56}$$

  for some positive integer i.

- *Map*: Takes a list as input and applies a function to each element. For example here the function applied to each plaintext is the encryption function i.e.

$$f(x_1, x_2, \dots, x_n) = (E(x_1), E(x_2), \dots, E(x_n)). \tag{3.57}$$

- *Reduce*: Takes a list of data and aggregates it in some way to output a single value. In this case the encrypted values are added together and the result passed on to go into the public cloud e.g.

$$f(x_1, x_2, x_3) = x_1 + x_2 + x_3. \tag{3.58}$$

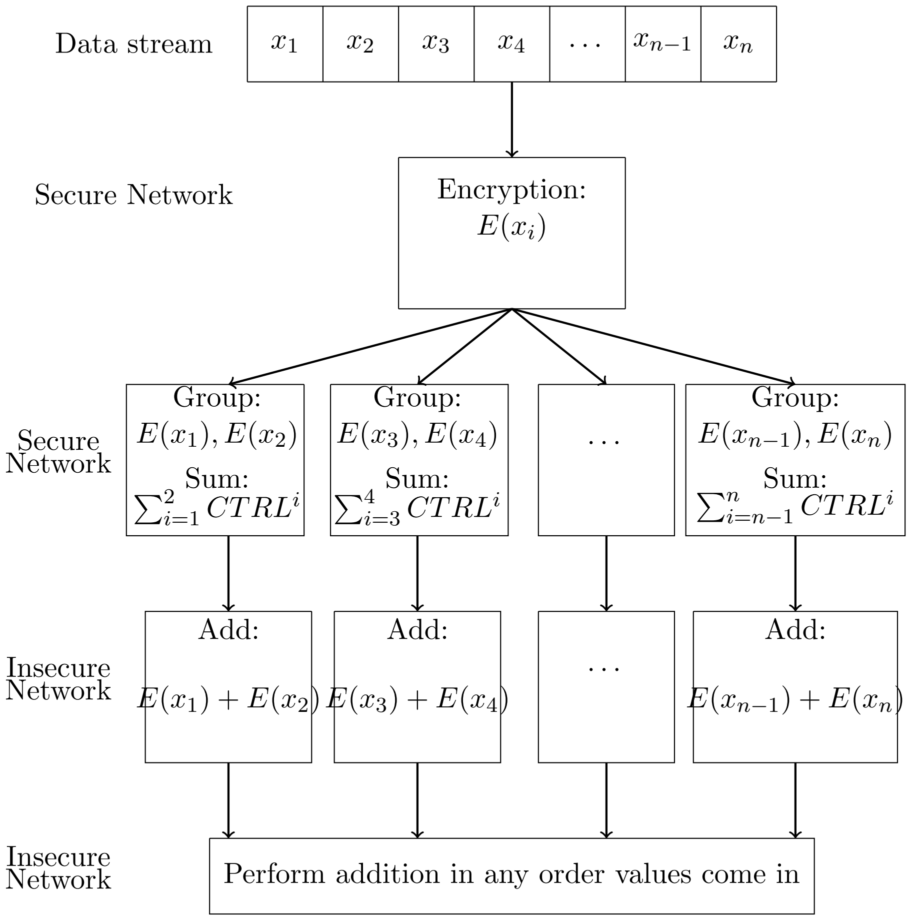

Data stream
x_1 x_2 x_3 x_4 $\dots$ x_{n-1} x_n
Secure Network
Encryption: $E(x_i)$
Secure Network
Group: $E(x_1), E(x_2)$ Sum: $\sum_{i=1}^{2} CTRL^i$
Group: $E(x_3), E(x_4)$ Sum: $\sum_{i=3}^{4} CTRL^i$
$\dots$
Group: $E(x_{n-1}), E(x_n)$ Sum: $\sum_{i=n-1}^{n} CTRL^i$
Insecure Network
Add: $E(x_1) + E(x_2)$
Add: $E(x_3) + E(x_4)$
$\dots$
Add: $E(x_{n-1}) + E(x_n)$
Insecure Network
Perform addition in any order values come in

Figure 3.6: Data analysed via both public and private clouds

The idea is to perform the first part of any kind of analysis in a private cloud, but then send the majority of the work off to the public cloud. This would mean that any values that were in the vulnerable public cloud are intermediate, as opposed to the original data. If more is done in the private cloud-first, less can be learned about the original data. Figure 3.6 gives an example requiring pairs of ciphertexts being added together offline.
While this may not come across as the most elegant solution, there are examples of technology available to achieve such goals. The company ADLink [1], for example, provides a product that has been specifically designed to help aggregate data collected by various sensors, reducing bandwidth costs of data transferred. The initial operations we desire would be handled well by such technology.
When establishing the original algorithm, there was concern about the practicality of the error correction method we established in section 3.2. Although the commutativity of the addition operation is helpful for efficient data analysis, restricting the order in which data is summed would ensure that only desired error cases appear. The vertices in the graph in figure 3.7 reference the 8 different $CTRL_3$ variables, $\{0, 1, 2, 3, 4, 5, 6, 7\}$. An edge exists between 2 vertices only if the addition of the 2 values (mod 8) in these vertices produces a “good” $CTRL_3$ variable i.e. $\{2, 3, 4, 5\}$.

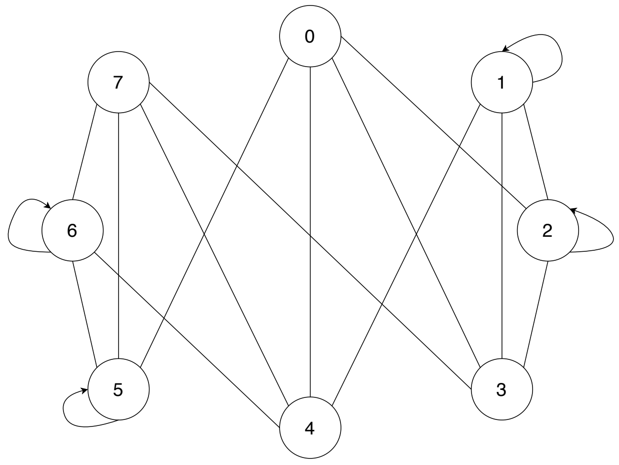


Figure 3.7: Addition graph

Let the vertices of the graph be labelled (x, t_x). Here x is the value of the $CTRL_3$ variable. The second value t_x is the current number of ciphertexts with the corresponding $CTRL_3$ value that is going to be part of our sum. If the goal is to add together all the ciphertexts, then given any ciphertext as a starting point, the following algorithm is an example of a reasonable set of instructions that should be followed to determine which order to add the ciphertexts in to avoid dealing with difficult cases.

[1]https://www.adlinktech.com/en/index.aspx

Algorithm 1 Addition graph ordering

```
1: procedure INPUT($G = (V, E)$), where $G$ is the addition graph for a collection ciphertexts to be added together.
2:     While $\sum_{i=0}^{7} t_i > 1$
3:         if $(0, t_0) \neq (0, 0)$ and $(x, t_x) \neq (x, 0)$ for at least one $x = 2, 3, 5$
4:             Add all ciphertexts in $(0, t_0)$ to a ciphertext in $(x, t_x)$.
5:         if $t_2 > 1$
6:             Add a ciphertext in $(2, t_2)$ to a ciphertext that belongs to an adjecent vertex.
7:         else if $t_3 > t_5$ and $\exists t_i, t_j \neq 0$ s.t. $i + j = 5 \pmod 8$
8:             Add together a ciphertext from $(i, t_i)$ to a ciphertext from $(j, t_j)$.
9:         else if $t_5 > t_3$ and $\exists t_i, t_j \neq 0$ s.t. $i + j = 3 \pmod 8$
10:            Add together a ciphertext from $(i, t_i)$ to a ciphertext from $(j, t_j)$.
11:        else if $\exists$ adjacent vertices $(i, t_i)$ and $(j, t_j)$ s.t. $t_i, t_j \neq 0$
12:            Add together a ciphertext from $(i, t_i)$ to a ciphertext from $(j, t_j)$.
13:        else
14:            Add together any 2 remaining ciphertexts.
15:    return $G$ with only one non empty vertex containing the sum of ciphertexts.
```

Of course, it is impossible to say how many ciphertexts will be associated with each of the vertices, the algorithm aims to prioritise adding certain ciphertexts first. We list the reasons for the priorities here:

- Add a ciphertext from $(0, t_0)$ to a ciphertext (x, t_x) will produce another ciphertext in (x, t_x). So in each iteration of the algorithm, we add all the ciphertexts in that vertex to ciphertexts in vertices where $x = 2, 3$ or 5 as we know these are some of our good cases.

- We observe that adding a ciphertext with $CTRL_3$ variable equal 2 to another with $CTRL_3$ variable equal to 3 gives a ciphertext with $CTRL_3$ value 5. Also adding two ciphertexts together with $CTRL_3$ value 5 will ouput a ciphertext with $CTRL_3$ value equal to 2. If we look at these two additions in terms of the $CTRL_3$ values we have

$$\begin{aligned}(2, 3) &\mapsto 5,\\ (5, 5) &\mapsto 2.\end{aligned} \tag{3.59}$$

  After one of each addition, the number of ciphertexts with $CTRL_3$ value equal to 3 and 5 have reduced by 1. the number of ciphertexts with $CTRL_3$ value equal to 2 remains the same.

- The main idea in this algorithm is to alternate between the two additions forementioned where possible since these are two of our good cases. To make as many of these additions happen as possible, the algorithm aims to equalise the number of ciphertexts with $CTRL_3$ value equal to 3, with ciphertexts with $CTRL_3$ value equal to 5. Also, since the number of ciphertexts with $CTRL_3$ value equal to 2 doesn't decrease in the additions listed above, the algorithm also aims to reduce the number of these ciphertexts to 1.

- Finally, if none of the above options remains, we try to add two ciphertexts whose vertex share an edge (i.e. a good addition case). Failing that we just add any remaining ciphertexts.

3.3.3 Alternative methods

Increasing the number of operations

Up till now, it has been assumed that error correction is performed immediately after performing one addition. Appendix A.2 gives an example of a circuit where more than one addition is performed. To perform more additions, we simply perform a permutation on the qubits so they are in the correct order to be input into the addition circuit. Increasing the number of additions performed before error correction will increase the number of terms in the superposition. Adding 3 ciphertexts together gives a superposition of the form

$$\frac{|B\mathbf{m} \diamond (-3\sigma)\rangle + \sqrt{3}|B\mathbf{m} \diamond (-\sigma)\rangle + \sqrt{3}|B\mathbf{m} \diamond \sigma\rangle + |B\mathbf{m} \diamond (3\sigma)\rangle}{2\sqrt{2}}. \tag{3.60}$$

The advantage of this approach is the superposition already contains the 2 terms for the desired final superposition. The only error correction that needs performing here is reducing the 2 outer terms to 0.

Example. Suppose we have a superposition of the form

$$\frac{|1010\cdots\rangle + \sqrt{3}|1110\cdots\rangle + \sqrt{3}|1001\cdots\rangle + |1101\cdots\rangle}{2\sqrt{2}}. \tag{3.61}$$

For ease of notation, assume the error $\sigma = 1$. The second, third and fourth digits are the only ones that differ in each term. Now if we measure the fourth qubit, the second and third qubits have the following form

$$\text{Fourth qubit} = 0\text{:} \begin{bmatrix} a_{00} \\ a_{10} \\ a_{01} \\ a_{11} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix}, \text{Fourth qubit} = 1\text{:} \begin{bmatrix} a_{00} \\ a_{10} \\ a_{01} \\ a_{11} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \\ 0 \\ 0 \end{bmatrix}. \tag{3.62}$$

We now wish to define the gates that will transform this superposition of 2 terms into the correct ciphertext of the sum. Firstly we apply a gate to make the distribution of the non zero probability qubits uniform.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{\sqrt{2}+i\sqrt{6}}{4} & \frac{\sqrt{6}-i\sqrt{2}}{4} \\ 0 & 0 & \frac{\sqrt{6}-i\sqrt{2}}{4} & \frac{\sqrt{2}+i\sqrt{6}}{4} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}. \tag{3.63}$$

$$\begin{bmatrix} \frac{\sqrt{2}+i\sqrt{6}}{4} & \frac{\sqrt{6}-i\sqrt{2}}{4} & 0 & 0 \\ \frac{\sqrt{6}-i\sqrt{2}}{4} & \frac{\sqrt{2}+i\sqrt{6}}{4} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} \\ \frac{1}{2} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{bmatrix}. \tag{3.64}$$

Finally, we now apply a gate on the second, third and fourth qubits to leave us only with the correct qubits, with no unwanted extra error terms.

$$\begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a_{000} \\ a_{100} \\ a_{010} \\ a_{110} \\ a_{001} \\ a_{101} \\ a_{011} \\ a_{111} \end{bmatrix}. \tag{3.65}$$

$$\begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a_{000} \\ a_{100} \\ a_{010} \\ a_{110} \\ a_{001} \\ a_{101} \\ a_{011} \\ a_{111} \end{bmatrix}. \tag{3.66}$$

3.3.4 Updating the algorithm

The whole point of the error-correcting method is that it is possible to tell which kind of error is being dealt with. Instead of dealing with the difficult $CTRL_3$ cases or being selective about which ciphertexts can be added together, the best course of action may be to alter the algorithm itself.

Step 4 in the online addition method is the step that deals with $CTRL_3$ values of 6,7,0 and 1. The step is now replaced with the following substeps.

(4.1) If $CTRL_3 = 0$ or 1, add σ to the ciphertext. Now perform the hadamard gate on the $k + 2^{th}$ qubit and an X gate on the $k + 1^{th}$ qubit. Move to step 7.

(4.2) If $CTRL_3 = 6$ or 7, subtract σ from the ciphertext. Now perform a Z gate on the $k + 2^{th}$ qubit.

(4.3) Now perform the hadamard gate on the $k + 2^{th}$ qubit and an X gate on the $k + 1^{th}$ qubit.

(4.4) Add σ to the ciphertext.

The following is an example using the updated algorithm. Note here that to avoid a conflict in notation, if an addition sign appears outside of brackets, that will be standard addition. Addition used to represent superposition will always be within brackets.

Example. Suppose we have two ciphertexts $\mathbf{c}_1 = \frac{|9\rangle + |11\rangle}{\sqrt{2}}$ and $\mathbf{c}_2 = \frac{|20\rangle + |22\rangle}{\sqrt{2}}$. With these 2 inputs, represented in binary, the addition algorithm proceeds as follows:

1 Addition of the ciphertexts, followed by the ϵ correction gives

$$(\frac{|100100\rangle}{\sqrt{2}} + \frac{|110100\rangle}{\sqrt{2}}) + (\frac{|001010\rangle}{\sqrt{2}} + \frac{|011010\rangle}{\sqrt{2}}) - |100000\rangle = (\frac{|001110\rangle}{2} + \frac{|011110\rangle}{\sqrt{2}} + \frac{|000001\rangle}{2}). \tag{3.67}$$

2 Now upon measurement of the 2^{nd} qubit, a 0 is observed. The superposition collapses down to

$$(\frac{|001110\rangle}{\sqrt{2}} + \frac{|000001\rangle}{\sqrt{2}}). \tag{3.68}$$

3 The sum of the $CTRL_3$ variables minus 1 is 6 therefore the previous measurement hasn't collapsed down to a single value.

4.2 Subtracting 1 first gives $(\frac{|110110\rangle}{\sqrt{2}} + \frac{|111110\rangle}{\sqrt{2}})$. Now applying the first stage of error correction

$$|11\rangle Z(\frac{0}{\sqrt{2}} + \frac{1}{\sqrt{2}})|110\rangle = |11\rangle(\frac{0}{\sqrt{2}} - \frac{1}{\sqrt{2}})|110\rangle. \tag{3.69}$$

4.3 The second error correction gates are applied giving

$$|1\rangle X|1\rangle H(\frac{0}{\sqrt{2}} - \frac{1}{\sqrt{2}})|110\rangle = |101110\rangle. \tag{3.70}$$

4.4 Re-adding the error just taken off in (4.2) produces the ciphertext with no error component $|011110\rangle$.

7 Finally applying the error outputs the correct summed ciphertext

$$|011110\rangle + (\frac{|000000\rangle + |010000\rangle}{\sqrt{2}}) = \frac{|30\rangle}{\sqrt{2}} + \frac{|32\rangle}{\sqrt{2}}. \tag{3.71}$$

3.4 A multiplication method for the quantum ciphertext

3.4.1 Potential functions

As discussed already in the previous chapter, for meaningful computation to be possible, a multiplication method is needed. Although multiplication of qubits has been established [Ekert *et al.* (2001)], a method of multiplication of ciphertexts in this vector space may not be as natural as addition.

A somewhat homomorphic encryption scheme based on the ring learning with errors problem has been presented in [Naehrig *et al.* (2011)]. When the data is encrypted, a finite number of addition and multiplication operations can be performed. After this limit has been reached the data needs to be decrypted and encrypted again to reduce the error growth. Despite this, the paper discusses statistical functions that can be performed on the ciphertexts.

Firstly, the mean, $m = \frac{\sum_{i=1}^{n} c_i}{n}$, is not a function that requires multiplication. Addition of the ciphertexts, $c_1, \ldots, c_n$, can be performed with the error correction method and the total number of ciphertexts can easily be counted. The pair $(\sum_{i=1}^{n} c_i, n)$ is returned to the offline device to perform the final division step.

The standard deviation is defined as $s = \sqrt{\frac{\sum_{i=1}^{n}(c_i - m)^2}{n}}$. Its calculation begins with a series of additions, followed by squaring, followed again by addition. In the RLWE scheme, the denominator and numerator are once again returned to be processed offline.

Logistic regression is the final function discussed. Here, the input for the prediction function $x = \sum_{i=1}^{n} \alpha_i x_i$, where α_i is a weighting constant. The prediction function, $f(x) = \frac{e^x}{1+e^x}$, is simple enough that this phase can be calculated offline. Therefore the operations performed online are a series of scalar multiplications followed by a sum of the resulting values.

Due to the component-wise property of addition in vectors, it seems reasonable to look for a component-wise multiplication method in this vector space. If both addition and multiplication have this property, the vectors being operated on will be able to be processed in parallel.

3.4.2 Component wise multiplication operation

Although component wise multiplication of the form

$$\mathbf{V}_1 \Box \mathbf{V}_2 = [\alpha_1, \ldots, \alpha_n] \diamond [\beta_1, \ldots, \beta_n] = [\alpha_1\beta_1, \ldots, \alpha_n\beta_n], \tag{3.72}$$

would be useful, the encryption method is not homomorphic with respect to this operation. Let

$$B = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{bmatrix}. \tag{3.73}$$

be the public basis used for encryption. Given plaintexts, $\mathbf{m}_1 = [\alpha_1, \ldots, \alpha_n]$ and $m_2 = [\beta_1, \ldots, \beta_n]$, attempting to perform this operation on ciphertexts $\mathbf{c}_1 = E(\mathbf{m}_1)$ and $\mathbf{c}_2 = E(\mathbf{m}_2)$ would give

$$\begin{aligned} \mathbf{c}_1 \diamond \mathbf{c}_2 &= \begin{bmatrix} b_{1,1}\alpha_1 + \cdots + b_{1,n}\alpha_n + \epsilon \\ b_{2,1}\alpha_1 + \cdots + b_{2,n}\alpha_n + \epsilon \\ \vdots \\ b_{n,1}\alpha_1 + \cdots + b_{n,n}\alpha_n + \epsilon \end{bmatrix} \square \begin{bmatrix} b_{1,1}\beta_1 + \cdots + b_{1,n}\beta_n + \epsilon \\ b_{2,1}\beta_1 + \cdots + b_{2,n}\beta_n + \epsilon \\ \vdots \\ b_{n,1}\beta_1 + \cdots + b_{n,n}\beta_n + \epsilon \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^n \sum_{j=1}^n b_{1,i}b_{1,j}\alpha_i\beta_j + \epsilon\sum_{i=1}^n b_{1,i}\alpha_i + \epsilon\sum_{j=1}^n b_{1,j}\beta_j + \epsilon^2 \\ \sum_{i=1}^n \sum_{j=1}^n b_{2,i}b_{2,j}\alpha_i\beta_j + \epsilon\sum_{i=1}^n b_{2,i}\alpha_i + \epsilon\sum_{j=1}^n b_{2,j}\beta_j + \epsilon^2 \\ \vdots \\ \sum_{i=1}^n \sum_{j=1}^n b_{n,i}b_{n,j}\alpha_i\beta_j + \epsilon\sum_{i=1}^n b_{n,i}\alpha_i + \epsilon\sum_{j=1}^n b_{n,j}\beta_j + \epsilon^2 \end{bmatrix} \\ &\neq \begin{bmatrix} \sum_{j=1}^n b_{1,j}\alpha_j\beta_j + \epsilon \\ \sum_{j=1}^n b_{2,j}\alpha_j\beta_j + \epsilon \\ \vdots \\ \sum_{j=1}^n b_{n,j}\alpha_j\beta_j + \epsilon \end{bmatrix} = E(\mathbf{m}_1 + \mathbf{m}_2). \end{aligned} \tag{3.74}$$

The $\epsilon\sum_{i=1}^n b_{\cdot,i}\alpha_i + \epsilon\sum_{j=1}^n b_{\cdot,j}\beta_j + \epsilon^2 = \epsilon(\sum_{i=1}^n b_{\cdot,i}\alpha_i + \sum_{j=1}^n b_{\cdot,j}\beta_j + \epsilon)$ part of each sum could be considered as the error term of a ciphertext. While the form of the error looks like it would be complicated to deal with, the bigger issue is the remaining summation. The product of α and β is multiplied by the public basis values twice.

Although it can't be used for decryption, the inverse of the public matrix can be used to make an operation that behaves more like the homomorphism required. The inverse matrix can be written as

$$B^{-1} = \begin{bmatrix} b'_{1,1} & b'_{1,2} & \cdots & b'_{1,n} \\ b'_{2,1} & b'_{2,2} & \cdots & b'_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b'_{n,1} & b'_{n,2} & \cdots & b'_{n,n} \end{bmatrix}. \tag{3.75}$$

(Note $b'_{i,j}$ is not the inverse of $b_{i,j}$).

If the inverse matrix is used on each ciphertext then, after performing the component multiplication, it will be necessary to multiply the result by the public basis matrix again.

The operation therefore will be of the form

$$\mathbf{V}_1 \Box_B \mathbf{V}_2 = B(d_1 \Box d_2), \tag{3.76}$$

where $d_i = B^{-1}\mathbf{v}_i$. Consider the form of the multiplication expressed in terms of the plaintext

$$B[(B^{-1}\mathbf{c}_1)\Box(B^{-1}\mathbf{c}_2)] = B\left(\begin{bmatrix}\alpha_1 + \epsilon(b'_{1,1} + b'_{1,2} + \cdots + b'_{1,n})\\ \alpha_2 + \epsilon(b'_{2,1} + b'_{2,2} + \cdots + b'_{2,n})\\ \vdots \\ \alpha_n + \epsilon(b'_{n,1} + b'_{n,2} + \cdots + b'_{n,n})\end{bmatrix} \Box \begin{bmatrix}\beta_1 + \epsilon(b'_{1,1} + b'_{1,2} + \cdots + b'_{1,n})\\ \beta_2 + \epsilon(b'_{2,1} + b'_{2,2} + \cdots + b'_{2,n})\\ \vdots \\ \beta_n + \epsilon(b'_{n,1} + b'_{n,2} + \cdots + b'_{n,n})\end{bmatrix}\right). \tag{3.77}$$

Here the error is only being multiplied by elements of B^{-1}. The error is therefore a scalar multiple of the original error so we can work out a new error correction method. Denote $b'_i = b'_{i,1} + b'_{i,2} + \cdots + b'_{i,n}$. Then

$$B[(B^{-1}\mathbf{c}_1)\Box(B^{-1}\mathbf{c}_2)] = B\left(\begin{bmatrix}\alpha_1 + \epsilon b'_1\\ \alpha_2 + \epsilon b'_2\\ \vdots \\ \alpha_n + \epsilon b'_n\end{bmatrix} \Box \begin{bmatrix}\beta_1 + \epsilon b'_1\\ \beta_2 + \epsilon b'_2\\ \vdots \\ \beta_n + \epsilon b'_n\end{bmatrix}\right) \tag{3.78}$$

The result is

$$B[(B^{-1}\mathbf{c}_1)\Box(B^{-1}\mathbf{c}_2)] = B(\mathbf{c}_1 \diamond \mathbf{c}_2) + \begin{bmatrix}\epsilon b'_1(\alpha_1 + \beta_1)\\ \epsilon b'_2(\alpha_2 + \beta_2)\\ \vdots \\ \epsilon b'_n(\alpha_n + \beta_n)\end{bmatrix}. \tag{3.79}$$

Assuming the product of the 2 error terms is still of an appropriate size, the result is a correct encryption plus a product of error and data. As already stated, the b' terms are fixed. Therefore, assuming the data being analysed is bounded by sufficiently small constants, this product can still be correctly decrypted with the appropriate choice of parameters for the protocol.

We have shown that although component wise multiplication is not homomorphic under our encryption function, we can adapt the multiplication method used on our ciphertexts to give us an acceptably close product.

In this chapter, it has been established that as many additions as needed can be performed. This was all that was required for calculating the mean and now it can be used on a data set of any size. A multiplication can be performed assuming decryption allows an increase in error size (error now a function of the data). For the standard deviation method, the $(c_i - m)^2$ part of the calculation can certainly be performed. The final addition part poses a problem as the error at this point is unknown. Scalar multiplication however, can be dealt with. Because the α_i in the regression are already established, adapting the error method from the original ϵ to $\alpha_i\epsilon$ for $1 \leq i \leq n$ will make the calculation of the input for the prediction function x possible.

Chapter 4

Ideal membership protocol

In this chapter, we consider cryptography schemes that use the *ideal membership problem* as the foundation of their security. The membership problem has the following definition. We will begin by defining the problem of interest in section 1, along with the algorithm by Buchberger that makes the ideal membership an interesting problem for cryptography. We will build on this in section 2 by looking at Gröbner bases in a non-commutative setting. Here we will look at work by Mora to adapt Buchberger's algorithm for non-commutative ideals.
The remaining sections will contain original work with section 3 establishing a key exchange protocol and addressing some potential weaknesses. Finally, in section 4 we will consider a Hecke algebra setting for the same protocol and examine the advantages gained from moving into this setting.

4.1 Gröbner basis solution to the membership problem

Definition. Let $R = k[x_1, x_2, \ldots, x_n]$ be a polynomial ring over k. Suppose the polynomials $f_0, f_1, \ldots, f_m \in R$. Let $I = \langle f_1, f_2, \ldots, f_m \rangle \subset R$. Is it true that $f_0 \in I$?

The introduction of Gröbner bases has given us a representation of ideals that make it a much less daunting test to check membership [April *et al.* (2012)]. To start with let's state the definition and an important theorem about Gröbner bases which cause a key difference between the commutative and non-commutative case. We use the notation of chapter 2, for leading monomials and leading terms, $LM(f)$ and $LT(f)$, of a polynomial f.

Definition. Given f, g, h in $k[x_1, \ldots, x_n]$, $g \neq 0$. If $LM(g)$ divides a non-zero term X of f then the polynomial

$$h = f - \frac{X}{LT(g)} \cdot g \tag{4.1}$$

is called a *one-step reduction* of f modulo g, and we write

$$f \xrightarrow{g} h. \tag{4.2}$$

Furthermore, suppose we have a collection of polynomials $F = \{f_1, \dots, f_s\}$. We say f *reduces* to h modulo F, which we express as

$$f \xrightarrow{F}_{+} h, \tag{4.3}$$

if and only if there exists a sequence of indices $i_1, i_2, \dots, i_t \in \{1, \dots, s\}$ and a sequence of polynomials $h_1, \dots, h_{t-1} \in k[x_1, \dots, x_n]$ such that

$$f \xrightarrow{f_{i_1}} h_1 \xrightarrow{f_{i_2}} h_2 \xrightarrow{f_{i_3}} \cdots \xrightarrow{f_{i_{t-1}}} h_{t-1} \xrightarrow{f_{i_t}} h. \tag{4.4}$$

Definition. A set of non-zero polynomials $G = \{g_1, \dots, g_t\}$ contained in an ideal I, is called a *Gröbner basis* for I if and only if for all $f \in I$, $f \neq 0$, there exists $i \in \{1, \dots, t\}$ such that $\mathrm{LM}(g_i)$ divides $\mathrm{LM}(f)$.

We are going to show how to transform a given basis for an ideal into a Gröbner basis. Any element of an ideal generated by $f_1, \dots, f_s$ can be expressed as

$$f = \sum_{i=1}^{s} h_i f_i, \tag{4.5}$$

where $h_i \in k[x_1, \dots, x_n]$. The largest of the $LM(h_i f_i) = LM(h_i)LM(f_i)$, i.e. the leading monomial of f, is divisible by $LM(f_i)$, therefore we need to worry about instances where this monomial cancels out in order to complete our Gröbner basis. This prompts the following definition.

Definition. Let $f, g \in k[x_1, \dots, x_n]$ be non-zero. Let $L = lcm(LM(f), LM(g))$. The polynomial

$$S(f, g) = \frac{L}{lt(f)} f - \frac{L}{lt(g)} g \tag{4.6}$$

is called the *S-polynomial* of f and g.

Buchberger showed that these S-polynomials are key to creating a Gröbner basis with the following theorem.

Theorem 4.1 (Buchberger (1965))**.** *Let $G = \{g_1, \dots, g_t\}$ be a set of non-zero polynomials in $k[x_1, \dots, x_n]$. Then G is a Gröbner basis for the ideal $I = \langle g_1, \dots, g_t \rangle$ if and only if $\forall i \neq j$,*

$$S(g_i, g_j) \xrightarrow{G}_{+} 0. \tag{4.7}$$

Theorem 4.2 (Adams *et al.* (1994))**.** *Let I be a non-zero ideal of $k[x_1, \dots, x_n]$. The following statements are equivalent for a set of non-zero polynomials $G = \{g_1, \dots, g_t\} \subseteq I$.*

1. *G is a Gröbner basis for I.*
2. *$f \in I$ if and only if $f \xrightarrow{G}_+ 0$.*
3. *$f \in I$ if and only if $f = \sum_{i=1}^{t} h_i g_i$ with $LM(f) = max_{1 \leq i \leq t}(LM(h_i)LM(g_i))$.*
4. *$LT(I) = LT(G)$, where $LT(S) = \langle LT(s) | s \in S \rangle$.*

Algorithm 2 Buchberger's Algorithm

```
1: procedure BUCHBERGER($F$), $F = \{f_1, \dots, f_s\} \subseteq k[x_1, \dots, x_n]$ with $f_i \neq 0$ $(1 \leq i \leq s)$
2:     $G := F$
3:     $\mathcal{G} := \{\{f_i, f_j\} | f_i \neq f_j \in G\}$
4:     while $\mathcal{G} \neq \emptyset$ do
5:         Choose any $\{f, g\} \in \mathcal{G}$
6:         $\mathcal{G} := \mathcal{G} - \{\{f, g\}\}$
7:         $S(f, g) \xrightarrow{G}_+ h$, where $h$ is reduced WRT $G$
8:         if $h \neq 0$ then
9:             $\mathcal{G} := \mathcal{G} \cup \{\{u, h\}|$ for all $u \in G\}$
10:            $G := G \cup \{h\}$
11:    return $G = \{g_1, \dots, g_t\}$ a Gröbner basis for $\langle f_1, \dots, f_s \rangle$
```

Hilbert's basis theorem tells us an important property about the termination of Buchberger's algorithm [Adams *et al.* (1994)].

Theorem 4.3 (Hilbert (1890)). *The following two properties hold in the commutative ring $k[x_1, \dots, x_n]$:*

- *If I is any ideal of $k[x_1, \dots, x_n]$, then there exists polynomials $f_1, \dots, f_s \in k[x_1, \dots, x_n]$ such that $I = \langle f_1, \dots, f_s \rangle$.*
- *If $I_1 \subseteq I_2 \subseteq I_3 \subseteq \cdots \subseteq I_n \subseteq \cdots$ is an ascending chain of ideals of $k[x_1, \dots, x_n]$, then there exists N such that $I_N = I_{N+1} = I_{N+2} = \dots$.*

Theorem 4.4 (Becker & Weispfenning (1993)). *Given $F = \{f_1, \dots, f_s\}$, all non-zero, Buchberger's algorithm will produce a Gröbner basis for the ideal $I = \langle f_1, \dots, f_s \rangle$.*

Proof. Let us assume the algorithm does not terminate. As the algorithm runs a new set G_i will be constructed from G_{i-1} creating a strictly infinite sequence

$$G_1 \subsetneq G_2 \subsetneq G_3 \subsetneq \dots \tag{4.8}$$

Each iteration a polynomial, h, a non-zero reduction with respect to G_{i-1} is added to form G_i. Because it is reduced WRT G_{i-1}, it must be the case that $LT(h) \notin LT(G_{i-1})$. This means that there is a strictly ascending chain of ideals

$$LT(G_1) \subsetneq LT(G_2) \subsetneq LT(G_3) \subsetneq \dots, \tag{4.9}$$

which contradicts Hilbert's basis theorem.
Because only elements from the ideal generated by f are added at each iteration $I = \langle f_1, \dots, f_s \rangle \subseteq \langle g_1, \dots, g_t \rangle \subseteq I$, therefore G is a generating set for I. □

4.2 Non-commutative Gröbner bases

In order to find Gröbner bases in non-commutative variables, we must first understand the concept of overlaps and from that how to perform division [Evans (2006)].

Definition. Let p_1 and p_2 be elements of a polynomial ring with an admissible order O. We say that p_2 can be *reduced* by p_1 if $\mathrm{LM}(p_1)$ divides a monomial m in p_2. This means that $m = m_l \cdot LM(p_1) \cdot m_r$, for some monomials m_l and m_r. The calculation of the reduction is

$$p_2 - (c \cdot LC(p_1)^{-1}) m_l p_1 m_r, \tag{4.10}$$

where c is the coefficient of m in p_2.

S-polynomials, which we determine below, allow us to ensure that if we have a polynomial p that is reducible by 2 other polynomials p_1 and p_2, we can achieve a unique remainder when reducing p by a set of polynomials containing p_1 and p_2. In the commutative case there is only one way to reduce one polynomial by another. When in the non-commutative setting however there may be multiple ways for us to reduce one polynomial by another and thus we need to understand what it means for polynomials to overlap [Evans (2006)].

Definition. For a monomial m in a non-commutative polynomial ring define:

- *prefix*(m, i) to be the initial subword of length i of m,
- *suffix*(m, i) to be the terminal subword of length i of m,
- *subword*(m, i, j) to be the subword starting at position i and ending at position j in m.

Definition. Two monomials m_1 and m_2 with degrees $d_1 \geq d_2$ (respectively) *overlap* if any of the following conditions are true

- $\text{prefix}(m_1, i) = \text{suffix}(m_2, i)$ $(1 \leq i < d_2)$,

- subword$(m_1, i, i + d_2 - 1) = m_2$ $(1 \le i \le d_1 - d_2 + 1)$,
- suffix$(m_1, i) =$ prefix(M_2, i) $(1 \le i < d_2)$.

Definition. Assume we have polynomials p_1 and p_2 such that $l_1 \cdot LM(p_1) \cdot r_1 = l_2 \cdot LM(p_2) \cdot r_2$, where we choose l_1, l_2, r_1 and r_2 in a way such that at least one of l_1 and l_2 and at least one of r_1 and r_2 are equal to 1. The *S-polynomial* associated with this overlap is

$$\text{S-pol}(l_1, p_1, l_2, p_2) = c_1 l_1 p_1 r_1 - c_2 l_2 p_2 r_2, \tag{4.11}$$

where $c_1 = LC(p_2)$ and $c_2 = LC(p_1)$.

With these definitions in mind, we can now state the non-commutative version of Buchberger's algorithm. $Rem(s_1, G)$ represents a reduction of s_1 by all of the polynomials in G.

Algorithm 3 Mora's Algorithm [Mora (1985)]

```
1: procedure MORA($F$), $F = \{f_1, \dots, f_s\}$ is a basis for an ideal $J$ over a non-commutative polynomial ring
2:     $G := F$
3:     $A := \emptyset$
4:     For each pair of polynomials $(g_i, g_j)$ in $G$, add an S-Polynomial to $A$ for each of the overlaps of the lead monomials of $g_i$ and $g_j$
5:     while $A \neq \emptyset$ do
6:         Remove first entry $s_1$ from $A$;
7:         $s_1' = \text{Rem}(s_1, G)$
8:         if $s_1' \neq 0$ then
9:             Add $s_1'$ to $G$ and then add all the S-polynomials of the form S-pol$(l_1, g_i, l_2, s_1')$ to $A$ $\forall g_i \in G$
10:    return $G = \{g_1, \dots, g_t\}$ a Gröbner basis for $J$ (Assuming termination)
```

While we now have an algorithm to calculate a Gröbner basis, there is a lot of redundancy in the generators. We can take it one step further to remove some of the unneeded elements from the basis found.

Definition. Let $G = \{g_1, \dots, g_p\}$ be a Gröbner basis for an ideal in a polynomial ring. Then G is a *reduced* Gröbner basis if the following 2 conditions are true.

- $LC(g_i) = 1$ for all $g_i \in G$.
- None of the terms in the polynomials $g_i \in G$ are divisible by any LT(g_j), $j \neq i$.

Once we have a Gröbner basis, we can begin to test if a given polynomial is a member of the ideal generated by that basis. Because we are now using a Gröbner basis, this algorithm has a much lower complexity than if we had tried to perform membership testing with any basis for an ideal.

Algorithm 4 Membership testing

```
1: procedure INPUT($G = \{g_1, g_2, \dots, g_t\},\ f$), where $G$ is a Gröbner basis and $f$ is a polynomial to test if it is a member of the ideal generated by $G$.
2:     $p = f$.
3:     Reduction = TRUE.
4:     While Reduction == TRUE
5:         if $\exists g_i \in G$ s.t. $LM(p) = l \cdot LM(g_i) \cdot r$ for some non zero monomials $l, r$ and $1 \leq i \leq t$.
6:             $p = p - l \cdot g_i \cdot r$.
7:         else
8:             Reduction = FALSE.
9:     if $p = 0$
10:        return TRUE.
11:    else
12:        return FALSE, $p$.
```

The algorithm below takes as input the Gröbner basis calculated in Mora's algorithm and outputs the unique reduced Gröbner basis.

Algorithm 5 Noncommutative unique reduced Gröbner basis

```
1: procedure REDUCE($G$), $G = \{g_1, \dots, g_m\}$ is a basis for an ideal $j$ over a noncommutative ring
2:     $G' := \emptyset$
3:     for each $g_i \in G$ do
4:         Multiply $g_i$ by LC$(g_i)^{-1}$;
5:         if (LM$(g_i)$ = $l$LM$(g_j)r$ for some monomials $l, r$ and some $g_j \in G$ $(g_j \neq g_i)$) then
6:             $G = G \backslash \{g_i\}$;
7:         end if
8:     end for
9:     for each $g_i \in G$ do
10:        $g_i' = \text{Rem}(g_i, (G\backslash\{g_i\}) \cup G'$;
11:        $G = G\backslash\{g_i\}; G' = G' \cup \{g_i'\}$;
12:    end for
13:    return $G' = \{g_1', \dots, g_p'\}$ a unique reduced Gröbner basis for $J$
```

Because non-commutative polynomials do not satisfy a theorem similar to Hilbert's basis theorem, we can not guarantee that Mora's algorithm will terminate. Indeed we have the following theorem.

Theorem 4.5 (Rai (2004))**.** *Let K be a finite field and $K\langle x, y, z\rangle$ the noncommutative free algebra over K. Let $g_1 = xzy + yz \in K\langle x, y, z\rangle$, $g_2 = yzx + zy \in K\langle x, y, z\rangle$. Then, $I = \langle g_1, g_2\rangle$ does not have a finite Gröbner basis under any admissible order.*

Polly Cracker is a name used to encompass a variety of cryptosystems that make use of Gröbner bases [Ackermann & Kreuzer (2006)]. The non-commutative Polly Cracker cryptosystem takes advantage of the existence of ideals that do not have a finitely generated Gröbner basis. In the next section, we present our version in the form of a key exchange protocol.

4.3 A key exchange protocol based on the ideal membership problem

In this section, we will attempt to adapt the non-commutative encryption process [Rai (2004)] into a key exchange. From there we will use the key generated from such an exchange to introduce a simple encryption function that has homomorphic properties. The remainder of the section we will discuss a potential attack and discuss how we can set up the scheme in such a way that is resistant to this kind of attack.

4.3.1 Agreeing in secret on a key from a public list of polynomials

Make public an ideal $I \subset K\langle x_1, \dots, x_n\rangle$, for which Mora's algorithm will not terminate. Alice makes her choice of private Gröbner basis

$$G_A = \{g_{A,1}, g_{A,2}, \dots, g_{A,n_A}\} \tag{4.12}$$

for a super ideal I_A of the public key I.
Alice secretly generates a polynomial as a linear combination of generators of her private ideal

$$S_{A,0} = \sum_{i=1}^{n_A} l_i g_{A,i} r_i \in I_A, \tag{4.13}$$

where l_i, $r_i \in K\langle x_1, \dots, x_n\rangle$. She also generates a collection of m elements that aren't in the public ideal, but instead are each in pairwise distinct cosets of I_A. We do not need to choose a polynomial from every coset and in fact, a suggested number of polynomials is the average number of terms that each of those polynomials contains (e.g. if the polynomials chosen each have 5 terms, then 5 cosets should be used). This choice isn't extremely important in this section but in the next section we will look at adapting the scheme and

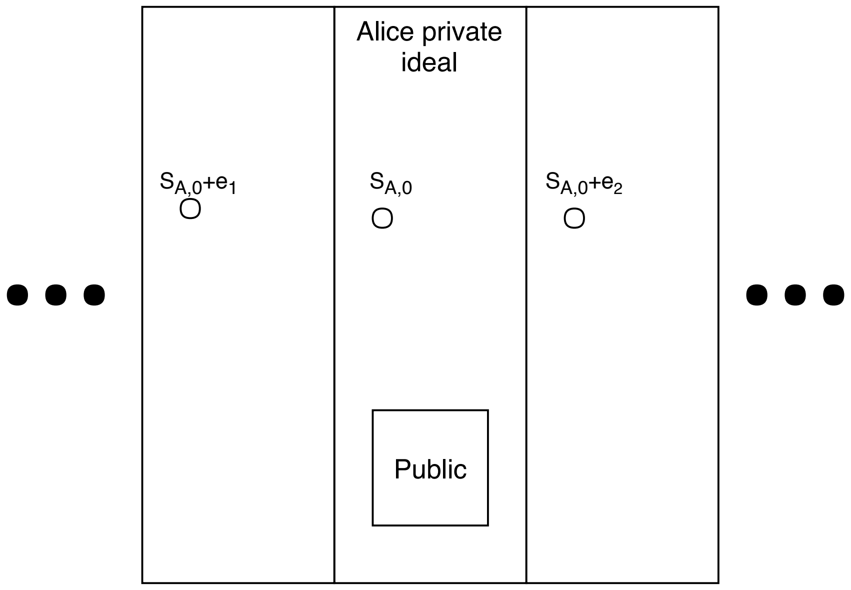


Figure 4.1: Alice selects polynomials from her ideal and its cosets

this choice will prove to be efficient. The m elements generated are denoted

$$\begin{aligned} S_{A,1} &= \sum_{i=1}^{n_A} l_i g_{A,i} r_i + \epsilon_1 \notin I_A, \\ &\vdots \\ S_{A,m} &= \sum_{i=1}^{n_A} l_i g_{A,i} r_i + \epsilon_m \notin I_A, \end{aligned} \tag{4.14}$$

where $\epsilon_i - \epsilon_j \notin I_A$, $\forall i \neq j$.

Figure 4.1 gives an idea of how Alice generates her polynomials. Although the public ideal (and by extension Alice's private ideal) are infinite in size, the figure gives an impression of how the polynomial ring has been split up and how Alice chooses one polynomial per coset.

$S_{A,0}, S_{A,1}, \ldots, S_{A,m}$ are sent over a public channel to Bob in a random order known to Alice. A potential eavesdropper has access only to the public ideal and it should therefore be difficult for them to work out which $S_{A,i}$ is a member of Alice's ideal.

Bob now chooses an arbitrary element of the public ideal, denoted S_B. He chooses at random one of Alice's messages, $S_{A,j}$ that he wants to be the shared key. He calculates the difference between his own element and his choice from Alice. This gives him

$$B_{\text{diff}} = S_B - S_{A,j}. \tag{4.15}$$

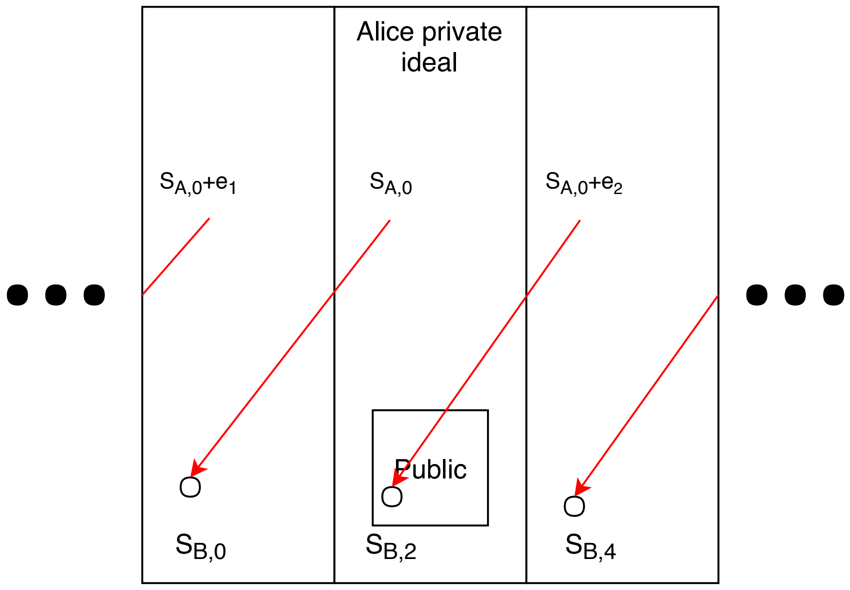


Figure 4.2: Bob adds a polynomial to each of Alice's so one lies in the public ideal

Bob adds B_{diff} to each of Alice's messages giving the list

$$\begin{aligned} S_{B,0} &= S_{A,0} + B_{\text{diff}} \\ S_{B,1} &= S_{A,1} + B_{\text{diff}} \\ &\vdots \\ S_{B,j} &= S_{A,j} + B_{\text{diff}} \in I \\ &\vdots \\ S_{B,m} &= S_{A,m} + B_{\text{diff}} \end{aligned} \tag{4.16}$$

Alice receives all these updated versions of her messages from Bob in the same order she sent them. Bobs goal is to use an element of the public ideal as the secret key. In Figure 4.2 $j = 2$ so $S_{B_2} \in I$ but $S_{B_k} \notin I$, when $k \neq 2$.
Once again any eavesdropper only has the public ideal to work with, so they cannot tell which element Bob chose to be in the ideal. Alice then reduces the element that corresponded to her choice of $S_{A,0}$. If this is 0 then Bob has chosen that element, if there was another polynomial with that property, say S_{B_k}, then

$$\begin{aligned} S_{B_k} \in I_A \Rightarrow S_{A_k} + B_{\text{diff}} &\in I_A \\ S_{A_j} - S_{A_k} &\in I_A \\ \epsilon_j - \epsilon_k &\in I_A. \end{aligned} \tag{4.17}$$

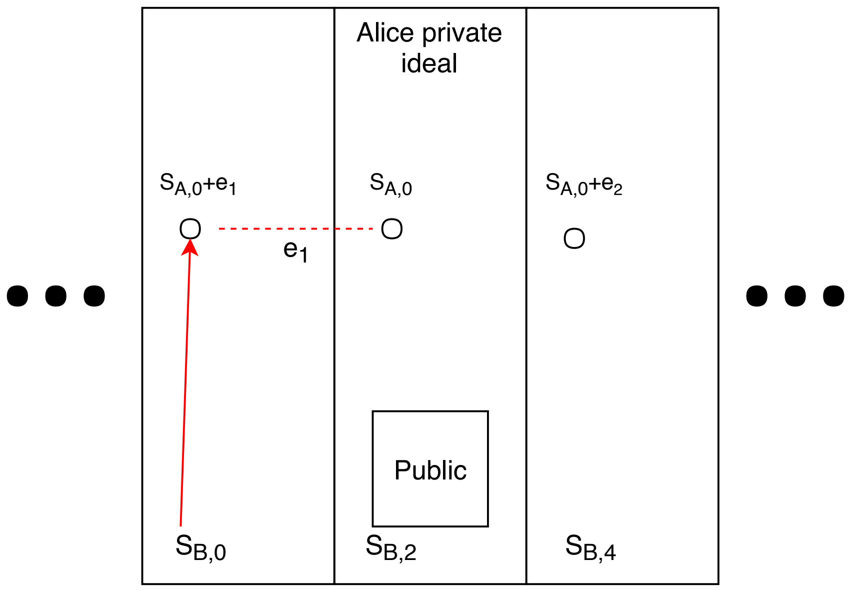


Figure 4.3: Alice knows the difference between her polynomials so can get back $S_{A,0}$

This is a contradiction.
If when Alice reduces Bob's polynomial and does not get a 0, then she can easily work out which polynomial Bob wanted by checking which ϵ_j she has remaining using

$$\begin{aligned} S_{B,0} &= S_{A,0} + B_{\text{diff}} \\ &= S_{A,0} + (S_B - S_{A,0} - \epsilon_j) \\ &= S_B - \epsilon_j, \end{aligned} \tag{4.18}$$

where S_B reduces to zero as the public ideal is contained in Alice's ideal. Figure 4.3 illustrates why it was so important that Bob added the same polynomial to each polynomial that he received from Alice. Even though Alice doesn't know which polynomial she receives from Bob is his public key choice, she can locate $S_{B,0}$ as the order is maintained. She then computes $S_{B,0} - S_{A,0} = (S_{A,0} - S_B) + \epsilon_j$ and can find ϵ_j since $S_{A,0} - S_B \in I$.
Alice and Bob use S_{B_j} as the secret key.

4.3.2 Encryption process

Building on the ideas presented in [Ko *et al.* (2000)] for an encryption scheme based on conjugacy, we will present a cryptosystem that uses our generated private keys. To develop a homomorphic encryption system we will need a pair (p, q) of private keys with particular properties, and we shall find these in a Hecke algebra setting, as will be described in Section 4.4. In this section, we shall describe the encryption process that uses that pair of keys. We will denote the private key p and a second private key q. The key q is not the inverse of p in the ring R, but in the factor ring R/J where

$$J = R\langle pq - 1, qp - 1\rangle R, \tag{4.19}$$

we have $pq = qp = 1$.

Encryption method:

$$Enc(m) = pmq + qmp. \tag{4.20}$$

While the encryption method needs to be simple to make it possible for weak devices to compute, the computational operations performed on the encrypted data and the decryption are performed on more powerful devices. The time consuming parts of the method, therefore, are concentrated on the latter.

Homomorphisms:

$$\begin{aligned} Enc(m_1) + Enc(m_2) &= pm_1q + qm_1p + pm_2q + qm_2p \\ &= p(m_1 + m_2)q + q(m_1 + m_2)p \\ &= Enc(m_1 + m_2) \end{aligned} \tag{4.21}$$

$$\begin{aligned} Enc(m_1) \cdot Enc(m_2) &= (pm_1q + qm_1p) \cdot (pm_2q + qm_2p) \\ &= pm_1qpm_2q + pm_1q^2m_2p + qm_1p^2m_2q + qm_1pqm_2p \\ &= pm_1m_2q + qm_1m_2p + pm_1q^2m_2p + qm_1p^2m_2q \\ &= Enc(m_1 \cdot m_2) + pm_1q^2m_2p + qm_1p^2m_2q \end{aligned} \tag{4.22}$$

We assume that we are in some way able to kill off the extra terms that come as a result of the multiplication. We wish to use the fact that there is a square of one of our private keys in each of these terms. A quotient space that incorporates this property could be used.

Decryption method:

$$Dec(c) = q \cdot Enc(m) \cdot p = qpmqp + q^2mp^2 = m + q^2mp^2. \tag{4.23}$$

In our quotient space the second term will be killed off, leaving us with just our original message.

Example. Key exchange

Alice and Bob agree to use the ideal I from Theorem 4.5 as their public information where $I = \langle xzy + yz, yzx + zy\rangle$. Alice creates her private ideal that has a finite Gröbner basis

$$I_A = \langle xzy + yz, yzx + zy, xz + x, z^4y - z^3y\rangle. \tag{4.24}$$

She also chooses her first polynomial

$$\begin{aligned} S_{A,0} &= y(xzy + yz)x + z(yzx + zy)y + (xz + x)z + y(z^4y - z^3y)y \\ &= yxzyx + y^2zx + zyzxy + z^2y^2 + xz^2 + xz + yz^4y^2 - yz^3y^2, \end{aligned} \tag{4.25}$$

followed by selecting coset elements $\epsilon_1 = y^3 + yzy - xz^2 + xz$ and $\epsilon_2 = z^5 - z^2yz$. She sends to Bob the polynomials $S_{A,1} = S_{A,0} + \epsilon_1$, $S_{A,0}$ and $S_{A,2} = S_{A,0} + \epsilon_2$ over a public channel, recording the order of the polynomials in which she sends.
Bob now chooses an element of the public ideal

$$\begin{aligned} S_B &= xy(xzy + yz)x + z(yzx + zx)yz \\ &= xyxzyx + xy^2zx + zyxyz + z^2xyz. \end{aligned} \tag{4.26}$$

He then chooses at random one of Alice's polynomials, say $S_{A,1}$. The difference between the two polynomials is

$$\begin{aligned} B_{diff} =& xyxzyx + xy^2zx + zyxyz + z^2xyz - yxzyx - \\ & y^2zx - zyzxy - z^2y^2 - y^3 - yzy - yz^4y^2 + yz^3y^2. \end{aligned} \tag{4.27}$$

He then sends back the list S_{B_1}, S_{B_0}, S_{B_2}, where $S_{B_1} = S_B$,

$$\begin{aligned} S_{B,0} =& yxzyx + y^2zx + zyzxy + z^2y^2 + xz^2 + xz + yz^4y^2 - yz^3y^2 \\ & + xyxzyx + xy^2zx + zyxyz + z^2xyz - yxzyx - y^2zx - zyzxy - z^2y^2 \\ & - y^3 - yzy - yz^4y^2 + yz^3y^2 \end{aligned} \tag{4.28}$$

and

$$\begin{aligned} S_{B,2} =& yxzyx + y^2zx + zyzxy + z^2y^2 + xz^2 + xz + yz^4y^2 - yz^3y^2 \\ & + z^5 - z^2yz + xyxzyx + xy^2zx + zyxyz + z^2xyz - yxzyx - y^2zx - zyzxy \\ & - z^2y^2 - y^3 - yzy - yz^4y^2 + yz^3y^2. \end{aligned} \tag{4.29}$$

Having received the 3 polynomials, Alice now looks at $S_{B,0}$.

$$\begin{aligned} S_{B,0} =& [y(xzy + yz)x + z(yzx + zy)y + (xz + x)z + y(z^4y - z^3y)y + xy(xzy + yz)x \\ & + z(yzx + zx)yz - y(xzy + yz)x - z(yzx + zy)y - y(z^4y - z^3y)y] - y^3 - yzy. \end{aligned} \tag{4.30}$$

Alice can reduce everything within the square brackets to zero using her generators in the membership testing algorithm leaving just $-y^3 - yzy$. This tells Alice that Bob chose to use the polynomial associated with ϵ_1 and she can subsequently determine the shared private key S_B.

Encryption, operations and decryption

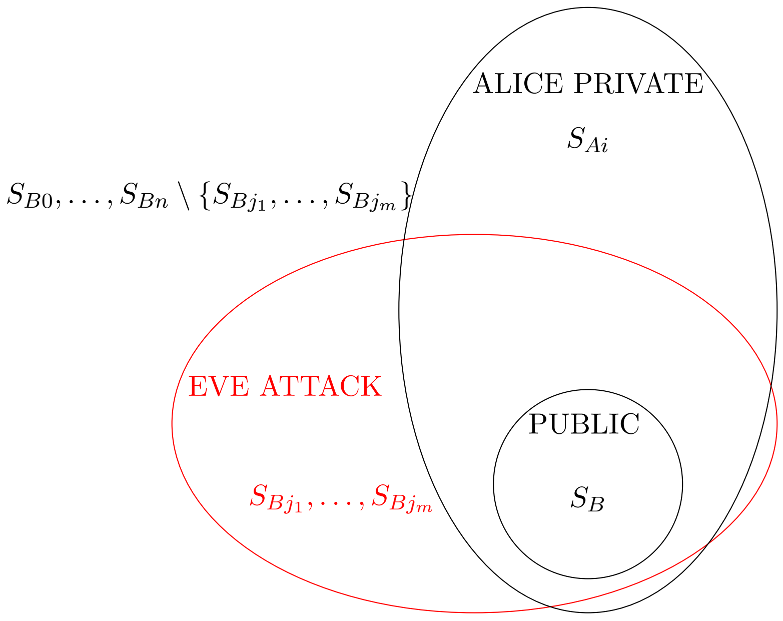
The details of encryption, decryption and the operations can be found in the appendix A1.

4.3.3 Choosing an appropriate quotient space

We already discussed that an appropriate choice of quotient space is needed to the make the calculations work. However, we have full control over what that quotient space is. Alice and Bob repeat the key exchange process again to generate the second private key q, independent of the first exchange and form the quotient R/J as in (4.19).
When looking at both the decryption function and the homomorphic property of our system, there is a strong reliance on the choice of quotient space R/J to make sure the calculations are performed correctly. The issue with this procedure is that we have made public, the product of our 2 secret keys in R. Unlike in the integers, factoring 2 polynomials is not a difficult problem. This means any eavesdropper that can observe the operations being performed can find both keys.
The solution to this problem is to put off correcting the extra multiplication error in the decryption. Since decryption is performed offline, we don't need to be concerned with our choice of quotient space revealing important information. The problem with this approach is that since our ciphertexts won't be corrected while being processed, their size will increase at a faster rate. This means that computation and storage requirements will increase quickly. Our choice of second key is independent of our first key. Therefore, we can publicly quotient out by a function of the second key without revealing any information about the first key. Although this still won't give us a perfect homomorphism, but it will reduce the rate at which the product of ciphertexts will grow.

4.3.4 Potential attack

Although Eve does not have access to the private ideal that Alice has generated, she could create her own ideal with a finite Gröbner basis that contains the public ideal. This would allow her to successfully perform the membership test on Bob's choice of a secret key. While this is a concern, Eve's choice of ideal may also accept polynomials other than that corresponding to Bob's choice; making it unclear which is the actual choice.
We will denote the set of polynomials that Bob sends back to Alice that fall into Eve's ideal as the set $\{S_{Bj_1}, S_{Bj_2}, \ldots, S_{Bj_m}\}$.


ALICE PRIVATE
S_{Ai}
$S_{B0}, \dots, S_{Bn} \setminus \{S_{Bj_1}, \dots, S_{Bj_m}\}$
EVE ATTACK
$S_{Bj_1}, \dots, S_{Bj_m}$
PUBLIC
S_B

Our goal to prevent an attack like this is to maximise the value of m.
To get an idea of the ideal that Eve may make, intersections of ideals should be understood. To begin with, elimination ideals need to be understood followed by the introduction of a new type of ordering.
Elimination theory studies the methods of eliminating variables from systems of polynomial equations. The process of finding the intersection of ideals involves introducing new variables in a specific way then eliminating all elements of the new ideal which include that variable [Cox *et al.* (1992)].

Definition. Given $I = \langle f_1, \dots, f_s \rangle \subseteq K[x_1, \dots, x_n]$, the l-th *elimination ideal* I_l is the ideal of $K[x_{l+1}, \dots, x_n]$ defined by

$$I_l = I \cap K[x_{l+1}, \dots, x_n]. \tag{4.31}$$

All of the orderings seen so far to assign are constructed by first assigning an order on the individual variables, say here $x_1 < x_2 < \cdots < x_n$. However, for the following definition, it is necessary to use 2 separate orderings, denoted $<$ and $<<$.

Definition. Let $OCC(M, V)$ be a function that takes a monomial $M \in \langle x_1, \dots, x_n \rangle$ and V, a subset of $\{x_1, \dots, x_n\}$ and outputs the number of times the variables in V occur in M. For example $OCC(M = x_1x_2x_4x_1 \in \langle x_1, \dots, x_4 \rangle, V = \{x_1, x_2\}) = 3$.
Suppose $\mathbf{a} = [\alpha_1, \dots, \alpha_n], \mathbf{b} = [\beta_1, \dots, \beta_n] \in \mathbb{Z}^n_+$. We will say $\mathbf{a} < \mathbf{b}$ if the first non zero element working from the right of the vector $\mathbf{b} - \mathbf{a}$ is positive.
Specify a subset $\{a_1, a_2, \dots, a_l\} \subseteq \{1, \dots, n-1\}$. If M and N are monomials in $x_1, \dots, x_n$ then $M < N$ with respect to the *multigraded lex* order if one of the following two conditions hold:

- $[OCC(M, V_1), \ldots, OCC(M, V_l)] < [OCC(N, V_1), \ldots, OCC(N, V_l)]$,
- $[OCC(M, V_1), \ldots, OCC(M, V_l)] = [OCC(N, V_1), \ldots, OCC(N, V_l)]$ and $M < N$ with respect to the deglex order $x_1 < \cdots < x_n$,

where

$V_j = \{x_j | a_{j-1} < j \leq a_j\}$ for $1 \leq j \leq l$, we also set $a_0 = 0$ and $a_{l+1} = n$.
We denote the relation between the variables with the following sequence

$$x_1 R_1 x_2 R_2 \cdots R_{n-1} x_n \tag{4.32}$$

where R_i is $<<$ if i is one of the a_j's and $<$ otherwise.

Example. Consider the multigraded lex order on the variables x_1, x_2, x_3 denoted $x_1 << x_2 < x_3$. The order on monomials in these variables up to degree 2 is

$$1 < x_1 < x_1x_1 < x_2 < x_3 < x_1x_2 < x_1x_3 < x_2x_1 < x_3x_1 < x_2x_3 < x_3x_2 < x_3x_3. \tag{4.33}$$

Definition. Let j and n be natural numbers where $1 \leq j \leq n$. A monomial order is of j^{th} *elimination type* if any monic monomial involving any of $x_1, x_2, \ldots x_j$ is greater than any monomial in $K[x_{j+1}, \ldots, x_n]$.

Theorem 4.6 (Drakos (1996))**.** *Let $>$ be a monomial order on the monic monomials of $K[x_1, \ldots, x_n]$, let $I \subset K[x_1 \ldots, x_n]$ and G a Gröbner basis of I with respect to $>$. If $1 \leq j \leq n$ and $>$ is of j-th elimination type, then $G \cap K[x_{j+1}, \ldots, x_n]$ is a Gröbner basis for $I \cap K[x_{j+1} \ldots, x_n]$.*

The theory up until now for commutative rings has been the same as the theory for noncommutative rings, the following theorem about intersections of ideals is the where the process begins to diverge [Buchberger & Winkler (1998)].

Theorem 4.7 (Cox *et al.* (1992))**.** *For 2 ideals $I = (f_1, \ldots, f_k), J = (g_1, \ldots, g_l) \in K\langle x_1, \ldots, x_n\rangle$ then $I \cap J = H \cap K\langle x_1, \ldots, x_n\rangle$ where*

$$H = (tf_i, (1-t)g_j, tx_m - x_m t | 1 \leq i \leq k, 1 \leq j \leq l, 1 \leq m \leq n) \in K\langle x_1, \ldots, x_n, t\rangle. \tag{4.34}$$

Proof. If $F \in I \cap J$, then $F = \sum_{i=1}^{k} p_{i_L} f_i p_{i_R} = \sum_{j=1}^{l} q_{j_L} g_j q_{j_R}$ for some polynomials $p_{i_L}, p_{i_R}, q_{j_L}, q_{j_R} \in K\langle x_1, \ldots, x_n\rangle$. In $K\langle x_1, \ldots, x_n, t\rangle$ the polynomial can be expressed as

$$F = tF + (1-t)F = \sum_{i=1}^{k} p_{i_L} t f_i p_{i_R} + \sum_{j=1} q_{j_L}(1-t) g_j q_{j_R} + \sum_{m=1}^{n} r_{m_L}(tx_m - x_m t) r_{m_R}, \tag{4.35}$$

where $r_{m_L}, r_{m_R} \in K\langle x_1, \ldots, x_n, t\rangle$. The final sum on the RHS is introduced as a consequence of moving t (and $1-t$) to the generators f_i (g_j). It must be the case therefore that

$F \in H \cap K\langle x_1, \ldots, x_n \rangle$.

To see why that last term is important consider the two ideals $I = \{x^2, xy^2\}$ and $J = \{xyx\}$. Let $F = x^2(yxy + yx) + (0)xy^2 = x^2yxy + x^2yx = (x)xyx(y+1)$. Introducing the t variable produces the 2 polynomials $tF_I = t \cdot x^2(yxy + yx) = tx^2yxy + tx^2yx$ and $(1-t)F_j = (x)(1-t)xyx(y+1) = x^2yxy + x^2yx - xtxyxy - xtxyx$, from which it is clear to see that the terms with a t variable within do not cancel, thus the need for the extra sum.

Conversely, assume $F \in H \cap K\langle x_1, \ldots, x_n \rangle$. $F \in H$ means that it is of the form

$$F = \sum_{i=1}^{k} p_{i_L} t f_i p_{i_R} + \sum_{j=1} q_{j_L}(1-t) g_j q_{j_R} + \sum_{m=1}^{n} r_{m_L}(tx_m - x_m t) r_{m_R}, \tag{4.36}$$

where $p_{i_L}, p_{i_R}, q_{j_L}, q_{j_R}, r_{m_L}, r_{m_R} \in K\langle x_1, \ldots, x_n, t \rangle$. Since $F \in K\langle x_1, \ldots, x_n \rangle$ also, F is indenpendant of t. $F = \sum p'_{i_L} f_i p'_{i_R} \in I$ when substituting 1 for t and $F \in J$ if $t = 0$ which implies $F \in I \cap J$ as required. □

Corollary 4.1. *Let G be a Gröbner basis for H according to the elimination order in $K\langle x_1, \ldots, x_n, t \rangle$ with $t \geq x_1, \ldots, x_n$. Then $G \cap K\langle x_1, \ldots, x_n \rangle$ is a Gröbner basis for $I \cap J$.*

These theorems lead to the algorithm for finding the intersection of ideals. Let $I = \langle f_1, \ldots, f_r \rangle$ and $J = \langle g_1, \ldots, g_s \rangle$ be ideals in $K\langle x_1, \ldots, x_n \rangle$. Take the ideal

$$\langle tI, (1-t)J, tx_1 - x_1 t, \ldots, tx_n - x_n t \rangle \subseteq K\langle x_1, \ldots, x_n \rangle, \tag{4.37}$$

then compute a Gröbner basis with respect to an appropriate ordering where t is greater than all the x_i. The elements of the basis that do not contain t will form a basis, notably a Gröbner basis, of $I \cap J$.

Example. Let $F_1 = \{x, xy\} \subset K\langle x, y \rangle$ and $F_2 = \{y, xy\} \subset K\langle x, y \rangle$, under the multi-graded lex ordering with $t >> x > y$. To find the intersection of those 2 ideals and its Gröbner basis, Mora's algorithm will be run on the basis elements

$$H = \{tx, txy, (1-t)y, (1-t)xy, tx - xt, ty - yt\}. \tag{4.38}$$

The s-polynomials from this initial basis are: $(tx)y - (txy - xy) = xy$, $(tx) - (tx - xt) = xt$, $(txy) - (txy - xy) = xy$, $(ty - y) - (ty - yt) = yt - y$, $(txy - xy) - (tx - xt)y = xty - xy$. The first polynomial in the list, xy, does not reduce with respect to H. The only new s-polynomial added is $t(xy) - (txy - xy) = xy$, which clearly does not need to be added as it will be reduced to 0 by the original xy. The remaining polynomials in the list do not reduce either and add no new s-polynomials to the list. Thus the Gröbner basis is

$$G_{K\langle t,x,y \rangle} = \{tx, txy, ty - y, txy - xy, tx - xt, ty - yt, xy, xt, yt - y, xty - xy\}. \tag{4.39}$$

The Gröber basis for the intersection of the two original ideals in $K\langle x, y\rangle$ is, therefore, $\{xy, yx\}$. This shouldn't be too hard to justify even without the algorithm. The reduced Gröbner bases of the original 2 ideals were $\{x\}$ and $\{y\}$. i.e. any polynomial that contained at least one x (y) in each term would have been an element of the first (second) ideal. The intersection, therefore, would have to be all of the polynomials that included at least one of each variable in every term.

Our assumption for a potential attacker is that they would try and create an ideal similar to Alice's. The attacker may not know Alice's ideal but they can certainly create an ideal which contains the public ideal. Recall that we chose 3 polynomials to transfer to Bob. We will now show that these choices come from distinct cosets of Alice's private ideal. In fact, they were each a member of different ideals, namely

- $\langle I, xz + x, z^4y - z^3y\rangle$,
- $\langle I, yz + y, xz^4 - yzxz\rangle$,
- $\langle I, zx + z, xz^4 - z^2yx, y^3x + yzy, z^3y - zyz\rangle$,

where I is the public ideal from the example. To make it easier to find polynomials that lie in one ideal but not the others we will find the reduced Gröbner basis. The reduced Gröbner bases of the 3 ideals are

- $\langle xy, yz, xz + x\rangle$,
- $\langle xy + y^2, zy + y, y^2 + zy, -yx + zy, yz + y, xz^4 - yzxz\rangle$,
- $\langle -zy^3 + (zy)^2, zy^2x + yzy, zyx + zy, z^2y^2 - z^2yz, z^5 - z^2yz, yz - zy, xz^2y - yz^2x, xzy + yz, zx + z, xz^4 - z^2yx, y^3x + yzy, z^3y - zyz\rangle$.

To begin with, we will start with generating an element from the 3rd ideal, this should be the simplest place to start as there are fewer leading monomials in the other two ideals to be concerned about and therefore easier to membership test. From the first ideal, we can see that the leading monomial cannot contain the substrings xy, yz, xz. From the second ideal, we see that the leading monomial cannot contain the substrings zy, y^2, yx, xz^4. The first thing we notice is that there is no possible combination of letters containing a y allowed. Also, due to the xz monomial, the only polynomials left that we can form out of the generators of the 3rd ideal have a leading monomial of the form $z^m x^n$, for $m, n \geq 1$. We could spend time working out all the possible lead monomials for the polynomial in ideal 2, however for sake of example we note that this is the only ideal that contains a lead monomial with only $y's$.
As for ideal 1, once again, any monomial containing y will also be divisible by the lead monomial of at least one polynomial in ideal 2. This eliminates all but one of our generators from ideal 1. Our polynomial used here, therefore, will be of the form $l(xz + x)r$, ensuring

that l and r are not chosen in such a way to contain a substring of the lead monomials from the other ideals.
Now that we understand how Alice chose her ϵ's, we would like to ensure that these choices of ideals were suitable by examining their intersection. Our starting point for creating these ideals was to use the public ideal I, so our goal is to check that each pairwise intersection is a larger ideal than I.
Now that we understand how Alice chose her ϵ's, we would like to ensure that these choices of ideals were suitable by examining their intersection. Our starting point for creating these ideals was to use the public ideal I, so our goal is to check that each pairwise intersection is a larger ideal than I.
Even for seemingly small ideals finding a Gröbner basis for the intersections takes a long time. We don't need to find the full basis, however, just examples of elements that aren't in the public ideal. In the intersection of the first 2 ideals we have the polynomial

$$xzy + xy = (xz + x)y = x(zy + y). \tag{4.40}$$

The middle representation shows it as a sum of ideal 1s basis, whereas the right representation shows it as a sum of ideal 2s basis. It cannot be a member of our public ideal as the second term doesn't contain a z. Every term in the public ideal does contain a z so it would be impossible for it to be a member.
Now within the intersection of ideal 1 and ideal 3 we have the polynomial

$$z^4y - z^3y = z(z^3y - zyz) + z^2(yz - zy). \tag{4.41}$$

The left side of the equation is a generator of ideal 1 and the right-hand side is expressed as a sum of generators from ideal 3. [Rai (2004)] discusses how the public ideal could not contain polynomials whose leading term of the form z^my, therefore our polynomial could not be a member.
As you can see, the public ideal is included (not equal) to the intersection of Alice's ideal and each of her decoys. This means that there is a chance that an Eavesdropper may mistakenly believe one of Alice's coset polynomials is a member of the public ideal.

4.3.5 Practical concerns

Questions that may need to be addressed if there was to be a practical implementation of the key exchange protocol are:

- There is an important requirement that the polynomials sent back and forth maintain their order. What can be done to ensure that Alice and Bob are confident that they have received the polynomials in the intended order?
- Depending on how the ordering is recorded, even if only a subset of polynomials fails

to be transferred, all may have to be resent.

4.4 Improving the protocol in a Hecke algebra setting

In this section, we will look to be more specific about the setting we wish to use for our scheme. We will look at how we can use Hecke algebras in cryptosystems that use Gröbner bases. In particular, we will see how using a Hecke algebra can greatly improve the way in which we generate our private keys.

4.4.1 Infinite Gröbner bases with Hecke Algebra relations

The work on algorithms in various algebraic settings was studied in depth by Shirshov [Shirshov (1999)]. Due to his contribution, in the settings he worked in, we use the name *Gröbner-Shirshov basis.* Work in Hecke algebras follows this convention. Here we form the Gröbner-Shirshov basis for Hecke algebras where we adopt the notation

$$T_{i,j} = T_i T_{i-1} \cdots T_j \text{ for } i \geq j. \tag{4.42}$$

Just as we did with our standard polynomial space, we can create a Gröbner-Shirshov basis from a public Hecke algebra to easily solve the membership problem privately.

Proposition 4.1 (Kang *et al.* (2002))**.**

The following relations form a Gröbner-Shirshov basis for $\mathcal{H}_q(W)$ with respect to the monomial order DegLex:

$$\mathcal{R}_{q,DegLex} = \begin{cases} T_iT_j - T_jT_i, & \textit{for } i > j+1, \\ T_i^2 - (q-1)T_i - q, & \textit{for } 1 \leq i \leq n-1, \\ T_{i+1,j}T_{i+1} - T_iT_{i+1,j}, & \textit{for } i \geq j. \end{cases} \tag{4.43}$$

If we are interested in using Hecke algebras for the protocol, then we need to know that ideals can be formed that have an infinite Gröbner basis. In the following proposition we introduce a new ideal that also does not have a finite Gröbner basis. We do this by adapting the ideal from Theorem 4.5 to create a new ideal by adding relations from a Hecke algebra as generators.

Proposition 4.2. *Let K be a finite field and $K\langle x, y, z\rangle$ the noncommutative free algebra over K. The ideal I does not have a finite Gröbner basis under the DegLex order where*

$$I = \langle xzy + yz, yzx + zy, xz - zx, yzy - zyz, xyzx - yxyz, xyx - yxy\rangle. \tag{4.44}$$

Proof. Firstly, consider the poylnomials $g_0 = xzy+yz$ and $g_1 = xz-zx$. Now $O(g_0, g_1, r_2 = y) = yz+zxy$. Since we using DegLex ordering here, $zxy > yz$ so we denote $g_2 = zxy+yz$.

Next we consider the polynomial $O(g_2, g_1, r_2 = xy) = xyz + zx^2y$. Once again under DegLex we have $g_3 = zx^2y + xyz$.
We now begin to see a pattern emerge when we calculate $O(g_3, g_1, l_1 = x, r_2 = x^2y) = x^2yz + zx^3y$, which under DegLex gives us $g_4 = zx^3y + x^2yz$.
Continuing inductively, we get the infinite sequence:

$$g_n = O(g_{n-1}, g_1, l_1 = x, r_2 = x^{n-1}y) = zx^{n+1}y + x^nyz, \text{ for } n \geq 2. \tag{4.45}$$

If I had a finite Gröbner basis then the tip of some element of the basis would have to divide infinitely many $zx^{n+1}y$. The tip of this element would therefore have to be one of

- x^my for some $m \geq 0$ or
- zx^m for some $m \geq 0$.

Firstly, let's assume for a fixed m, that $x^my \in Tip(I)$. This means there must exist an $F = l_1(xzy + yz)r_1 + l_2(yzx + zy)r_2 + \cdots + l_6(xyx - yxy)r_6 \in I$ such that $tip(F) = x^my$.
This turns out to be impossible though as none of the terms in our basis are a subword of x^my for any $m \geq 0$.
We now assume for a fixed m, that $zx^m \in Tip(I)$. Now the only way zx^m can appear in a term in F is if $F = \cdots + (xz - zx)x^{m-1} + \ldots$. However, this means the term xzx^{m-1} also appears. Now $xzx^{m-1} > zx^m$ so it needs to be subtracted off.
To have xzx^{m-1} appear in some other way, the polynomial $x(xz - zx)x^{m-2}$ must appear.
Now the term $x^2zx^{m-2} > xzx^{m-1}$ must be subtracted off.
We continue this process inductively until we reach $x^{m-1}(xz - zx)$, where the term x^mz needs to be subtracted off. There is no other way to form that term with the generators in I, therefore, $zx^m \notin Tip(I)$.
Since neither of the monomials can appear in $Tip(I)$ there must be infinitely many terms in our Gröbner basis under this ordering. □

We have seen how the braid relations can be added to a basis with an infinite Gröbner basis. So far we have only added these relations to the ideal $\langle xzy + yz, yzx + zy\rangle$ however, using only one ideal for all encryption would give attackers plenty of time to try break that particular example. Theorem 4.5 has a following corollary that provides an infinite set of ideal options.

Corollary 4.2 (Rai (2004)). *Let K be a finite field, and let $K\langle x_1, x_2, \ldots, x_n\rangle$ be the noncommutative free algebra in n variables with $n \geq 5$. Let*

$$A = \prod_{i=1}^{n} x_i, \; B = x_1(\prod_{i=2}^{n-1} \rho(x_i))x_n \text{ and } C = x_1(\prod_{i=2}^{n-1} \sigma(x_i))x_n, \tag{4.46}$$

where ρ and σ are nontrivial permutations of $\{x_2, x_3, \dots, x_{n-1}\}$. Let $g_1 = ACB + BC$, $g_2 = BCA + CB$. Then $I = \langle g_1, g_2 \rangle$ does not have a finite Gröbner basis under and admissible order.

4.4.2 Invertible elements of Hecke algebras

Recall the description of the key exchange that we needed to choose a specific quotient space to work in. The reason for this was so that the 2 polynomial keys exchanged were the inverses of one another. One of the main advantages of using Hecke algebras is that individual monomials have an inverse defined that is not dependent on the choice of ideal. This means that there is no need to worry about the tradeoff between security and ciphertext growth. The inverse of the generator T_s is

$$T_s^{-1} = q^{-1}T_s - 1 + q^{-1}. \tag{4.47}$$

We can check this quickly by multiplying this on the left by T_s

$$\begin{aligned} T_s(q^{-1}T_s - 1 + q^{-1}) &= q^{-1}T_s^2 - T_s + q^{-1}T_s \\ &= q^{-1}((q-1)T_s + q) - T_s + q^{-1}T_s \\ &= q^{-1}(qT_s - T_s + q) - T_s + q^{-1}T_s \\ &= T_s - q^{-1}T_s + 1 - T_s + q^{-1}T_s = 1. \end{aligned} \tag{4.48}$$

Multiplying by T_s on the right looks very similar.

Theorem 4.8 (Kazhdan & Lusztig (1979))**.** *Let $u, v \in W$. There is a family $R_{u,v}(q)$ of polynomials such that*

$$(T_{v^{-1}})^{-1} = q^{-l(v)} \sum_{u \leq v} (-1)^{l(v)-l(u)} R_{u,v}(q) T_u, \tag{4.49}$$

where $l(w)$ is the length of the reduced word w.

This family is known as *R-polynomials*. Note that $R_{u,u}(q) = 1$. R-polynomials can be computed inductively using the following theorem.

Theorem 4.9 (Kazhdan & Lusztig (1979))**.** *Let $s \in S$ be such that $vs < v$. Then*

$$R_{u,v}(q) = R_{us,vs}(q) \tag{4.50}$$

if $us < u$, and

$$R_{u,v}(q) = qR_{us,vs}(q) + (q-1)R_{u,vs}(q) \tag{4.51}$$

otherwise.

Example. Suppose we have a Coxeter system with presentation $< s_1, s_2 | s_1^2 = 1, s_2^2 = 1, (s_1s_2)^{m_{1,2}} = 1 >$. Find the inverse of $T_{s_1s_2}$.
According to the formula for the inverse we have

$$\begin{aligned}(T_{(s_2s_1)^{-1}})^{-1} &= q^{-2}\sum_{u\leq s_2s_1}(-1)^{2-l(u)}R_{u,s_2s_1}(q)T_u \\ &= q^{-2}(R_{1,s_2s_1}T_1 - R_{s_1,s_2s_1}T_{s_1} - R_{s_2,s_2s_1}T_{s_2} + R_{s_2s_1,s_2s_1}T_{s_2s_1}).\end{aligned} \tag{4.52}$$

Even for a seemingly simple choice of T, some work has to go into finding the R polynomials.

$$R_{1,s_2s_1} = qR_{s_1,s_2} + (q-1)R_{1,s_2} = (q-1)(qR_{s_2,1} + (q-1)R_{1,1}) = q^2 - 2q + 1. \tag{4.53}$$

$$R_{s_1,s_2s_1} = R_{1,s_2} = qR_{s_2,1} + (q-1)R_{1,1} = q - 1. \tag{4.54}$$

$$R_{s_2,s_2s_1} = qR_{s_2s_1,s_2} + (q-1)R_{s_2,s_2} = (q-1)R_{1,1} = q - 1. \tag{4.55}$$

$$R_{s_2s_1,s_2s_1} = 1. \tag{4.56}$$

Substituting these values back into our formula we have

$$\begin{aligned}&q^{-2}((q^2-2q+1)T_1 - (q-1)(T_{s_1} + T_{s_2}) + T_{s_2s_1}) \\ &= T_1 - 2q^{-1}T_1 + q^{-2}T_1 - q^{-1}T_{s_1} + q^{-2}T_{s_1} - q^{-1}T_{s_2} + q^{-2}T_{s_2} + q^{-2}T_{s_2s_1}.\end{aligned} \tag{4.57}$$

We check this is the correct inverse by multiplying on the right of $T_{s_1s_2}$ and state that the same property holds if multiplied on the left.

$$T_{s_1s_2}(T_1 - 2q^{-1}T_1 + q^{-2}T_1 - q^{-1}T_{s_1} + q^{-2}T_{s_1} - q^{-1}T_{s_2} + q^{-2}T_{s_2} + q^{-2}T_{s_2s_1}) \tag{4.58}$$

Applying the relations $T_iT_j = T_{ij}$ and $T_{s_2}^2 = (q-1)T_{s_2} + q$, the above expands to

$$\begin{aligned}&T_{s_1s_2} - 2q^{-1}T_{s_1s_2} + q^{-2}T_{s_1s_2} - q^{-1}T_{s_1s_2s_1} + q^{-2}T_{s_1s_2s_1} - T_{s_1s_2} + q^{-1}T_{s_1s_2} \\ &- T_{s_1} + q^{-1}T_{s_1s_2} - q^{-2}T_{s_1s_2} + q^{-1}T_{s_1} + q^{-1}T_{s_1s_2s_1} - q^{-2}T_{s_1s_2s_1} + T_{s_1} - q^{-1}T_{s_1} + 1 \\ &= 1.\end{aligned} \tag{4.59}$$

We have established that individual monomials in Hecke algebras have an inverse that can be calculated. When performing our key exchange, however, Alice and Bob are sending polynomials. Finding an inverse for those is a greater challenge. It would be advantageous to use just a single monomial instead of a polynomial in terms of the cost of moving and storing this information.
The beauty of using the Hecke algebra is that the braid relations involved should help to fortify against other attacks. Recent papers on braid group problems still suggest that it is

an open problem to solve the conjugacy problem for braid in polynomial time [Schleimer & Wiest (2019)]. This suggests that this type of problem may have cryptographic use. Our ciphertext is now equal to our plaintext multiplied on the left by a monomial and on the right by a polynomial. We can think of this multiplying on the left, however, as multiple ciphertexts, each of which has been multiplied on the right by an individual monomial. This, in essence, is an example of the multiple conjugacy problem. That means it should be hard to find p or q given knowledge of a plaintext and corresponding ciphertext. Such a property means that this revised version of the scheme is resistant to known (and chosen) plaintext attacks, assuming this instance of the multiple conjugacy problem is still hard.
Because of the way the secret key is chosen, we can't just send monomials back and forth in the key exchange. We must, therefore, construct a method for Alice and Bob to decide on a monomial based on the polynomials being sent. We achieve this with the following method.
Monomial selection process

1. Alice sends Bob her polynomials $S_{A,0}, S_{A,1}, \dots S_{A,n-1}$, where $S_{A,i}$ is the polynomial from her private ideal. Bob sends back to Alice the polynomials $S_{B,0}, S_{B,1}, \dots S_{B,n-1}$ where $S_{B,j}$ is the polynomial they agree on.

2. Because Alice and Bob maintain the order the polynomials were sent, Alice knows the positions of both her original choice and the choice Bob made, positions i and j respectively.

3. Alice can then calculate the difference between the locations $j - i \bmod(n)$ and sends that value to Bob (Eve doesn't know i or j so the difference means nothing).

4. Upon receiving the value from Alice, Bob can calculate the position of Alice's original polynomial $j - (j - i) \bmod(n) = i$. Alice and Bob both know that position and can use that value to select the monomial at the i^{th} position from the secret polynomial.

The first thing to note about this method is that, if the number of polynomials sent is not equal to the number of terms in each polynomial, there will be some redundancy in the method. For example, if 5 polynomials are sent and some have 6 or more terms. The 6th terms and onwards couldn't possibly be chosen because we are assuming the position value is in $\mathbb{Z}_5$. We could return to the idea of performing the exchange multiple times if we wish to generate more combinations of position values.
The only new value being made public is the value Alice sends back to Bob. This represented the difference in location between Alice's and Bob's choice in polynomial. Because we believe an eavesdropper has no idea the location of either polynomial, the new value Alice sent will not provide information about the location of either polynomial.

4.4.3 Updating the key exchange protocol

1. Alice and Bob agree on the number of polynomials they wish to communicate and how many times they need to perform the exchange.
2. Alice and Bob agree publicly on generators, $x_1, \dots, x_n$ for an ideal. This ideal includes braid relations and has the property that Mora's algorithm will not terminate if used with this basis.
3. Alice selects a subset of $[1, n]$, say $[a, b]$ and forms a Gröbner basis of the form in proposition 4.1, along with powers of the variables $x_1, \dots, x_{a-1}, x_{b+1}, \dots, x_n$.
4. Bob as usual chooses which polynomial will be the secret key.
5. Alice calculates Bobs choice.
6. Alice and Bob perform the monomial selection process.
7. Alice and Bob both calculate the inverse of the chosen monomial.

Example. Alice and Bob agree to perform one key exchange that involves four polynomials. They also publicly agree to use the generators x, y and z where $x > y > z$. Their public, I, will be the one from proposition 4.2.
Alice chooses the variables y and z to use for the Hecke algebra relations and constructs her private ideal

$$A_I = \langle I, y^2 - (q-1)y - q, z^2 - (q-1)z - q, x^3 + x^2 \rangle. \tag{4.60}$$

She sends the polynomials $S_{A_0}, S_{A_1}, S_{A_2}, S_{A_3}$ to Bob, where S_{A_1} is the polynomial from her ideal and

$$\begin{aligned} S_{A_0} &= xzy + yz - y(yx^2 + zx^2)z = -y^2x^2z - yzx^2z + xzy + yz \\ S_{A_1} &= xyzx - yxyz + x^3 + x^2 \\ S_{A_2} &= yzx + zy + (xz - z^2)x = xzx + yzx - z^2x + zy \\ S_{A_3} &= yzy - zyz + (zyx + x^2)z = zyxz + x^2z + yzy - zyz. \end{aligned} \tag{4.61}$$

Bob chooses the secret polynomial $S_B = (yzy - zyz) + (yzx + zy)xz + x(xz - zx) =$. He finds the difference between S_B and S_{A_3}

$$S_{B_{diff}} = yzx^2z - xzx. \tag{4.62}$$

He therefore sends back

$$\begin{aligned}
S_{B_0} &= -y^2x^2z - yzx^2z + xzy + yz + yzx^2z - xzx = -y^2x^2z - xzx + xzy + yz \\
S_{B_1} &= xyzx - yxyz + x^3 + x^2 + yzx^2z - xzx = yzx^2z + xyzx - yxyz + x^3 - xzx + x^2 \\
S_{B_2} &= xzx + yzx - z^2x + zy + yzx^2z - xzx = yzx^2z + yzx - z^2x + zy \\
S_{B_3} &= (yzy - zyz) + (yzx + zy)xz + x(xz - zx) = yzxxz + zyxz + x^2z - xzx + yzy - zyz.
\end{aligned} \tag{4.63}$$

Alice proceeds to work out that Bob has chosen S_{B_3}. She now knows that her polynomial was at position 1 and that Bob's polynomial is at position 3. She sends the difference mod 4 back to Bob. Now Alice and Bob both know to use the 2^{nd} monomial in S_{B_3}, $zyxz$, as the private key. Now they have a monomial in a Hecke algebra they can calculate the second secret key as the inverse of $zyxz$.
In step 3 Alice chooses a subset of the generators so that it is more difficult for an attacker to guess her private basis. She needs to ensure that there is no gap in her choice of generators for the relations. Using the variables $x_a, x_{a+1}, \dots, x_{c-1}, x_{c+1}, \dots, x_b$, where $a < c < b$, would not be an acceptable choice. This would exclude some of the relations that are part of the Hecke algebra Gröbner basis, meaning Alice could not form the relevant Gröbner-Shirshov basis. When choosing her polynomials to send, Alice, therefore, needs to make sure that her polynomials all look like they would come from a similar structure.

Chapter 5

Understanding the complexity of protocols

In this chapter, we will study the complexity of the problem we based our cryptosystem on in the previous chapter. Section 1 will introduce the different measures of complexity in order to establish why we choose in particular to use generic complexity.
Section 2 and onwards contains original material with this section providing a general theorem about finding the generic complexity of Mora's algorithm on different ideals.
Section 3 and 4 will take the theorem given in the previous section and apply it in more detail to our example.

5.1 Measures of complexity

In this section, we will study different measures of complexity, started with worst case complexity and build up to what we believe to be the best representation of complexity for cryptographic problems, generic complexity. We will use the quicksort algorithm [Hoare (1961)] as an example throughout to help understand the measures better.

5.1.1 Calculating bounds

When understanding how efficient an algorithm is, knowing exactly how many basic operations isn't as important as knowing some kind of bound on the number of operations required. This bound is a function of the length of the input to some algorithm [Arora & Barak (2009)].

Definition. Given 2 functions, f, g, from $\mathbb{Z}$ to $\mathbb{Z}$, define

- $f = O(g)$ if there exists a constant c such that $f(n) \leq c \cdot g(n)$ for large enough n,
- $f = \Omega(g)$, if $g = O(f)$,

- $f = \Theta(g)$, if both $f = O(g)$ and $g = O(f)$,
- $f = o(g)$ if for every $\epsilon > 0$, $f(n) \leq \epsilon \cdot g(n)$ for large enough n,
- $f = \omega(g)$ if $g = o(f)$.

Because we are dealing with algorithms that may not terminate, we adopt the following definition.

Definition. An algorithm is referred to as *partially correct* if it is possible that it won't terminate but, if it does terminate, the output will be correct.

5.1.2 Worst case complexity

The *worst case complexity* of an algorithm refers to the longest running time (space) needed for any input of size n. Knowing this bound guarantees that any possible input of the same size will take less than or the same amount of time (space).
Later on in this chapter, a sorting algorithm will be used together with Mora's algorithm. Here we give a concrete example, the quicksort algorithm that would be used in this way. The quicksort uses a recursive divide and conquers approach to sort an array.

Algorithm 6 Quicksort

```
1: procedure QUICKSORT($A, p, r$), $A$ is an array that needs sorting, $p$ is the initial
   index, $r$ is the final index
2:     if $p < r$ then
3:         $q$ = PARTITION($A, p, r$)
4:         QUICKSORT($A, p, q - 1$)
5:         QUICKSORT($A, q + 1, r$)
6:     return $A$ sorted in ascending order
```

The partition algorithm called during the quicksort algorithm has the job of splitting the array into 2 sub-arrays. The algorithm uses the last element of the array as a pivot and partially sorts the whole array so that all values less than the pivot value now appear before all values greater than the pivot [Cormen *et al.* (2009)].

Algorithm 7 Partition

```
procedure PARTITION($(A, p, r)$), $A$ is an array that needs sorting, $p$ is the initial index, $r$ is the final index
    $x = A[r]$
    $i = p - 1$
    for $j = p$ to $r - 1$
        if $A[j] \leq x$
            i = i + 1
            Swap $A[i]$ with $A[j]$
        end if
    end for
    Swap $A[i+1]$ with $A[r]$
    return $i + 1$
```

Theorem 5.1. *The worst case complexity of the quicksort is $O(n^2)$.*

Proof. To prove this theorem, consider the case where the array is already sorted (a similar argument also applies if the order is completely reversed). As the algorithm works through the loops of an array of length n, at the i^{th} iteration the array will be split into 2 sub arrays, one of length 0 and the other of length $n - i$. If $T(n)$ is the time it takes to complete the entire quicksort and the partition algorithm is $O(n)$ then this gives the recurrence relation

$$\begin{aligned} T(n) &= T(n-1) + T(0) + O(n) \\ &= T(n-1) + O(n), \end{aligned} \tag{5.1}$$

where $T(0) = O(1)$. Summing through this recursion gives an arithmetic series which evaluates to $O(n^2)$. □

As an example, figure 5.1 shows the iterations of running the quick sort array on an already sorted array of length 5.

5.1.3 Average case complexity

Worst case complexity is a common way to measure the complexity of a computation problem. The problem with measuring this way is that these worst cases can be few and far between and therefore are not representative inputs. *Average case complexity* considers the distribution of possible inputs where we calculate the expected number of operations

$$E(t_n) = \sum_{I \in B_n} t_n(I) Pr(I), \tag{5.2}$$

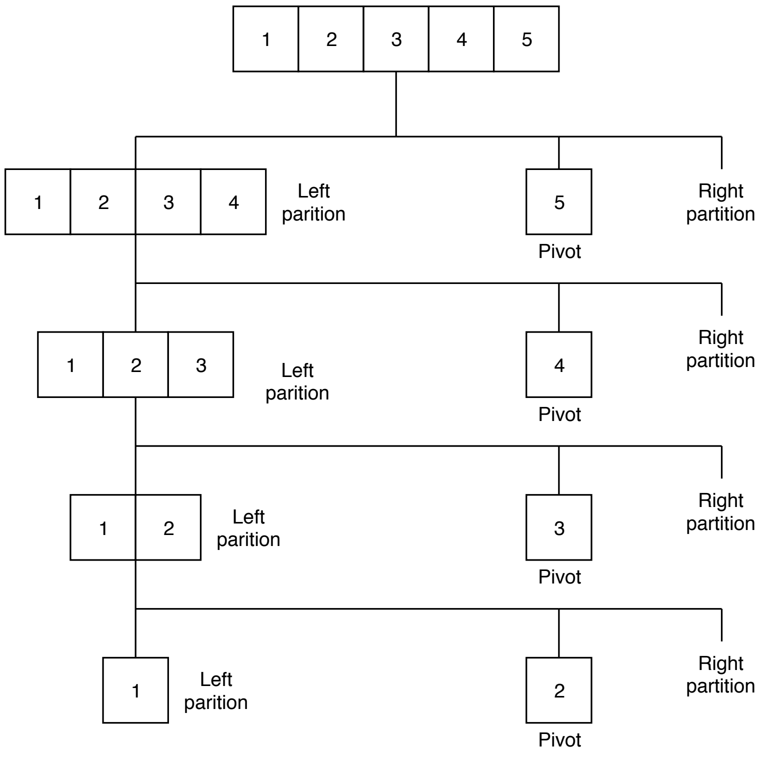


Figure 5.1: Worst case scenario for quick sort

where the probability Pr is typically uniform and B_n represents the collection of all inputs of length n. $t_n(I)$ is the time taken to complete the algorithm on input I.

Theorem 5.2. *The average case complexity of the quicksort is $O(n\ log(n))$.*

Proof. To prove this theorem, the algorithm is updated slightly to randomly swap an element of the array with the final element before assigning the pivot. The difference in complexity arises from the number of times the comparison operation is performed. Define the set

$$Z_{i,j} = \{z_i, z_{i+1}, \ldots, z_j\}. \tag{5.3}$$

The first important observation is that every pair of elements in this set are compared at most once. To understand why consider elements z_k and $z_{k'}$, where $i \leq k < j$. For any pair of elements in $Z_{i,j}$, one of three things will happen

- z_k will be compared with z_j as it is the pivot.
- If $z_k < z_j < z_{k'}$ then z_k and $z_{k'}$ will be separated into different partitions and therefore never compared.
- If $z_k, z_{k'} < z_j$ or $z_k, z_{k'} > z_j$ then they will both placed in the same partition and once again we will have to evaluate which of the 3 cases listed here they are in.

The pivot element isn't ever included in the following recursive calls so there will be no more opportunities to compare anything to z_j.
Because each comparison occurs either once or not at all, it is natural to introduce the variable $X_{i,j}$, the definition of which is

$$X_{i,j} = \begin{cases} 1, & \text{if } z_i \text{ and } z_j \text{ were compared at some point throughout the quicksort,} \\ 0, & \text{otherwise.} \end{cases}$$

With this in mind it is possible to calculate the total number of comparisons performed by the algorithm.

$$\begin{aligned} X &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j} \Rightarrow \\ E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{i,j}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr(z_i \text{ is compared with } z_j) \end{aligned} \tag{5.4}$$

Suppose now that there is an array $\{z_i, \ldots, z_k, \ldots, z_j\}$ (put into a random order) where z_k is chosen to be the pivot value. z_i and z_j will never be compared since $z_i < z_k$ and will, therefore, be put in the lower list, while $z_j > z_k$ and will subsequently be put in the upper list. This means that the probability that z_i is compared with z_j in $Z_{i,j}$ is the probability that one of them is chosen as the pivot.

$$\begin{aligned} Pr(z_i \text{ is compared with } z_j) &= Pr(z_i \text{ or } z_j \text{ is the first chosen pivot from } Z_{i,j}) \\ &= 2Pr(z_i \text{ is the first pivot chosen from } Z_{i,j}) \\ &= \frac{2}{j-i+1} \end{aligned} \tag{5.5}$$

This follows because the 2 events are mutually exclusive with the same probability. Substituting this value into the previous results gives the following result for time complexity.

$$\begin{aligned} E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\ &< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\ &= \sum_{i=1}^{n-1} O(lg\ n) \{\text{Harmonic series}\} \\ &= O(n\ lg\ n). \end{aligned} \tag{5.6}$$

□

While the average-case complexity does a better job of considering how difficult a problem is for any input the, potentially very limited, number of difficult instances do influence the calculation. This is shown in the next example, where for each n there is only a single hard instance of the problem, the other instances are linear yet our definition of average-case tells us that the problem has exponential average-case complexity.

Example. (Taken from Wikipedia: Generic-case complexity) Let ω be a string of binary digits. Let $I_n = \{0,1\}^n$ be the set of all binary strings of length n and suppose one is selected at random from a uniform distribution. Suppose we have an algorithm such that the running time of the algorithm for a string of length n is

$$t(\omega) = \begin{cases} 2^{2^n}, & \text{if the string is all 1's} \\ n, & \text{for all other strings in } I_n. \end{cases} \tag{5.7}$$

The average complexity is calculated as follows

$$\begin{aligned} \sum_{\omega \in I_n} t_n(\omega) Pr(\omega) &= (2^{2^n} \cdot \frac{1}{2^n}) + (n \cdot \frac{2^n - 1}{2^n}) \\ &= 2^n + n - \frac{n}{2^n} \\ &= O(2^n). \end{aligned} \tag{5.8}$$

5.1.4 Generic complexity

The average case complexity makes an effort to take into account all possible inputs. As seen in the previous subsection, the issue with taking an average is that the calculation can be heavily influenced by a small set of values that skew very high in complexity. *Generic* *complexity* is used to capture the complexity of the problem when it comes to most of the possible inputs, ignoring a small unrepresentative set of inputs [Kapovich *et al.* (2003)].

Definition. Let I be a countably infinite set. We also let $|.| : I \to \mathbb{Z}_{\geq 0}$ represent a size function. The ball of radius n is defined as $B_n = \{x \in I | |x| \leq n\}$.
For a subset $S \subseteq I$ the *lower asymptotic density* $\underline{\rho}_I(S)$ of S in I as

$$\underline{\rho}_I(S) = \underset{n\to\infty}{limsup} \frac{\#(B_I(n) \cap S)}{\#(B_I(n))}. \tag{5.9}$$

We can drop the supremum requirement of the definition if the actual limit exists, in which case we denote the limit $\rho_I(S) = \underset{n\to\infty}{lim} \frac{\#(B_I(n)\cap S)}{\#(B_I(n))}$. We call a subset S *generic* if $\rho_I(S) = 1$.

We will use $t_{\mathcal{U}}(w)$ to represent the time it takes for an algorithm $\mathcal{U}$ to run on input w. $T_{\mathcal{U}}(w) \in \mathbb{N}$ if the algorithm terminates with a desired output and ∞ otherwise.

Definition. Let $\mathcal{U}$ be a partial deterministic (Same input will always give the same output) algorithm that takes an input from I and outputs an element from a countable set U.
An algorithm $\mathcal{U}$ has *generic-case time complexity* less than a monotone non decreasing function, $f(n) \geq 0$, if there exists a generic subset $S \subseteq I$, such that for every $x \in I$ we have $t_{\mathcal{U}}(x) \leq f(|x|)$.

Example. Returning to the example given in the average case complexity discussion, define the set $X = \{\text{Strings of } 1's \text{ and } 0's \text{ with at least one } 0\} \subset I_n$. The running time for this set of inputs was defined to be $O(n)$. Inputting these sets into the asymptotic density formula gives

$$\underset{n\to\infty}{limsup} \frac{\#(I_n \cap X)}{\#(I_n)} = \frac{2^n - 1}{2^n} = 1. \tag{5.10}$$

Therefore, despite the average-case complexity of this problem is exponential, the generic complexity is $O(n)$.

5.2 Outline of the generic complexity proof

The remainder of the work in this chapter is the authors own. In this part, we will look at our approach to finding the generic complexity of ideal membership testing in infinite Gröbner bases. The following steps will cover the general idea for this type of problem. Afterwards, we will look in more detail each step for the infinite Gröbner basis that has been used in the previous section.
As part of the generic complexity calculation, we need to find the total number of instances of the problem. First, the total number of degree n polynomials in the ideal needs to be found. To do this its necessary to check each degree n polynomial to identify whether or not it is in the ideal. We begin by looking at the general form for a polynomial in an ideal

$I =< g_1, \dots, g_t >$, namely

$$l_1 g_1 r_1 + \cdots + l_t g_t r_t, \tag{5.11}$$

where $l_i, r_i \in R$.
In order to have a degree n polynomial it must be the case that

$$max_{1 \leq i \leq t}(deg(l_i) + deg(g_i) + deg(r_i)) = n. \tag{5.12}$$

We can find the number of potential l's and r's, however, some degree n terms will cancel, resulting in a polynomial of a lower degree. Therefore, it is necessary to see which leading monomials of ideal elements can be formed in more than one way from multiple generators. Now we are interested in which of these polynomials in the ideal (if any) require a long time to be recognised as being whether they are in the ideal or not by an adversary's algorithm i.e. the hard instances. In order to look into this question, we start with a new definition.

Definition. Let $I = k\langle x_1, \dots, x_t \rangle$ be an ideal in a finite number of non-commuting variables $x_1, \dots, x_t$. We define $p_{i,1}$ and $p_{i,2}$ to be polynomials with the properties

$$LM(p_{i,1}) = x_i W_{i,1} \tag{5.13}$$

and

$$LM(p_{i,2}) = W_{i,2} x_i \tag{5.14}$$

for $i \in \{1, \dots, t\}$. Here $W_{i,j}$ are monomials in $\{x_1, \dots, x_t\}$. The set $\{p_{i,j} \in I, 1 \leq i \leq t, 1 \leq j \leq 2\}$ is called a *complete overlapping subset* of the ideal I.

All $p_{i,j}$ need not be unique. For example $p = x_1 x_2 + x_3$ under degLex ordering could satisfy the requirements for both $p_{1,1}$ and $p_{2,2}$. We will now discuss the theorem that is the key property of our infinite Gröbner bases that suggests the membership problem is difficult.

Theorem 5.3. *Let R be a polynomial ring in a finite number of non-commuting variables under deglex ordering, and let I be a finitely generated ideal of R. Suppose after a finite number of iterations of Mora's algorithm on I that every polynomial tested is added to the Gröbner basis and we can construct a complete overlapping subset. Then the number of S-polynomials generated grows exponentially in the degree of the polynomials.*

Proof. The existence of a complete overlapping subset in the ideal means that every new polynomial tested will overlap with at least 2 polynomials.
Consider all the possible degree n polynomials. If the lead monomial of a polynomial is of degree n, then the largest possible degree of the second terms is also n. The same reasoning tells us that the largest possible degree of the second terms in the complete

overlapping subset polynomials is

$$d_{p_{i,j}} = max_{i,j}\{\text{degree of } p_{i,j}\}. \tag{5.15}$$

The maximum degree of an S-polynomial formed from a degree n polynomial and the complete overlapping subset is therefore

$$n + d_{p_{i,j}} - 1. \tag{5.16}$$

Let $P_n = \{$ Polynomials of degree at most $n\}$. Then $p_n = |P_n|$ is the total number of polynomials to be tested of degree n or less, then the linear increase of $d_{p_{i,j}} - 1$ in the degree of results in

$$p_{n+d_{p_{i,j}}-1} \geq 2p_n. \tag{5.17}$$

Continuing inductively we must have

$$p_{n+k(d_{p_{i,j}}-1)} \geq 2^k p_n. \tag{5.18}$$

□

We would like to know how many terms are in each polynomial in the ever-growing basis. This will allow us to have an understanding of how many polynomials are difficult to classify. Let $P_n^{final} = T_1 + \cdots + T_m$ be the final degree n polynomial to be added by Mora's algorithm. Here $T_1, \ldots, T_m$ are terms in R. A lower bound for the number of hard instances can be given by

$$|\{P_n^{final} + Q | Q \in I \setminus \{T_1, \ldots, T_m\}, deg(Q) \leq n\}|. \tag{5.19}$$

The idea here is that any polynomial in our ideal that contains P_n will be incorrectly seen as not a member of I for a long time.
Although a single example will be covered in more detail in section 5.3, figure 5.2 gives an impression of the distribution of degrees in some infinite Gröbner bases. Although not identical, they all appear to increase similarly, with what appears to be some periodicity.

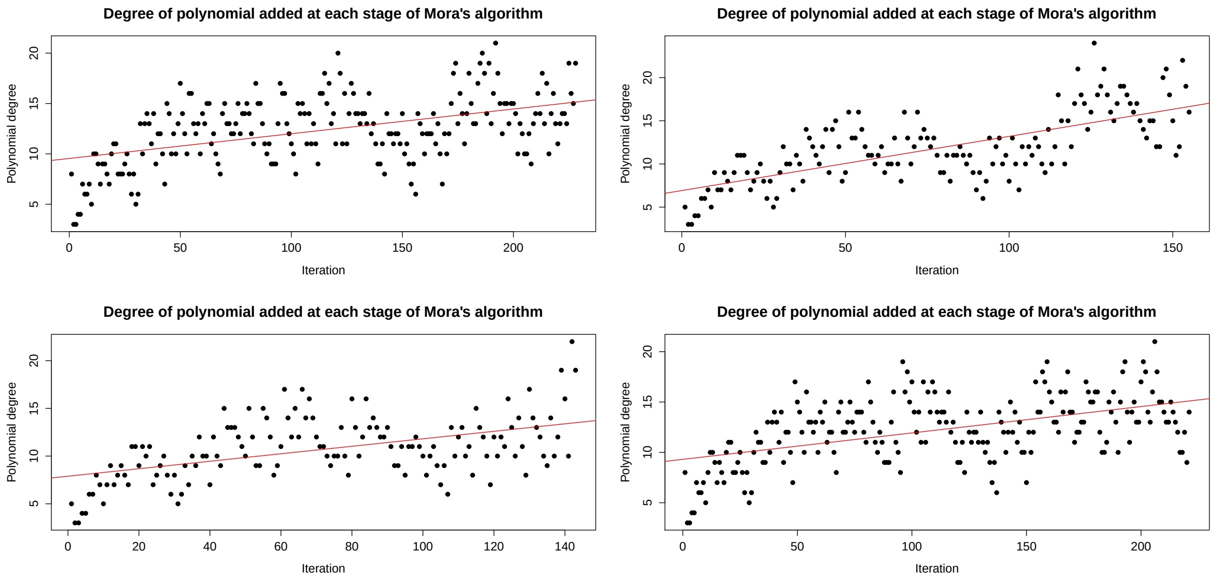


Figure 5.2: Experiments with different infinite Gröbner bases

5.3 Calculating the size of the balls in polynomial ideal

In this section, we will focus on finding the total number of polynomials of degree n in the ideal we have been studying in the previous chapter. This will give us a way of calculating the denominator of the fraction used to establish the generic complexity that was stated in the definition in section 5.1.4.

5.3.1 Monomial restriction on example ideal

The process will begin by just considering members of the ideal $\langle xzy + yz, yzx + zy \rangle$ that are of the form

$$m_{l1}(xzy + yz)m_{r1} + m_{l2}(yzx + zy)m_{r2}, \tag{5.20}$$

where $m_{l1}, m_{r1}, m_{l2}, m_{r2}$ are monomials in $k\langle x, y, z \rangle$.
A combinatorial argument will allow us to count how many monomials, in a finite number of variables, can be multiplied by the basis elements to produce a polynomial of degree n. Care needs to be taken however in checking if the leading terms cancel.
Taking into consideration degree 6 polynomials and higher, xzy and yzx can appear at non-overlapping points of the lead monomial in the form

$$LM = \cdots xzy \cdots yzx \cdots \text{ or } LM = \cdots yzx \cdots xzy \cdots . \tag{5.21}$$

Position of xzy	Position of yzx	Leading monomial
$[1,2,3]$	$[4,5,6]$	xzy^2zx
$[4,5,6]$	$[1,2,3]$	yzx^2xzy
$[1,2,3]$	$[3,4,5]$	$xzyzx?$
$[3,4,5]$	$[1,2,3]$	$yzxzy?$
$[2,3,4]$	$[4,5,6]$	$?xzyzx$
$[4,5,6]$	$[2,3,4]$	$?yzxzy$

Table 5.1: Possible ways for degree 6 leading terms to cancel

There are 2 overlapping patterns, these are of the form

$$\cdots xz(y)zx\cdots \text{ or } \cdots yz(x)zy\cdots, \tag{5.22}$$

where the variable in brackets is the overlap. Here the degree 6 monomials act as a motivating example in table 5.1. The question mark represents a variable that could be any one of the set $\{x, y, z\}$. This means in degree 6, there are 14 instances of leading terms cancelling.
To consider higher degrees it is useful to split up the investigation into the 2 sets: those where there is an overlap of the leading monomials and those where there isn't.

Lemma 5.1. *In the ideal generated by $\langle xzy + yz, yzx + zy\rangle \subset K\langle x, y, z\rangle$, the number of monomials of length $n \geq 6$ formed from monomials in $K\langle x, y, z\rangle$ multiplied by the leading monomials of the generators that cancel is equal to*

$$2 * 3^{n-6}(3(n-4) + T_{n-5}), \tag{5.23}$$

where T_n is equal to the n^{th} triangle number.

Proof. As mentioned above, we have 2 cases to deal with. First where the leading monomials overlap when calculating the S-polynomial i.e. of the form $l_1(xzyzx)r_1$ or $l_2(yzxzy)r_2$. The second case is where the 2 leading monomials appear at separate locations in the leading monomial of a sum of the generators i.e. of the form $l_1(xzy)m_1(yzx)r_1$ or $l_2(yzx)m_2(xzy)r_2$.
Case 1:
The length of $xzyzx$ is 5, so $length(l_1) + length(r_1) = n - 5$. Suppose without loss of generality we can choose the length of l_1 to be any value from $\{0, 1, \ldots, n-5\}$. This choice will decide the length of r_1. We therefore have $|\{0, 1, \ldots, n-5\}| = n-4$ ways to distribute the number of variables to l_1 and r_1. Since each variable can be one of x, y or z, there are $3^{n-5}(n-4)$ monomials of the form $l_1(xzyzx)r_1$ that meet our criteria. The same reasoning applies to $l_2(yzxzy)r_2$ giving us a total of $2 * 3^{n-5}(n-4)$ monomials.

Case 2:
Distributing the number of variables between l_1, m_1 and r_1 is just an application of the combinatorial result covering stars and bars, where we have $length(l_1) + length(m_1) + length(r_1) = n - 6$ stars and 2 bars (being the monomials xzy and yzx). This gives us $\binom{(n-6)+2}{2} = \binom{n-4}{2} = T_{n-5}$ ways to distribute the variables. Once again taking into account there are 3 possibilities for each variable and the same method applies to $l_2(yzx)m_2(xzy)r_2$, we have a total of $2 * 3^{n-6}T_{n-5}$ possibilities.
Summing the two cases gives us the result stated in the lemma. □

Now that it has been shown how many monomials will cancel in the leading terms of the generators, we need to check how this trend will continue. To count the number of polynomials formed in the ideal up to degree n, it is important to see if any linear combination of generators could equal zero.

Lemma 5.2. *In the ideal $I =< xzy + yz, yzx + zy >$, there do not exist non-zero polynomials $\alpha, \beta, \delta, \gamma \in K < x, y, z >$ such that $\alpha(xzy + yz)\beta + \delta(yzx + zy)\gamma = 0$.*

Proof. Firstly the general form of an element in the choice of ideal is

$$\begin{aligned}&(\alpha_1 + \alpha_2 + \cdots + \alpha_M)(xzy + yz)(\beta_1 + \beta_2 + \cdots + \beta_N)\\&+ (\gamma_1 + \gamma_2 + \cdots + \gamma_P)(yzx + zy)(\delta_1 + \delta_2 + \cdots + \delta_Q),\end{aligned} \tag{5.24}$$

where all the $\alpha, \beta, \gamma, \delta \in K < x, y, z >$. Suppose all the γ's and δ's equal 0 i.e. an ideal element that is a formed from a single generator. Let

$$maxDegree\{\alpha_1, \alpha_2, \dots, \alpha_M\} + maxDegree\{\beta_1, \beta_2, \dots, \beta_N\} = n. \tag{5.25}$$

Suppose without loss of generality $\text{LM}((\alpha_1+\cdots+\alpha_M)(xzy+yz)(\beta_1+\cdots+\beta_N)) = \alpha_1 xzy\beta_1$. Now for the whole polynomial to be equal to zero it must first be the case that the leading monomial must cancel. There must exist therefore α_i, β_j, i and j not both equal to 1 such that

$$\alpha_1 xzy\beta_1 - \alpha_i xzy\beta_j = 0. \tag{5.26}$$

It must be the case that xzy appears at a different location in second monomial therefore

$$\begin{aligned}&deg\{\alpha_1\} \neq deg\{\alpha_i\} \;\&\; deg\{\beta_1\} \neq deg\{\beta_j\},\\&\Rightarrow deg\{\alpha_1\} + deg\{\beta_j\} > deg\{\alpha_1\} + deg\{\beta_1\},\\&\text{or } deg\{\alpha_i\} + deg\{\beta_1\} > deg\{\alpha_1\} + deg\{\beta_1\},\end{aligned} \tag{5.27}$$

since $deg\{\alpha_i\} + deg\{\beta_j\} = n$. This a contradiction to what we have claimed to be the maximum degree earlier. Therefore, at least one of the γ's or δ's is not zero. If that is true then there are 4 possible cases we need to look at. We will focus on how the lead monomial will cancel.

1. Our leading monomial contains the overlap $xzyzx$.

$$\begin{aligned}&(\alpha_1^1\alpha_1^2\cdots\alpha_1^R+\cdots+\alpha_M)(xzy+yz)(zx\beta_1^1\beta_1^2\cdots\beta_1^S+\cdots+\beta_N)\\&-(\alpha_1^1\alpha_1^2\cdots\alpha_1^Rxz+\gamma_2+\cdots+\gamma_P)(yzx+zy)(\beta_1^1\beta_1^2\cdots\beta_1^S+\delta_2+\cdots+\delta_Q)\\&=\alpha_1^1\alpha_1^2\cdots\alpha_1^Ryzzx\beta_1^1\beta_1^2\cdots\beta_1^S-\alpha_1^1\alpha_1^2\cdots\alpha_1^Rxzzy\beta_1^1\beta_1^2\cdots\beta_1^S+\ldots\end{aligned}\tag{5.28}$$

2. Our leading monomial contains the overlap $yzxzy$.

$$\begin{aligned}&(\alpha_1^1\alpha_1^2\cdots\alpha_1^Ryz+\cdots+\alpha_M)(xzy+yz)(\beta_1^1\beta_1^2\cdots\beta_1^S+\cdots+\beta_N)\\&-(\alpha_1^1\alpha_1^2\cdots\alpha_1^R+\gamma_2+\cdots+\gamma_P)(yzx+zy)(zy\beta_1^1\beta_1^2\cdots\beta_1^S+\delta_2+\cdots+\delta_Q)\\&=\alpha_1^1\alpha_1^2\cdots\alpha_1^Ryzyz\beta_1^1\beta_1^2\cdots\beta_1^S-\alpha_1^1\alpha_1^2\cdots\alpha_1^Rzyzy\beta_1^1\beta_1^2\cdots\beta_1^S+\ldots\end{aligned}\tag{5.29}$$

3. Our leading monomial is of the form $\cdots xzy\cdots yzx\cdots$.

$$\begin{aligned}&(\alpha_1^1\alpha_1^2\cdots\alpha_1^R+\cdots+\alpha_M)(xzy+yz)(\beta_1^1\beta_1^2\cdots yzx\cdots\beta_1^S+\cdots+\beta_N)\\&-(\alpha_1^1\alpha_1^2\cdots\alpha_1^Rxzy\beta_1^1\beta_1^2\cdots\beta_1^i+\gamma_2+\cdots+\gamma_P)(yzx+zy)(\beta_1^{i+4}\cdots\beta_1^S+\delta_2+\cdots+\delta_Q)\\&=\alpha_1^1\alpha_1^2\cdots\alpha_1^Ryz\beta_1^1\beta_1^2\cdots yzx\cdots\beta_1^S-\alpha_1^1\alpha_1^2\cdots\alpha_1^Rxzy\beta_1^1\beta_1^2\cdots\beta_1^izy\beta_1^{i+4}\cdots\beta_1^S+\ldots\end{aligned}\tag{5.30}$$

4. Our leading monomial is of the form $\cdots yzx\cdots xzy\cdots$.

$$\begin{aligned}&(\alpha_1^1\alpha_1^2\cdots\alpha_1^jyzx\alpha_1^{j+4}\cdots\alpha_1^R+\cdots+\alpha_M)(xzy+yz)(\beta_1^1\beta_1^2\cdots\beta_1^S+\cdots+\beta_N)\\&-(\alpha_1^1\alpha_1^2\cdots\alpha_1^j+\alpha_2+\cdots+\alpha_M)(yzx+zy)(\alpha_1^{j+4}\cdots\alpha_1^Rxzy\beta_1^1\beta_1^2\cdots\beta_1^S+\delta_2+\cdots+\delta_Q)\\&=\alpha_1^1\alpha_1^2\cdots yzx\cdots\alpha_1^Ryz\beta_1^1\beta_1^2\cdots\beta_1^S-\alpha_1^1\alpha_1^2\cdots\alpha_1^jzy\alpha_1^{j+4}\cdots\alpha_1^Rxzy\beta_1^1\beta_1^2\cdots\beta_1^S+\ldots\end{aligned}\tag{5.31}$$

Similar to before, for the whole polynomial to be equal to zero, something must cancel the leading monomial of the remaining polynomial. Note here that if we are to cancel this time then it must be from something of the form l_1yzr_1 or l_2zyr_2, i.e. product on the left and right of the second terms in our generators. Note that if the new leading monomial was a product of the first terms of our generators again we would simply repeat the previous step.
Once again we will work through the 4 cases of the original leading monomial and show that the new leading terms can't cancel. When we considered only products of the generator $xzy+yz$, $p_1=l_1(xzy+yz)r_1$ and $p_2=l_2(xzy+yz)r_2$ we made an argument about the position of the xzy being different for the 2 polynomials being subtracted. This caused a contradiction as at least one of l_1, r_1, l_2, r_2 had a higher degree than what we asserted was the largest degree. We will now construct a similar argument to show we can't cancel the new leading monomial that contains either yz or zy.

1. The original lead monomial was $\alpha_1^1\alpha_1^2\cdots\alpha_1^R(xzy)zx\beta_1^1\beta_1^2\cdots\beta_1^S$.

We have $Deg(\alpha_1^1\alpha_1^2\cdots\alpha_1^R) = R$ and $Deg(zx\beta_1^1\beta_1^2\cdots\beta_1^S) = S+2$.
After that monomial has canceled we have LM $= \alpha_1^1\alpha_1^2\cdots\alpha_1^R xzzy\beta_1^1\beta_1^2\cdots\beta_1^S$.
The best choice is

$$\underbrace{(\alpha_1^1\alpha_1^2\cdots\alpha_1^R xzz)}_{\text{degree too big}}(xzy+yz)(\beta_1^2\cdots\beta_1^S). \tag{5.32}$$

The degree of the left monomial is $R+3 > R$.

2. The original lead monomial was $\alpha_1^1\alpha_1^2\cdots\alpha_1^R(yzx)zy\beta_1^1\beta_1^2\cdots\beta_1^S$.
We have $Deg(\alpha_1^1\alpha_1^2\cdots\alpha_1^R) = R$ and $Deg(zy\beta_1^1\beta_1^2\cdots\beta_1^S) = S+2$
After that monomial has canceled we have LM $= \alpha_1^1\alpha_1^2\cdots\alpha_1^R yzyz\beta_1^1\beta_1^2\cdots\beta_1^S$.
The best choice is

$$\underbrace{(\alpha_1^1\alpha_1^2\cdots\alpha_1^R y)}_{\text{degree too big}}(yzx+zy)(z\beta_1^1\beta_1^2\cdots\beta_1^S). \tag{5.33}$$

The degree of the left monomial is $R+1 > R$.

3. The original lead monomial was $\alpha_1^1\alpha_1^2\cdots\alpha_1^R(xzy)\beta_1^1\beta_1^2\cdots yzx\cdots\beta_1^S$.
We have $Deg(\alpha_1^1\alpha_1^2\cdots\alpha_1^R) = R$ and $Deg(\beta_1^1\beta_1^2\cdots yzx\cdots\beta_1^S) = S$.
After that monomial has canceled we have LM $= \alpha_1^1\alpha_1^2\cdots\alpha_1^R xzy\beta_1^1\beta_1^2\cdots\beta_1^i zy\beta_1^{i+4}\cdots\beta_1^S$.
The best choice is

$$\underbrace{(\alpha_1^1\alpha_1^2\cdots\alpha_1^R xz)}_{\text{degree too big}}(xzy+yz)(\beta_1^2\cdots\beta_1^i zy\beta_1^{i+4}\cdots\beta_1^S). \tag{5.34}$$

The degree of the left monomial is $R+2 > R$.

4. The original lead monomial was $\alpha_1^1\alpha_1^2\cdots\alpha_1^j(yzx)\alpha_1^{j+4}\cdots\alpha_1^R xzy\beta_1^1\beta_1^2\cdots\beta_1^S$.
We have $Deg(\alpha_1^1\alpha_1^2\cdots\alpha_1^j) = j$ and $Deg(\alpha_1^{j+4}\cdots\alpha_1^R xzy\beta_1^1\beta_1^2\cdots\beta_1^S) = S+R-j$.
After that monomial has canceled we have LM $= \alpha_1^1\alpha_1^2\cdots\alpha_1^j yzx\alpha_1^{j+4}\cdots\alpha_1^R yz\beta_1^1\beta_1^2\cdots\beta_1^S$.
The best choice is

$$(\alpha_1^1\alpha_1^2\cdots\alpha_1^{j-1})(yzx+zy)\underbrace{(zx\alpha_1^{j+4}\cdots\alpha_1^R yz\beta_1^1\beta_1^2\cdots\beta_1^S)}_{\text{degree too big}}. \tag{5.35}$$

The degree of the left monomial is $S+R-j+1 > S+R-j$.

In each case there has been a contradiction in degree size therefore these terms cannot cancel. This means that the overall polynomial could not equal zero. □

With the knowledge that there is no way to cause the two generators to fully cancel, it is now possible to establish the size of the balls of polynomials, degree n and below.

Corollary 5.1. *The total number of polynomials up to degree n in the ideal $K < xzy + yz, yzx + zy >$ is double exponential in n.*

Proof. Let k be the size of the coefficient K space for our polynomials in $K < x, y, z >$. Up to degree n, in 3 variables, there will be $\sum_{i=0}^{n} 3^i$ different monomials. Therefore the total number of polynomials up to degree n will be

$$k^{\sum_{i=0}^{n} 3^i} = k^{\frac{3(1-3^n)}{1-3}} = k^{\frac{3^{n+1}-3}{2}}, \tag{5.36}$$

by geometric progression.
Every polynomial that is not a member of the ideal will be correctly identified as not a member, regardless of how long Mora's algorithm has run. Therefore if all hard instances are contained within the ideal, it makes sense to compare the number of degree n polynomials and smaller in the ideal. For each of the generators, we are multiplying on the left and right by a polynomial. The formula above covers the total number of polynomials up to a given degree but now there are 2 collections. The total degree of the left and right-hand size must not exceed $n-3$ (since the degree of both generators is 3). Therefore now, for each degree i, $0 \leq i \leq n-3$, of the left polynomial, the right polynomial can have any degree, $0 \leq j \leq (n-3)-i$. The above formula can, therefore, be expanded on to give

$$k^{\sum_{i=0}^{n}(3^i \cdot \sum_{j=0}^{n-i} 3^j)}, \tag{5.37}$$

once again the summation in brackets is a geometric progression.
Because there are 2 generators in this ideal, this calculation needs to be used twice. The degree of one the generators linear combination has no impact on the other. Therefore, it is just a case of squaring the formula for the total number of polynomials in the ideal up to degree n i.e.

$$k^{\frac{1}{2}\sum_{i=0}^{n-3} 3^{n-2}-3^{i+1}} \cdot k^{\frac{1}{2}\sum_{i=0}^{n-3} 3^{n-2}-3^{i+1}} = k^{\sum_{i=0}^{n-3} 3^{n-2}-3^{i+1}}. \tag{5.38}$$

Both the total number of polynomials in $K < x, y, z >$ and the total number of polynomials in $K < xzy + yz, yzx + zy >$ grow according to double exponential functions. □

Since we know our hard instances are within our ideal, it is a good start to know this space isn't going to get small relative to the whole polynomial space.

5.4 How many hard instances are there in each ball?

In this section, we try to understand how many polynomials of a given degree are difficult to classify as a member or not of the ideal from Theorem 4.5. We will attempt to make an argument about what exactly is the proportion of these polynomials to the total number of polynomials established in the previous section.

5.4.1 Sorted Mora's algorithm

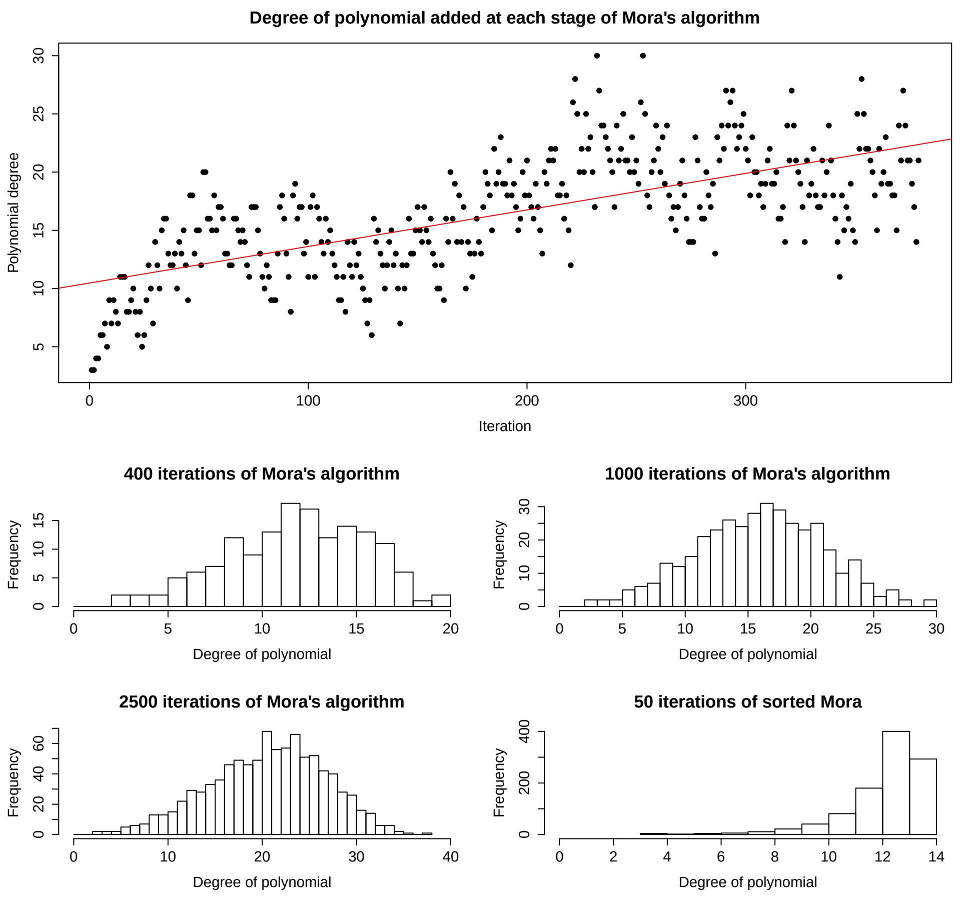


Figure 5.3: The top plot shows a time series of degrees of polynomial added at each iteration. The bottom right plot shows how many of each degree are added in the sorted variant of Mora's algorithm. The remaining plots show the distribution of polynomial degrees added after various iterations

To try and understand which polynomials are actual hard instances of the ideal membership problem, it is worth examining the order in which polynomials are added to an ideal during Mora's algorithm. Figure 5.3 shows various plots that give more of an insight into the distribution of polynomials being added. The top plot shows the degree of the polynomial added to the Gröbner basis at each iteration of Mora's algorithm for the ideal $I =< xzy + yz, yzx + zy >$. The bottom right plot was generated by altering Mora's algorithm to sort the polynomials waiting to be added to the basis in ascending order by degree. The Haskell implementation used here took drastically longer to run and therefore only the first 50 iterations are included. The final three plots all show the distribution of polynomial degrees added to I over various iterations of Mora's algorithm.

From the top plot it appears there is an increasing trend in the degrees of polynomials

being added however, it is by no means monotone and would be difficult given an observation at the t^{th} iteration, to predict the degree of the polynomial at the $t+1^{st}$ iteration. For any given polynomial we wish to perform the membership test on, a finite Gröbner basis will always suffice for testing. If the polynomial has degree n, then we do not need to worry about adding all the polynomials of degree $n+1$ and higher. As for polynomials of degree n and lower, all of the examples deal with polynomials in a finite number of variables, therefore, there must be a finite number of them. To find a hard instance, it is key to understand how hard it is to capture all polynomials of a given degree.
As the algorithm runs, a collection of S-polynomials are collected and are individually examined to see if they should be added to the Gröbner basis. Sorting this list by ascending order has the potential to ensure that the smallest polynomial, currently missing from the Gröbner basis, would almost always be added next (this is, of course, ignoring the fact that a large polynomial that is not even in the list yet may reduce to the smallest polynomial). The bottom right plot in figure 5.3 gives a sense that as the value of n increases, the number of degree n polynomials added during Mora's algorithm increases rapidly. This section shows that sorting the s-polynomials added to the basis in the algorithm is unfeasible for large n.

Lemma 5.3. *There are only 2 types of S-polynomials formed when running Mora's algorithm on the ideal $\langle xzy + yz, yzx + zy\rangle \subset K\langle x, y, z\rangle$. The 2 forms are*

1. $m_{1,j} \pm m_{2,j}$,
2. $m_{1,j} \pm m_{2,j-1}$,

where $m_{i,j}$ is a monomial of length j.

Proof. The initial ideal contains two type 2 polynomials. The s-polynomial formed from 2 polynomials, $p_1 = m^1_{1,j^1} + m^1_{2,j^1-1}, p_2 = m^2_{1,j^2} + m^2_{2,j^2-1}$, of this type is as follows

$$\begin{aligned} spol(p_1, p_2) &= l_{1,j^2-o}(m_{1,j^1} + m_{2,j^1-1}) - (m^2_{1,j^2} + m^2_{2,j^2-1})r_{1,j^1-o}, \\ &= l_{1,j^2-o}m_{2,j^1-1} - m'_{2,j^2-1}r_{1,j^1-o}, \end{aligned} \tag{5.39}$$

where l_{1,j^2-o} and r_{1,j^1-o} are monomials of length $j^2 - o$ and $j^1 - o$ respectively, chosen such that $l_{1,j^2-o}LM(p_1) = LM(p_2)r_{1,j^1-o}$ and o is the length of the leading monomial overlap.
Here we can assume without loss of generality that the overlap is of this form. Suppose instead we multiplied p_1 on the right by some r and multiplied p_2 on the left by some l. The two monomials l and r would still have the same degree as each other so the resultant S-polynomial would have the same form. We can't express p_1 as $l \cdot p_2 \cdot r$ or p_2 as $l \cdot p_1 \cdot r$ since the two leading monomials are the same length.
The two terms remaining in the s-polynomial each have length $j^1 + j^2 - o - 1$, i.e. are of

type 1.
Now calculating the s-polynomial formed from a type 2 polynomial p_1 and a type 1 polynomial $p_3 = m^3_{1,j^3} + m^3_{2,j^3}$ gives

$$\begin{aligned} spol(p_1, p_3) &= l_{2,j^3-o}(m_{1,j^1} + m_{2,j^1-1}) - (m^3_{1,j^3} + m^3_{2,j^3})r_{2,j^1-o}, \\ &= l_{2,j^3-o}m_{2,j^1-1} - m^3_{2,j^3}r_{2,j^1-o}. \end{aligned} \tag{5.40}$$

The two remaining terms in the s-polynomial have lengths $j^1 + j^3 - o$ and $j^1 + j^3 - o - 1$ and are therefore of type 2.
The only other option left is the s-polynomial formed from two type 1 polynomials, p_3 as above and $p_4 = m^4_{1,j^4} + m^4_{2,j^4}$ namely

$$\begin{aligned} spol(p_3, p_4) &= l_{3,j^4-o}(m^3_{1,j^3} + m^3_{2,j^3}) - (m^4_{1,j^4} + m^4_{2,j^4})r_{3,j^3-o} \\ &= l_{3,j^4-o}m^3_{1,j^3} - m^4_{1,j^4}r_{3,j^3-o}. \end{aligned} \tag{5.41}$$

The two remaining terms in the s-polynomial both have length $j^3 + j^4 - o$ and therefore are of type 1. Since no new types of polynomials have been formed and all possibilities have been covered, only type 1 and type 2 polynomials may be formed during Mora's algorithm for this particular ideal. □

Note that we have omitted the coefficients of the terms in the proof. Every term in the polynomials generated by Mora's algorithm for this particular input has a coefficient of either 1 or -1. Therefore, we adopt the standard of dropping the 1.

Lemma 5.4. *Suppose for now that during the reduction stage in Mora's algorithm that no polynomials are reduced to zero. The size of the Gröbner basis calculated for the ideal $I = \langle xzy + yz, yzx + zy \rangle \subset K\langle x, y, z \rangle$ during a sorted run of Mora's algorithm grows exponentially in n, the maximum degree of basis polynomial that needs to be found.*

Proof. After the first few iterations of Mora's algorithm, the following four polynomials are in the Gröbner basis and form a complete overlapping subset for I:

$$P_{i,j} = \{xzy + yz, yzx + zy, yzyz - zyzy, zyzyx + yzzy\}. \tag{5.42}$$

This means that every time a new polynomial is added to the Gröbner basis, there is a guarantee that there will be multiple overlaps with the polynomials already in the basis. The reason for this being there must be an overlap with the start of the leading monomial and one with the end of the leading monomial, meaning at least 2 new S-polynomials will be added to the list.
We are now interested in examining the degrees off the polynomials in $P_{i,j}$. Starting with the first two degree 3 polynomials, it follows from Lemma 5.3 that an S-polynomial formed from one of those polynomials and a degree n polynomial with a single element overlap

will either have degree $n+2$ or $n+1$. Similar logic applies for the degree 4 polynomial, resulting in a degree $n+3$ and $n+2$ polynomial. Finally, degree 5 polynomials, result in degree $n+4$ or $n+3$ polynomials. Therefore after a linear increase in n, the number of s-polynomials has at least doubled. $\square$

All that is left is to check how frequently polynomials are reduced to zero during Mora's algorithm, which is what we do in the next section.
Now for the hard instances within the ideal, all the important information about the ideal can be brought together to understand the generic complexity.

Theorem 5.4. *Let I_n be the number of polynomials in the ideal $K < xzy+yz, yzx+zy >$ of degree n or less. Suppose the total number of s-polynomials at least doubles each time we increase the degree n by a fixed amount. Let E_n be the set of polynomials in I_n such that the membership test on elements of E_n requires exponential time. Then*

$$lim_{n\to\infty}\frac{E_n}{I_n}=1, \tag{5.43}$$

that is

$$E=\cup_n E_n \tag{5.44}$$

is generic.

Proof. Remember about the structure of the polynomials Mora's algorithm outputs. It shows that they all have two terms. Denote these two terms T_1 and T_2. We define the sets

$$I_n^{+T_1}=\{\text{All polynomials in } I_n \text{ that contain } T_1\} \tag{5.45}$$

and

$$I_n^{-T_1}=\{\text{All polynomials in } I_n \text{ that don't contain } T_1\} \tag{5.46}$$

where $I_n = I_n^{-T_1}\cup I_n^{+T_1}$ (with a similar argument for $I_n^{-T_2}$ and $I_n^{+T_2}$).
There is a clear bijective mapping between these two sets

$$\begin{aligned} I_n^{-T_1} &\to I_n^{+T_1} \\ p &\mapsto p+T_1. \end{aligned} \tag{5.47}$$

Furthermore if we define

$$\begin{aligned} (I_n^{+T_1})^{+T_2} &= \{\text{All polynomials in } I_n^{+T_1} \text{ that contain } T_2\} \\ (I_n^{+T_1})^{-T_2} &= \{\text{All polynomials in } I_n^{+T_1} \text{ that don't contain } T_2\} \end{aligned} \tag{5.48}$$

with a similar bijective mapping as before, then we must have

$$Pr(\text{randomly selecting a polynomial containing } T_1 \text{ and } T_2)=\frac{1}{4}. \tag{5.49}$$

The exponential growth discussed in Mora's algorithm as the degree increases have been established in Lemma 5.4. From this, it must be that the size of E_n increases by at least 1 for each degree increase. This increases the proportion of polynomials in I_n that have a difficult subset of terms.
Let X_t be the proportion in I_n of polynomials that include at least one of the difficult polynomials as a subset of its terms. As the number of polynomials grows by 1 at each time step, the proportion evolves in the following way

$$X_t \geq X_{t-1} + \frac{1}{4}(1 - X_{t-1}), X_1 = \frac{1}{4}, 0 \leq X_t \leq 1. \tag{5.50}$$

Under the restrictions on X_t, $\frac{1}{4}(1 - X_{t-1})$ is non-negative. Thus $X_t \to 1$ as $t \to \infty$. The exponential time ideal membership is therefore generic. □

5.5 Will we lose too many terms?

It has been shown so far that the number of polynomials added to the basis grows exponentially with the number of iterations. There is one step in Mora's algorithm where we perform the division algorithm on the current polynomial, with respect to the current partial Gröbner basis. If we want to continue to assert that we have exponential growth in polynomials, then we need to know if this step in the algorithm is drastically reducing the number of polynomials added to the basis. The structure of the Gröbner basis is not simple so working out how polynomials will be reduced is far from simple. Here we will try to justify our claim.

5.5.1 Haskell experiments

Looking back at the plot from 5.3, we see the number of polynomials grow exponentially as the degree increases. Diagrams show what happens as the number of iterations of the Haskell program on the standard Mora's algorithm increased. As identified in the middle plots and bottom left plot, the distribution of degrees appears to be fairly similar in each. This gives a reason to believe that as we increase the degree size of polynomials past the initial sorted run, the same exponential looking growth will continue.

5.5.2 Reording polynomials

We know that we can generate an infinite Gröbner basis because we can prove there are infinite sets that must be part of it. These inductively defined sets come in pairs. The first polynomial P_1 in the first set generates S-polynomials, one of which is Q_1, the first polynomial in the second set. One of the S-polynomials for Q_1 is then P_2, the second polynomial from the first set. This back and forth between these two sets go on infinitely as illustrated in figure 5.4. There will always be one of these polynomials in the collection

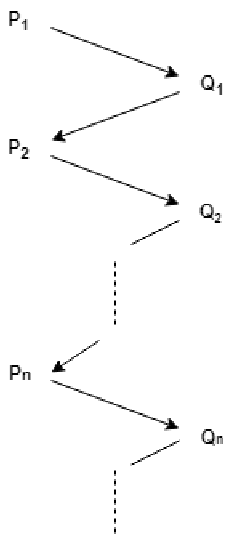


Figure 5.4: Two inductively sets create each other

waiting to be processed. Therefore, if we prioritize these polynomials in each iteration we know nothing will be removed by the division step. Of course, this differs greatly from the sorted variant of the algorithm, as much higher degree polynomials will be added to the basis before all the low degree polynomials.

5.5.3 Polynomial distributions

While the previous argument does mean that we will never actually lose a polynomial in Mora's algorithm, the order in which we go over the polynomials is extremely particular. As stated when looking at the sorted Mora's algorithm variant, sorting in each iteration is time-consuming. At least, in that case, we were benefiting by putting a priority on low degree polynomials. The typical (not necessarily sorted) version of Mora's algorithm deals with the polynomials in a "first on, first off" way. To study this, we must class every polynomial as one of 3 types.

- s_1 : The number of polynomials that will be removed in Mora's algorithm.
- s_2 : The number of polynomials that will not be removed in Mora's algorithm, but not part of an inductively defined set.
- s_3 : The number of polynomials that are part of an inductively defined set.

As we work through the polynomials waiting to be added to the Gröbner basis, the remaining collection will evolve depending on what type of polynomial is currently being processed. We will denote by the triple (s_1, s_2, s_3) the number of each type of polynomial waiting to be processed. Depending on what type is chosen for the current iteration of the algorithm, the values in this triple will be updated in one of 3 ways. The following denotes the evolution $\xrightarrow{s_i}$ as a type s_i polynomial is chosen, $i = 1, 2, 3$.

$$\begin{aligned} (s_1, s_2, s_3) &\xrightarrow{s_1} (s_1 - 1, s_2, s_3), \\ (s_1, s_2, s_3) &\xrightarrow{s_2} (s_1 + \delta_1, s_2 - 1 + \delta_2, s_3 + \delta_3), \\ (s_1, s_2, s_3) &\xrightarrow{s_3} (s_1 + \delta_4, s_2 + \delta_5, s_3 + \delta_6), \end{aligned} \tag{5.51}$$

where the δ's are subject to

$$\delta_1 + \delta_2 + \delta_3 \geq 2, \; \delta_4 + \delta_5 + \delta_6 \geq 1, \; \delta_i \in \mathbb{N} \cup \{0\}, i = 1, \ldots, 6. \tag{5.52}$$

The evolution of s_1 is straightforward. It states that the total number of s_1 polynomials waiting to be processed decreases by one after we process the current one. When we process an s_2 polynomial, we know from Lemma 5.4 that at least 2 new polynomials will be added to our collection, however, it is difficult to say what type they will be. Finally when processing an s_3 type polynomial, again at least 2 new polynomials will be added. The difference here is that at least one of those polynomials will be another s_3 type (which is why there is no negative 1 in this case).
It is very difficult to say what our values for δ can be without a lot of information about the polynomials. However, we can consider what sort of values would confirm our belief that not too many polynomials are removed. Earlier we worked under the assumption that no polynomials were being removed, so essentially we had $s_1 = \delta_1 = \delta_4 = 0$. We could, therefore, express the expected number of polynomials to be added during each iteration of Mora's algorithm to be

$$E[\text{new polynomials}] = \frac{s_2}{s_2 + s_3}(\delta_2 + \delta_3 - 1) + \frac{s_3}{s_2 + s_3}(\delta_5 + \delta_6) \geq 2. \tag{5.53}$$

We showed that this expectation will be greater than or equal to 2, however, as long as it is greater than 1 for each iteration we will see exponential growth. Therefore if we allow s_1, δ_1, δ_4 to be non zero we need them to satisfy

$$\frac{-s_1}{s_1 + s_2 + s_3} + \frac{s_2}{s_1 + s_2 + s_3}(\delta_1 + \delta_2 + \delta_3 - 1) + \frac{s_3}{s_1 + s_2 + s_3}(\delta_4 + \delta_5 + \delta_6) > 1. \tag{5.54}$$

Subbing in the previous part, assuming our expectation is as low as possible at 2, we have

$$\frac{-s_1}{s_1 + s_2 + s_3} + (1 - \frac{s_1}{s_1 + s_2 + s_3}) \cdot 2 > 1 \Rightarrow \frac{s_1}{s_1 + s_2 + s_3} < \frac{1}{3}. \tag{5.55}$$

Therefore, if on average s_1 makes up a third or less of the triple (s_1, s_2, s_3) then we still have exponential growth in polynomials.
With this result, we can provide the final version of the complexity theorem.

Theorem 5.5. *Let I_n be the number of polynomials in the ideal $K < xzy+yz, yzx+zy >$ of degree n or less. Suppose at each stage of Mora's algorithm at most one third of the polynomials waiting to be added have the property that they will be reduced to zero. Let E_n be the set of polynomials in I_n such that the membership test on elements of E_n requires exponential time. Then*

$$lim_{n\to\infty}\frac{E_n}{I_n} = 1, \tag{5.56}$$

that is

$$E = \cup_n E_n \tag{5.57}$$

is generic.

Proof. After the first few iterations of Mora's algorithm, the following four polynomials are in the Gröbner basis and form a complete overlapping subset for I:

$$P_{i,j} = \{xzy + yz, yzx + zy, yzyz - zyzy, zyzyx + yzzy\}. \tag{5.58}$$

This means that every time a new polynomial is added to the Gröbner basis, there is a guarantee that there will be multiple overlaps with the polynomials already in the basis. The reason for this being there must be an overlap with the start of the leading monomial and one with the end of the leading monomial, meaning at least 2 new S-polynomials will be added to the list.
We are now interested in examining the degrees off the polynomials in $P_{i,j}$. Starting with the first two degree 3 polynomials, it follows from Lemma 5.3 that an S-polynomial formed from one of those polynomials and a degree n polynomial with a single element overlap will either have degree $n+2$ or $n+1$. Similar logic applies for the degree 4 polynomial, resulting in a degree $n+3$ and $n+2$ polynomial. Finally, degree 5 polynomials, result in degree $n+4$ or $n+3$ polynomials. From our assumption, we know that at most a third of these polynomials will reduce to zero when we attempt to add them to our Gröbner basis. We have established that with that property, on average, we will be adding at least 2 new polynomials to our candidate list each time we test a candidate. Consequently, after a linear increase in n, the number of s-polynomials has at least doubled.
By Theorem 5.4 it must be that the exponential time for the ideal membership is therefore generic under the given conditions. □

Chapter 6

Conclusion

This thesis aimed to expand on the various methods available within fully homomorphic encryption. While partially homomorphic encryption methods were a good proof of concept for the theory, there was very little practical use for these methods. In particular, we wanted to keep in mind the real-world aspects of implementing such methods. For example, both the high levels of parallelism for the vector approach and the concept of pushing a lot of the computation to the initial setup in the polynomial approach were of great interest.
One concern with using homomorphic encryption protocols was that they provide a weaker privacy guarantee than randomized encryption [Curino *et al.* (2011)]. So, while in theory, we may perform any operation we wish on encrypted data, it may not always be the best approach to use these methods for any kind of data. For example, if our sole goal is to be able to search for particular data, then any non-randomised method would suffice. In fact, for a large proportion of sensitive data (addresses, bank details etc), there is no reason to move away from the strongest forms of encryption already available.
The complex structure of polynomials in comparison to integers may not be as daunting as initially expected. Servers do not last forever and when one breaks we lose access to all the information on it. We require methods to back up our data so that we can retrieve the information [Dimakis *et al.* (2010)]. Figure 6.1 gives a simple example of backing up data from two different full servers onto only one extra server. The data on any of these servers can be retrieved by taking a linear combination of the other two. More complex and efficient methods may be possible if we consider how to store the schemes we've looked at based on vector spaces or polynomials.
This current research in homomorphic encryption has the potential to make big differences in the world of data analysis, however, there remain limitations to consider. One example is the publisher-subscriber (often referred to as pub-sub) model. In this model, various users/devices publish data to different categories. Users who are interested in any of these categories can subscribe to them and collect that data (figure 6.2 gives an example of this

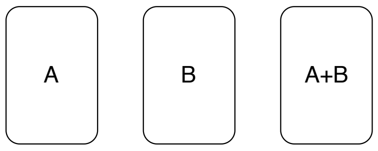


Figure 6.1: One server failure safe storage example

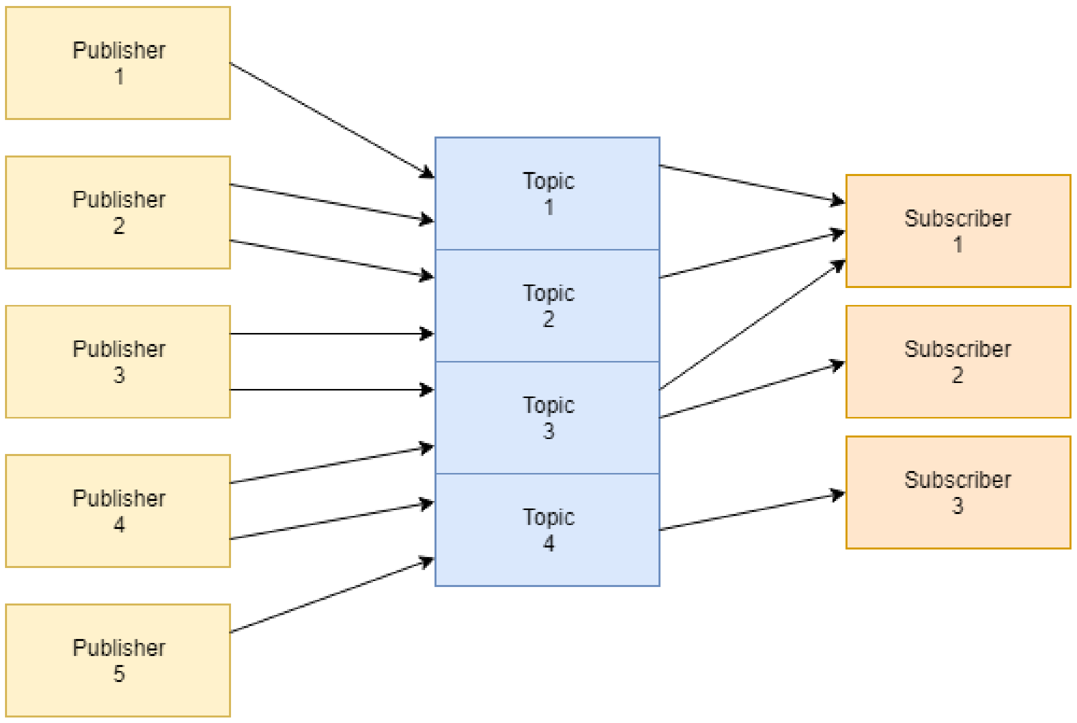


Figure 6.2: Pubsub style of data collection

setup). The issue here lies in the fact that if someone wishes to analyse encrypted data from a certain category, all publishers must encrypt using the same scheme and key. That is why we have public key systems but then we must ask who is the one deciding which method of encryption is being used? Also, if multiple subscribers want to analyse and decrypt the results, how do we ensure they can each get the key without other malicious users gaining access? Furthermore, the pub-sub model is something that is of interest to the IOT industry where computing power is limited so we wish to keep the amount of communication back and forth as low as possible. Publishing parties may also wish to learn from the collective data sets but not enclose their data to the other publishers. Fortunately, homomorphic encryption doesn't have to be the direct answer to all these problems as there are already promising results in areas such as multi-party private data analysis [Gascón *et al.* (2017)].

In chapter 3, we explored the capability of quantum computers to create a need for new schemes and identify what positive effect quantum properties would entail. The scheme we looked at was based on the closest vector problem. The idea is that, depending on a choice of basis, it can be difficult to tell which lattice point a random non-lattice point is closest to. The quantum adaption took a superposition of all these random points. Meaning that we had a better understanding of the error but, if an attacker tried to observe a ciphertext, it would collapse back down to the classic version.

The classical version was homomorphic under addition but would run the risk of incorrect decryption if performed too many times. A combination of a quantum addition circuit, measurements of particular qubits, and an extra control variable allowed us to perform ciphertext additions without the hassle of re-encryption that SHE's deal with.

The control variable that was introduced to help with error correction did present its own potential for an attack. Protecting this variable from malicious use required us to consider the variable in two different states. The first was when the data was at rest i.e. when we aren't operating on the data. We adapted the well known key exchange method and used it as a method for detection to see if someone had tried to observe the variable. As for the case of data in use, we used methods from classical computing. Although our goal is to do the majority of the work in the cloud, we accepted that a small sacrifice of performing the first few operations in a private cloud would ensure that individual data should be protected.

The GGH scheme may not be suitable for real-world use, however, it was worth studying as the encryption method is fairly simple, meaning the error correction concept should be easier to grasp. The main feature that we manipulated was the fact the error involved came from a finite distribution. Therefore, if this property can be ensured in more practical schemes, the same overall idea should theoretically work with them too. In particular, we looked at a fairly unnatural multiplication method here, it would be better to use a protocol where multiplication is more straightforward.

Although these lattice-based schemes certainly seem to be more protected from quantum attacks than schemes that are vulnerable to the prolific Shor's algorithm, recent work in [Joseph *et al.* (2020)] suggests that there may quantum methods that could put the security of these lattice-based schemes at risk.

In chapter 4, we shifted back to classical computing and looked at a homomorphic scheme in a polynomial ring. Similarly, as was discussed in the introduction, we looked to keep the encryption function conceptually and computationally simple, achieved by leaving most of the work to the initial key exchange. The encryption method was homomorphic under addition but a little more work was needed for the function to be homomorphic under multiplication. As suggested in [Vasco & Steinwandt (2015)], it is more important to think of the decryption method as homomorphic. The multiplication of ciphertexts wasn't exactly of the form we wanted, this was because multiplication of ciphertexts introduced extra terms that we didn't want. The choice of the quotient in the decryption did ensure we

output the polynomial we expected if we'd multiplied the plaintexts by reducing the unwanted terms to zero.
A concern with using polynomials as apposed to integers is that the structure is more computationally expensive. The problem of long computation was made worse because we could not cancel many terms until the very end. This was salvaged somewhat by being able to quotient out terms that included a square of our second key. Further work needs to be done to see if there is a subtler choice of quotient space that would still enable the required terms to cancel.
Although in our key exchange Alice does not make her choice of ideal public, we were able to identify a potential attack given that an eavesdropper could make an educated guess. We showed an example where Alice was able to choose polynomials that are members of ideals with two key properties. The first was that they include the public basis. The second was that the ideal contained elements outside of the public ideal that were not included in Alice's secret choice. These 2 properties suggest that there is a non zero probability that a random choice of ideal for an attack would not be able to distinguish between the key from the list which was chosen and a decoy. Future work should look into increasing this probability as much as possible.
Problems such as factoring integers have been around for a long time and no well-known solution (outside of a quantum solution) is available. We can, therefore, be fairly trustworthy of protocols such as RSA. The ideal membership problem is a lot newer and has yet to prove its resilience to the same extent from cryptanalysis. Despite our attempts to cover the potential alternative attacks, it may be that the initial problem itself needs a better understanding of which instances can be considered safe.
The theory of braid group cryptography has already been considered as a real-world security protocol. While this consideration may seem promising, various attacks on systems based on the multiple conjugacy search problem have been presented. One example of such attack is a length based attack which aims to reduce the length of an element of interest by repeatedly measuring its length after conjugating by choice words from a group of interest [Myasnikov & Ushakov (2007)]. Another example of an attack suggests that schemes based on the conjugacy search problem may be broken down to the easier decomposition problem (i.e. instead of trying to find conjugate elements, find any elements that belong to a given subset that we can multiply on the left and right to create an equal element) [Shpilrain & Ushakov (2006)]. Unfortunately, further studies have suggested that methods such as the algebraic eraser may be breakable [Blackburn & Robshaw (2016)]. Although fundamentally there is a difference between an algebra and a group, the fact that the Hecke algebra encryption method is based on conjugation does give rise to concern about the security of the method.
The status of Polly Cracker and Gröbner bases for the basis of cryptographic protocols appear to be unclear. Despite suggestions that these avenues may not be the best choice

for cryptography due to inefficiency and weaknesses found [Barkee *et al.* (1994)] [AL-Rummana & Shende (2018)], an effort is still made to try and find solutions surrounding these problems [Rai (2004)] [Albrecht *et al.* (2016)].

In chapter 5, we considered the generic difficulty of finding a partial Gröbner basis that would make ideal membership testing efficient. We claimed that this was exponentially difficult, but the proof did include an assumption about which polynomials Mora's algorithm would add to the basis.

The overall idea of the proof was that, if we are working with a finite number of variables, we wanted to add enough polynomials to the basis so that there would always be at least two s-polynomials generated at each iteration. We did this by running the algorithm long enough so that for each variable there was at least one polynomial whose lead monomial began with that variable. The same also went for the end of the lead monomial. This was enough to show that as the degrees increased, the number of iterations would at least double. An assumption we made here was that not too many polynomials would be reduced to zero.

The first justification of our assumption was from Haskell simulations. We saw that for differing numbers of iterations, a similar-looking distribution of polynomial degrees was output each time. We showed that for the first few degrees that there was exponential growth and so we have reason to believe that this trend would continue. The second justification used information that proves a basis has an infinite Gröbner basis in the first place. We show inductively defined sets are generated during Mora's algorithm and come in pairs. We argued that if we prioritize finding these first then we would never get to the polynomials that would be thrown away. The problem with this is that after reaching a certain degree, no new polynomials added would be useful for membership testing. The third approach looked to classify the polynomials that were generated. The classes were based on their future effect on the number of polynomials added to the Gröbner basis. As the algorithm iterated over these polynomials the proportions of each constantly changed. We stated an upper bound for the proportion of the class we didn't want to see at each iteration if we wanted membership testing to remain difficult. Further research into this area may consider using something like Lotka-Volterra [Freedman (1980)] to better model the population changes.

In summary, there appear to be three important aspects of designing a fully homomorphic cryptosystem. As expected, the fundamental aspect is that the cryptosystem is built on what is presumed to be a difficult mathematical problem. Unfortunately, that means a lot of these schemes are built on shaky foundations. Anyone wanting to use these schemes will have to put a lot more faith that these problems won't be solved due to the relatively smaller research in finding solutions than much older problems. Thankfully however there does appear to be multiple candidate problems.

The next aspect is finding an appropriate homomorphism that can be built around these problems. We've seen both in this work and many other publications that there are multiple approaches to finding homomorphisms under both addition and multiplication. Unlike methods such as RSA, where the homomorphism was a simple consequence of the function, we did need to put more effort into the choice of parameters and techniques for our operations to ensure addition and multiplication worked how we wanted.

Performing simple operations becomes computationally more complex brings us to the final aspect of what a practical system needs. All of this work to create a system where private data can be safely analysed in a public cloud is useless if it takes an unreasonable amount of time to get the results we are interested in. Therefore, we need to ensure that once we have something that works, it needs to be as efficient as possible. Now, due to the inherent more complicated structure, it seems unlikely that these systems will be as fast as what exists for standard encryption. Hopefully, as technology inevitably gets better this difference in speed will be compensated for.

Appendix A

A.1 Polynomial scheme example

Now either Alice or Bob can look at encrypting and operating on data. We already discussed that an appropriate choice of quotient space is needed to make the calculations work. Lets assume Alice and Bob repeat the process once again to get a second secret key. For the sake of brevity, well say $yxy + z$. Now consider the ideal

$$\langle S_B(yxy + z) - 1, (yxy + z)S_B - 1\rangle, \tag{A.1}$$

recalling that $S_B = xyxzyx + xy^2zx + zyxyz + z^2xyz$. Our requirement that we have two secret keys that satisfy the inverse property will hold if we quotient out our polynomials in our ciphertext by this ideal i.e. anytime we see the product of our two keys appear in a ciphertext, we can replace it with a 1.
To prevent tedious amounts of expanding brackets, we will keep a lot of the polynomials factored and we will also make the substitutions

$$A = xyxzyx, B = xy^2zx, C = zyxyz, D = z^2xyz, W = yxy. \tag{A.2}$$

This means that we have the public ideal $\langle (W + z)^2\rangle$ that we can use to reduce terms while in the cloud. We will also have the private ideal $\langle (A+B+C+D)(W+z)-1, (W+z)(A+B+C+D)-1\rangle$ to be used for further reduction once offline. Now lets do a simple calculation.

$$\begin{aligned}
&(Enc(x) + Enc(y)) * Enc(z)\\
=&((A+B+C+D)x(W+z) + (W+z)x(A+B+C+D)\\
+&(A+B+C+D)y(W+z) + (W+z)y(A+B+C+D))\\
*&((A+B+C+D)z(W+z) + (W+z)z(A+B+C+D))\\
=&((A+B+C+D)(x+y)(W+z) + (W+z)(x+y)(A+B+C+D))\\
*&((A+B+C+D)z(W+z) + (W+z)z(A+B+C+D))\\
=&(A+B+C+D)(x+y)(W+z)(A+B+C+D)z(W+z)\\
+&(A+B+C+D)(x+y)(W+z)^2z(A+B+C+D)\\
+&(W+z)(x+y)(A+B+C+D)^2z(W+z)\\
+&(W+z)(x+y)(A+B+C+D)(W+z)z(A+B+C+D).
\end{aligned} \tag{A.3}$$

Our public ideal space kills off multiples of $(W + z)^2$, so that leaves us with

$$\begin{aligned}
&(A+B+C+D)(x+y)(W+z)(A+B+C+D)z(W+z)\\
+&(W+z)(x+y)(A+B+C+D)^2z(W+z)\\
+&(W+z)(x+y)(A+B+C+D)(W+z)z(A+B+C+D).
\end{aligned} \tag{A.4}$$

Performing the decryption method, we have

$$\begin{aligned}
&Dec((Enc(x) + Enc(y)) * Enc(z))\\
=&(W+z)((A+B+C+D)(x+y)(W+z)(A+B+C+D)z(W+z)\\
+&(W+z)(x+y)(A+B+C+D)^2z(W+z)\\
+&(W+z)(x+y)(A+B+C+D)(W+z)z(A+B+C+D))(A+B+C+D)\\
=&(W+z)(A+B+C+D)(x+y)(W+z)(A+B+C+D)z(W+z)(A+B+C+D)\\
+&(W+z)^2(x+y)(A+B+C+D)^2z(W+z)(A+B+C+D)\\
+&(W+z)^2(x+y)(A+B+C+D)(W+z)z(A+B+C+D)^2.
\end{aligned} \tag{A.5}$$

Once again reducing according to our public ideal we are left with

$$(W+z)(A+B+C+D)(x+y)(W+z)(A+B+C+D)z(W+z)(A+B+C+D). \tag{A.6}$$

Now that we are offline we can use the private ideal which gives us

$$1 * (x+y) * 1 * z * 1 = (x+y) * z, \tag{A.7}$$

as required.

A.2 Full quantum circuit

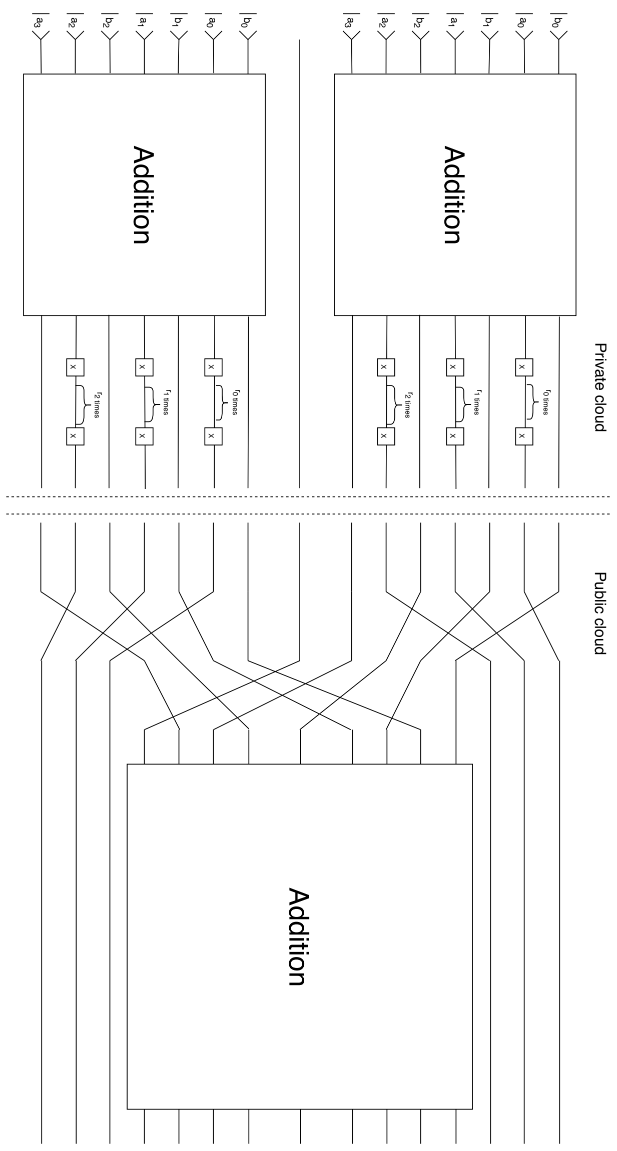


Figure A.1: Quantum encryption and addition of ciphertexts

A.3 Using the Haskell code

For testing purposes, code was written in haskell to find Gröbner bases from a list of polynomials (representing the original ideal). To run the code in the terminal first move to the directory where NonCommutative.hs is saved. To compile the code run the command

```
ghc NonCommutative.hs
```

The program is then run using

```
./NonCommutative
```

(\ instead of / if on windows). Upon running that command there will be a prompt “Enter your list of polynomials“. The entire list of polynomials must be surrounded by quotes. The polynomials are separated by commas (no spaces). Each term in a polynomial must be expressed by a triple:

1. Sign, either + or -,
2. Coefficient,
3. Monomial, capital letters for variables.

The ordering used is degLex and would need to be changed in the code if another ordering is desired. Letters M, T, P cannot be used as variables as they are reserved for defining monomials, terms and polynomials respectively. Note that unlike in normal notation, redundancy such as a coefficient of 1 or leading term having a plus in front can't be removed.
Example: If we wish to enter the ideal used in chapter 4, $\langle xzy + yz, yzx + zy \rangle$, we would type “+1XZY+1YZ,+1YZX+1ZY“ into the terminal. The number of iterations of the algorithm is set to 100 currently but can be modified in the code on line 453.

Bibliography

Acar, A., Aksu, H., Uluagac, A. S. & Conti, M. 2018 A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)* **51** (4), 1–35.

Ackermann, P. & Kreuzer, M. 2006 Gröbner basis cryptosystems. *Applicable Algebra in Engineering, Communication and Computing* **17** (3-4), 173–194.

Adams, W. W., Adams, W. W., ADAMS, W. H., Loustaunau, P. & Adams, W. W. 1994 *An introduction to Grobner bases*. American Mathematical Soc.

AL-Rummana, G. A. & Shende, G. 2018 Homomorphic encryption for big data security: A survey .

Alagic, G., Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Liu, Y.-K., Miller, C., Moody, D., Peralta, R. *et al.* 2019 *Status report on the first round of the NIST post-quantum cryptography standardization process*. US Department of Commerce, National Institute of Standards and Technology.

Albrecht, M. R., Faugère, J.-C., Farshim, P., Herold, G. & Perret, L. 2016 Polly cracker, revisited. *Designs, Codes and Cryptography* **79** (2), 261–302.

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M. *et al.* 2017 Understanding the mirai botnet. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 1093–1110.

April, C., Forbes, M., Ideal, T., Problem, M. & Gr, T. 2012 Membership Testing pp. 1–6.

Arora, S. & Barak, B. 2009 *Computational complexity: a modern approach*. Cambridge University Press.

Artin, E. 1947 Theory of braids. *Annals of Mathematics* pp. 101–126.

Barkee, B., Can, D. C., Ecks, J., Moriarty, T. & Ree, R. 1994 Why you cannot even hope to use gröbner bases in public key cryptography: an open letter to a scientist who failed and a challenge to those who have not yet failed. *Journal of Symbolic Computation* **18** (6), 497–501.

Barker, E., Barker, W., Burr, W., Polk, W. & Smid, M. 2012 Recommendation for key management part 1: General (revision 3). *NIST special publication* **800** (57), 1–147.

Barvinok, A. 2002 *A course in convexity*, , vol. 54. American Mathematical Soc.

Becker, A., Gama, N. & Joux, A. 2013 Solving shortest and closest vector problems: The decomposition approach. *IACR Cryptology ePrint Archive* **2013**, 685.

Becker, T. & Weispfenning, V. 1993 Gröbner bases. In *Gröbner Bases*, pp. 187–242. Springer.

BENNETT, C. 1984 Quantum crytography. In *Proc. IEEE Int. Conf. Computers, Systems, and Signal Processing, Bangalore, India, 1984*, pp. 175–179.

Bennett, C. H., Bessette, F., Brassard, G., Salvail, L. & Smolin, J. 1992 Experimental quantum cryptography. *Journal of cryptology* **5** (1), 3–28.

Blackburn, S. R. & Robshaw, M. J. 2016 On the security of the algebraic eraser tag authentication protocol. In *International Conference on Applied Cryptography and Network Security*, pp. 3–17. Springer.

Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C. & Zhandry, M. 2011 Random oracles in a quantum world. In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 41–69. Springer.

Buchberger, B. 1965 Ein algorithmus zum auffinden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal. *PhD thesis, Universitat Insbruck* .

Buchberger, B. & Winkler, F. 1998 *Gröbner bases and applications*, , vol. 17.

Bulygin, S. 2005 Chosen-ciphertext attack on noncommutative polly cracker. *CoRR* **abs/cs/0508015**.

Bump, D. 2010 Hecke Algebras pp. 1–120.

Cheon, J. H. & Jun, B. 2003 A polynomial time algorithm for the braid diffie-hellman conjugacy problem. In *Annual International Cryptology Conference*, pp. 212–225. Springer.

Colbeck, R. & Kent, A. 2011 Private randomness expansion with untrusted devices. *Journal of Physics A: Mathematical and Theoretical* **44** (9), 095305.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. 2009 *Introduction to algorithms*.

Cortés, R., Hernández, J. & Morales, E. 2007 Noncommutative gröbner basis public key cryptosystems and their cryptanalysis .

Cox, D., Little, J. & O'shea, D. 1992 *Ideals, varieties, and algorithms*, , vol. 3. Springer.

Curino, C., Jones, E. P., Popa, R. A., Malviya, N., Wu, E., Madden, S., Balakrishnan, H. & Zeldovich, N. 2011 Relational cloud: A database-as-a-service for the cloud .

Dean, J. & Ghemawat, S. 2008 Mapreduce: simplified data processing on large clusters. *Communications of the ACM* **51** (1), 107–113.

Deutsch, D. & Jozsa, R. 1992 Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* **439** (1907), 553–558.

Dimakis, A. G., Godfrey, P. B., Wu, Y., Wainwright, M. J. & Ramchandran, K. 2010 Network coding for distributed storage systems. *IEEE transactions on information theory* **56** (9), 4539–4551.

Drakos, N. 1996 A grobner basis theorem on elimination ideals .

Easttom, C. 2017 An overview of quantum cryptography with lattice based cryptography. *IOSR Journal of Mathematics* **13** (6), 11–17.

Ekert, A., Hayden, P. & Inamori, H. 2001 Basic concepts in quantum computation. In *Coherent atomic matter waves*, pp. 661–701. Springer.

Evans, G. A. 2006 Noncommutative involutive bases. *arXiv preprint math/0602140* .

Flores, R. & Kahrobaei, D. 2017 Cryptography with right-angled artin groups. *Theoretical and Applied Informatics* **28** (3), 8–16.

Freedman, H. I. 1980 *Deterministic mathematical models in population ecology*, , vol. 57. Marcel Dekker Incorporated.

Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S. & Evans, D. 2017 Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies* **2017** (4), 345–364.

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

Goldreich, O., Goldwasser, S. & Halevi, S. 1997 Public-key cryptosystems from lattice reduction problems. In *Annual International Cryptology Conference*, pp. 112–131. Springer.

Hilbert, D. 1890 Ueber die theorie der algebraischen formen. *Mathematische annalen* **36** (4), 473–534.

Hoare, C. A. R. 1961 Algorithm 64: quicksort. *Communications of the ACM* **4** (7), 321.

Hoffstein, J., Pipher, J., Silverman, J. H. & Silverman, J. H. 2008 *An introduction to mathematical cryptography*, , vol. 1. Springer.

Joseph, D., Ghionis, A., Ling, C. & Mintert, F. 2020 Not-so-adiabatic quantum computation for the shortest vector problem. *Physical Review Research* **2** (1), 013361.

Kahrobaei, D., Lam, H. T. & Shpilrain, V. 2019 System and method for private-key fully homomorphic encryption and private search between rings. US Patent 10,396,976.

Kang, S.-J., Lee, I.-S., Lee, K.-H. & Oh, H. 2002 Hecke algebras, specht modules and gröbner–shirshov bases. *Journal of Algebra* **252** (2), 258–292.

Kapovich, I., Myasnikov, A., Schupp, P. & Shpilrain, V. 2003 Generic-case complexity, decision problems in group theory, and random walks. *Journal of Algebra* **264** (2), 665–694.

Kazhdan, D. & Lusztig, G. 1979 Representations of coxeter groups and hecke algebras. *Inventiones mathematicae* **53** (2), 165–184.

Ko, K. H., Lee, S. J., Cheon, J. H., Han, J. W., Kang, J.-s. & Park, C. 2000 New public-key cryptosystem using braid groups. In *Annual International Cryptology Conference*, pp. 166–183. Springer.

Kuck, D. J. 1977 A survey of parallel machine organization and programming. *ACM Computing Surveys (CSUR)* **9** (1), 29–59.

Kulkarni, G., Chavan, N., Chandorkar, R., Waghmare, R. & Palwe, R. 2012 Cloud security challenges. In *2012 7th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, pp. 88–91. IEEE.

Lyubashevsky, V., Peikert, C. & Regev, O. 2010 On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 1–23. Springer.

Manger, J. 2001 A chosen ciphertext attack on rsa optimal asymmetric encryption padding (oaep) as standardized in pkcs# 1 v2. 0. In *Advances in CryptologyCRYPTO 2001*, pp. 230–238. Springer.

McCurley, K. S. 1990 The discrete logarithm problem. In *Proc. of Symp. in Applied Math*, , vol. 42, pp. 49–74.

Micciancio, D. 2001 Improving lattice based cryptosystems using the hermite normal form. In *Cryptography and lattices*, pp. 126–145. Springer.

Micciancio, D. & Warinschi, B. 2001 A linear space algorithm for computing the hermite normal form. In *Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pp. 231–236.

Mora, F. 1985 Gröbner bases for non-commutative polynomial rings. In *International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pp. 353–362. Springer.

Myasnikov, A. D. & Ushakov, A. 2007 Length based attack and braid groups: cryptanalysis of anshel-anshel-goldfeld key exchange protocol. In *International Workshop on Public Key Cryptography*, pp. 76–88. Springer.

Naehrig, M., Lauter, K. & Vaikuntanathan, V. 2011 Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pp. 113–124. ACM.

Nielsen, M. A. & Chuang, I. L. 2010 *Quantum computation and quantum information*. Cambridge university press.

Ostrovsky, R. & Skeith, W. E. 2008 Communication complexity in algebraic two-party protocols. In *Annual International Cryptology Conference*, pp. 379–396. Springer.

Pecen, M. *et al.* 2014 Quantum safe cryptography and security: An introduction, benefits, enablers and challenges, white paper. *European Telecommunications Standards Institute* .

van de Pol, J. 2011 Lattice-based cryptography .

Rai, T. S. 2004 Infinite gröbner bases and noncommutative polly cracker cryptosystems. PhD thesis, Virginia Tech.

Rivest, R. L., Adleman, L., Dertouzos, M. L. *et al.* 1978 On data banks and privacy homomorphisms .

Schleimer, S. & Wiest, B. 2019 Garside theory and subsurfaces: Some examples in braid groups. *Groups Complexity Cryptology* **11** (2), 61–75.

Shirshov, A. 1999 Some algorithmic problem for lie algebras, sibirsk. mat. z. 3 (1962) 292–296. *English translation in SIGSAM Bull* **33** (2), 3–6.

Shor, P. W. 1994 Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pp. 124–134. IEEE.

Shor, P. W. 1995 Scheme for reducing decoherence in quantum computer memory. *Physical review A* **52** (4), R2493.

Shor, P. W. & Preskill, J. 2000 Simple proof of security of the bb84 quantum key distribution protocol. *Physical review letters* **85** (2), 441.

Shpilrain, V. & Ushakov, A. 2006 The conjugacy search problem in public key cryptography: unnecessary and insufficient. *Applicable Algebra in Engineering, Communication and Computing* **17** (3-4), 285–289.

Silverman, J. H. 2006 An introduction to the theory of lattices and applications to cryptography. *Computational Number Theory and Applications to Cryptography, University of Wyoming, Jun* .

Spitzer, F. 2013 *Principles of random walk*, , vol. 34. Springer Science & Business Media.

Stinson, D. R. 2005 *Cryptography: theory and practice*. CRC press.

Takahashi, Y., Tani, S. & Kunihiro, N. 2009 Quantum addition circuits and unbounded fan-out. *arXiv preprint arXiv:0910.2530* .

Van Dijk, M., Gentry, C., Halevi, S. & Vaikuntanathan, V. 2010 Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 24–43. Springer.

Vasco, M. I. G. & Steinwandt, R. 2015 *Group theoretic cryptography*, , vol. 9. CRC Press.

Watson, P. 2012 A multi-level security model for partitioning workflows over federated clouds. *Journal of Cloud Computing: Advances, Systems and Applications* **1** (1), 15.

Wecker, D. & Svore, K. M. 2014 Liqui—¿: A software design architecture and domain-specific language for quantum computing. *arXiv preprint arXiv:1402.4467* .

Wootters, W. K. & Zurek, W. H. 1982 A single quantum cannot be cloned. *Nature* **299** (5886), 802–803.

Xiu, X. 2012 Non-commutative gröbner bases and applications. PhD thesis, Universitätsbibliothek der Universität Passau.

Yi, X., Paulet, R. & Bertino, E. 2014 *Homomorphic Encryption and Applications*. Springer.