Project Code: WJM5110

A Mathematical Approach to Fully Homomorphic Encryption

A Major Qualifying Project Report

Submitted to The Faculty

of the

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science in Mathematical Science

by

Rebecca Meissen

Approved:

Dr. William J. Martin

Abstract

Encryption is used to protect data against eavesdroppers who would otherwise intercept private communication. One party encrypts a message and sends the corresponding ciphertext to a second party, who then decrypts the ciphertext to recover the message. To prevent an untrusted third party from eavesdropping, the problem of recovering any information about the message from the ciphertext should be reasonably hard; in addition, the ciphertext should itself reveal no information about the message. Increasingly, data storage and computation is outsourced to these untrusted parties, which gives rise to the need for an encryption scheme that allows computation on the ciphertexts.

The homomorphic properties of various encryption schemes have been a fascination of the cryptographic community for decades. With the rise of cloud computing and decentralized processing, the need for security in such applications is increasing. Only recently, however, has the construction of a fully homomorphic encryption scheme been realized. I present a mathematical approach to Craig Gentry's proposed fully homomorphic scheme. I start with an overview of other homomorphic encryption schemes, followed by an examination of polynomial rings and their relation to lattices. Finally, I explore the scheme itself and provide a foundation from which to understand the challenges faced when constructing a fully homomorphic encryption scheme.

Contents

1 Introduction

Imagine a prospective employer has an algorithm that calculates your suitability for a job using only your bank account information. For example,

$$X_{\textit{suitability}} = \min\{C + D, E\}$$

where C may be your credit score, D may be the amount of debt you owe, and E may be your total annual expenses.

You would like to know your chances of landing that corporate office, but you don't feel comfortable giving your prospective employer access to your sensitive financial data. It's not that you don't trust him, you simply can't afford to risk *anyone* seeing some of the more embarrassing bits of your financial history!

Now imagine that you could somehow distort the data so that your prospective employer could work with it without being able to read it. This is the idea behind fully homomorphic encryption. In this context, we would call your employer *honest, but curious.* You don't expect them to actively try to undermine you, but you might expect them to peek at the data every once in a while. With fully homomorphic encryption, you can give them access to an encrypted version of your information to work with. The magic happens when they give you an encrypted version of your empirically determined suitability—and you can decrypt it!

Say you have the above algorithm to work with. Then a compatible encryption scheme might look like:

$$\texttt{Encrypt}(x) = 7 \cdot x$$
$$\texttt{Decrypt}(y) = \frac{1}{7} \cdot y$$

That way, when your employer sits down to calculate your suitability, and you give him your encrypted information, what he really calculates is the following.

$$Y_{\textit{suitability}} = \min\{\texttt{Encrypt}(C) + \texttt{Encrypt}(D), \texttt{Encrypt}(E)\} = \min\{7C + 7D, 7E\}$$

But we can factor the 7 out of our sum $7C + 7D = 7(C + D)$, and so the above is also equivalent to taking $\min\{7(C + D), 7E\}$. Also notice that multiplying two numbers by 7 will never change which one is larger. That is, if $A > B$, then $7A > 7B$. So the above is *also* equivalent to taking $7 \cdot \min\{C + D, E\}$.

Now $Y_{\textit{suitability}}$ may look like garbage to the one computing this function, but you can easily glean the useful data from the ciphertext. Say for instance that $C + D < E$. Then

$$X_{\textit{suitability}} = \texttt{Decrypt}(Y_{\textit{suitability}}) = \frac{1}{7} \cdot Y_{\textit{suitability}} = \frac{1}{7}(7C + 7D) = C + D$$

You then return this value to your prospective employer, and both of you can see your suitability for the job.

Furthermore, you can even allow them to encrypt their own input to the function by giving them the encryption key, 7. Say for instance they wanted to compute

$$X_{suitability} = \min\{C + D, E, F\}$$

where F is a constant. The encrypted version $\texttt{Encrypt}(F) = 7F$ can then be included to calculate

$$Y_{suitability} = \min\{\texttt{Encrypt}(C) + \texttt{Encrypt}(D), \texttt{Encrypt}(E), \texttt{Encrypt}(F)\} = \min\{7C + 7D, 7E, 7F\}$$

But as long as you retain the decryption key $\frac{1}{7}$, the data will remain encrypted.

This works because our encryption scheme is *additively homomorphic*, which means that we can work with encrypted data and decrypt the result to obtain a sum of the data. (It is also homomorphic with respect to the min and max operators.) With this technique, we can ask other people to compute sums of our data, without ever letting them see the true numbers! The applications are endless — from voting all the way to cloud computing.

According to cryptographers van Dijk and Juels,

> Clients' lack of direct resource control in the cloud prompts concern about the potential for data privacy violations, particularly abuse or leakage of sensitive information by service providers.[vDJ10]

And corporate entities aren't the only ones looking to benefit. One can also imagine a census of hospital patients. Medical data is subject to many levels of legal protection, making it extremely difficult to gain access to. But to give scientists and medical professionals access to it could make possible experiments and analysis that would further our understanding of medicine. For example, with broad access to patient information one could analyze the correlation between possible symptoms of a disease.

Unfortunately, it's not as simple as it sounds. In fact, cryptographers have been working for decades to solve the problem.

The Challenge

The homomorphic properties of various encryption schemes were recognized in 1978 by Rivest, Adleman, and Dertouzos in [RAD78]. They published a paper proposing the use of 'privacy homomorphisms' to not only secure data, but allow it to be used by untrusted parties. Rivest and Adleman, two of the three cryptographers behind the popular RSA encryption scheme, would later go on to found the RSA security firm in 1982.

Figure 1: Rivest and Adleman[Cal12], and Dertouzos[Cal09]

> Consider a small loan company which uses a commercial time-sharing service to store its records. The loan company's 'data bank' obviously contains sensitive information which should be kept private. On the other hand, suppose that the information protection techniques employed by the time sharing service are not considered adequate by the loan company. In particular, the systems programmers would presumably have access to the sensitive information. The loan company therefore decides to encrypt all of its data kept in the data bank and to maintain a policy of only decrypting data at the home office—data will never be decrypted by the time-shared computer.[RAD78]

However, they also realized the limitations placed on the untrusted party. The nature of encryption inherently obscures some features of the plaintext, making certain operations difficult, or downright impossible. But they left their readers with a final challenge:

> In addition, it remains to be seen whether it is possible to have a privacy homomorphism with a large set of operations which is highly secure.

In other words, they proposed the creation of an encryption scheme with nearly unlimited usability of the ciphertext. Such a scheme would be called *fully homomorphic*, and it would be decades before anyone successfully answered the call to arms.

The Response

Over the years, many schemes were proposed that allowed for a combination of addition and multiplication of ciphertexts. These schemes, however, usually only allowed a very limited number, if any, of one of these two operations. To be fully homomorphic, a scheme should allow an unlimited number of both. While some schemes inherently allowed an unlimited number of a single operation type, it was difficult to see a scheme whose structure would inherit the correct properties for both operations to be valid.

> Any computer algorithm — whether it sorts your mail or figures out whether you qualify for a tax deduction — boils down to a series of arithmetic steps. If an encryption scheme allowed any number of additions or multiplications, any computing application would be possible without decrypting data.[Gre09]

> "It's like one of those boxes with the gloves that are used to handle toxic chemicals," says Gentry. "All the manipulation happens inside the box, and the chemicals are never exposed to the outside world."[Gre09]

Figure 2: Craig Gentry[Gre09]

In the summer of 2008, a Ph.D. candidate at Stanford University studying under Dan Boneh (of the Boneh-Goh-Nissim encryption scheme) was sitting in a cafe in New York City. His name was Craig Gentry, and his revelation that day sparked an alternative approach to fully homomorphic encryption. Rather than relying on the structure of the encryption scheme, why not simply refresh the ciphertext periodically? That way, a scheme that allows for a limited number of operations can be promoted to one that allows an unlimited number.

> Gentry's insight was to double-encrypt the data in such a way that the errors could be removed, so to speak, in the dark. By periodically unlocking the inner layer of encryption underneath an outer layer of scrambling, the cloud computer would clean up its messes as it went along, without ever seeing the secret data. It took Gentry another 15 minutes to realize that he'd just solved an epic cryptographic problem.[Gre09]

The result was not so simple, in fact, and Gentry's proposed scheme requires the use of a key that grows substantially in length as the number of operations increases. Even with an appropriate key, re-encrypting the data periodically introduces a time delay in an otherwise routine computation. To encrypt and search for a single Google query, for example, would take 1 trillion times longer under this scheme. But despite its shortcomings, the cryptography community is buzzing with excitement.

> "There's a lot of engineering work to be done," [Rivest] says. "But until now we've thought this might not be possible. Now we know it is."[Gre09]

Similar to his advisor Boneh's scheme, Gentry's proposed method first makes use of a scheme that allows a large number of additions and a single multiplication of ciphertexts. His scheme makes use of lattices, similar to those found in nature, but much more complex.

The Aftermath

In the three years following Gentry's breakthrough, others have tried to fix the various caveats of the first fully homomorphic scheme. In 2011, researchers at MIT funded by two internet powerhouses, Google and Citigroup, have come up with a similar system called CryptDB.

> [CryptDB] allows users to send queries to an encrypted set of data and get almost any answer they need from it without ever decrypting the stored information...taking a

> fraction of a second to produce an answer where other systems that perform the same encrypted functions would require thousands of years.[Gre11]

In fact, CryptDB promises an added computation time between 15% and 26% of the original. They achieved this, not by constructing a new scheme, but by piecing together old ones. The data is encrypted under many levels of *varying* encryption schemes. Sometimes the data is decrypted, but the last level of encryption is never removed. It has its drawbacks, however. CryptDB cannot perform square roots, and depending on how many levels the data is decrypted, it can leak information about the original information. Despite these, it is a promising alternative. Butler Lampson, a Microsoft Research fellow, remains optimistic.

> “I dont think well see anyone using Gentry’s solution in our lifetimes,” [Lampson] says. “But its very likely well actually see [CryptDB] applied in practice. I dont see any real barrier.”[Gre11]

2 A Brief History of Homomorphic Encryption

2.1 Definitions

To understand homomorphic encryption, it is essential to understand the algebraic meaning of the phrase. We get the term *homomorphic* from the algebraic term *homomorphism*, which refers to a mapping φ between two groups $(G, \diamond)$ and $(H, *)$ such that

$$\varphi(x \diamond y) = \varphi(x) * \varphi(y)$$

for $x, y \in G$ and $\varphi(x), \varphi(y) \in H$. This notion can then be extended to rings or similar algebraic objects in the same category with multiple operations. In the context of cryptography, we consider our mapping to be

$$\texttt{Encrypt} : \mathcal{P} \to \mathcal{C}$$

where $(\mathcal{P}, +, \cdot)$ is the ring of plaintexts and $(\mathcal{C}, \oplus, \otimes)$ is the ring of ciphertexts. We call `Encrypt` a function, but this is meant in the programming sense of the word: a procedure that takes in inputs and returns a value. More precisely, `Encrypt` is a randomized function. We will see later that `Encrypt` samples randomly from a lattice according to some probability density function. This randomized input determines which of the many possible ciphertexts a plaintext may be mapped to. So we may say that $\texttt{Encrypt} : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$, where $\mathcal{K}$ is our space of randomized inputs. Intuitively, we have

$$\texttt{Encrypt}^{-1} = \texttt{Decrypt} : \mathcal{C} \to \mathcal{P}$$

keeping in mind that `Decrypt` will be many-to-one.

Definition 1. *We say an encryption scheme is* ***homomorphic*** *with respect to an operation $\diamond$ on $\mathcal{P}$ if we have*

$$\texttt{Decrypt}(\texttt{Encrypt}(m_1) * \texttt{Encrypt}(m_2)) = \texttt{Decrypt}(\texttt{Encrypt}(m_1 \diamond m_2)) = m_1 \diamond m_2$$

for some operation $*$ on $\mathcal{C}$.

A scheme is considered *somewhat* homomorphic if it can properly perform only a limited number of these operations due to an inability to properly decrypt after a certain threshold of noise introduced by the operations. When we refer to a scheme as additively homomorphic then, what we really mean is that we are able to perform an unlimited number of operations with ciphertexts that correspond exactly to an unlimited number of additions with plaintexts. As another example, a multiplicatively homomorphic scheme should allow unlimited operations (corresponding to multiplications of plaintexts) before proper decryption of the ciphertext result. We call a scheme *fully* homomorphic if it can perform an unlimited number of both types of operations.

2.2 Known Additively Homomorphic Schemes

Earlier we described the difference between a *fully* homomorphic scheme and a *somewhat* homomorphic scheme. There are many somewhat homomorphic schemes already in existence. They are distinguished by the operation which they can evaluate homomorphically, particularly addition or multiplication. An additively homomorphic scheme is one with a ciphertext operation that results in the sum of the plaintexts. That is,

$$\texttt{Encrypt}(m_1) \diamond \texttt{Encrypt}(m_2) = \texttt{Encrypt}(m_1 + m_2)$$

where the decryption of both sides yields the sum of the plaintexts.

There are many known additive schemes. Below we cover the additive variant of ElGamal encryption, but others include the Goldwasser-Micali, Benaloh, and Paillier schemes, which compute addition modulo some number q. The Boneh-Goh-Nissim scheme also allows for unlimited additions, and a single multiplication.[bgn]

2.2.1 ElGamal

One example of an additively homomorphic scheme is the additive variant of the ElGamal public key encryption scheme. The scheme described below from [PP10] allows the encrypter to send only one message (as opposed to the naive Diffie-Hellman protocol, which has her send two.) Also note that while this scheme is illustrative for our purposes, efficient decryption requires a nontrivial explanation and is therefore omitted.

To begin, Bob chooses a random group element β, and sends it to Alice along with the order p and generator α of the group. He keeps secret the parameter a, where $\beta = \alpha^a$. Using the public parameters, Alice computes both the ephemeral key $x = \alpha^k \bmod p$ and the masking key $\beta^k \bmod p$. She uses the masking key to encrypt $y = \alpha^m \cdot \beta^k \bmod p$, and sends along the ephemeral key for Bob to have an advantage when trying to decrypt her message. Her ciphertext is then $c = (x, y)$. Bob then computes $x^{-a} \cdot y = \alpha^m \bmod p$. He then must perform a brute force search to recover the message m.

Regarding decryption, it may be helpful to note that correctness is achieved by the below expansion, where recovering m from α^m is assumed to be efficiently implementable.

$$x^{-a} \cdot y \bmod p = (\alpha^k)^{-a} \cdot (\alpha^m \cdot \beta^k) = (\alpha^k)^{-a} \cdot (\alpha^m \cdot \alpha^{ak}) = \alpha^{-ak+m+ak} = \alpha^m$$

Homomorphism Given two plaintexts m_1 and m_2 and two corresponding ciphertexts $c_1 = \texttt{Encrypt}(m_1) = (x_1,\ y_1)$ and $c_2 = \texttt{Encrypt}(m_1) = (x_2,\ y_2)$ we can compute

$$\begin{aligned}(x_1 \cdot x_2,\ y_1 \cdot y_2) &= (\alpha^{k_1} \cdot \alpha^{k_2} \bmod p,\ \alpha^{m_1} \cdot \beta^{k_1} \cdot \alpha^{m_2} \cdot \beta^{k_2} \bmod p) \\ &= (\alpha^{k_1+k_2} \bmod p,\ \alpha^{m_1+m_2} \cdot \beta^{k_1+k_2} \bmod p) \\ &= \texttt{Encrypt}(m_1 + m_2)\end{aligned}$$

Thus, we can obtain an encryption of the sum of the plaintexts by computing the piecewise product of the ciphertexts.

A formal algorithm is given in Section 2.4, along with an example that makes use of the homomorphic properties of the scheme.

2.3 Known Multiplicatively Homomorphic Schemes

History lends us many homomorphic schemes, and we revisit them to gain a better understanding of the mathematical structure needed to construct a fully homomorphic scheme. Continuing our description of somewhat homomorphic schemes, we move on from additive to multiplicatively homomorphic schemes. A multiplicatively homomorphic scheme is one that has an operation on two ciphertexts that results in the product of the plaintexts. That is,

$$\texttt{Encrypt}(m_1) \diamond \texttt{Encrypt}(m_2) = \texttt{Encrypt}(m_1 \cdot m_2)$$

where the decryption of both sides yields the product of the plaintexts.

The most famous multiplicatively homomorphic scheme is RSA encryption. The original El-Gamal scheme is also multiplicatively homomorphic.

2.3.1 RSA

In 1978, Ron Rivest, Adi Shamir and Leonard Adleman created a public key encryption scheme called RSA. In the preliminary phase, Bob chooses two large primes p and q and computes $n = p \cdot q$. For the next step, he needs to compute Euler's totient function $\phi(n)$, which counts the number of positive integers less than n that are relatively prime to n. For instance, if $n = 6$, the integers found by $\phi(6)$ are $\{1, 5\}$. If n is prime, $\phi(n) = n - 1$. In addition, if n is composed of relatively prime factors, as in our case, $\phi(n) = \phi(p) \cdot \phi(q) = (p - 1) \cdot (q - 1)$.

To select his public and private key, Bob chooses two integers a and b such that $b = a^{-1} \bmod \phi(n)$. (The fact that $x^{\phi(n)} \equiv 1 \bmod n$ will come in handy for decryption.) The smaller of these two becomes the public key $k_{pub} = a$, and the other becomes the private key $k_{pr} = b$. He publishes n and k_{pub}, but keeps p,q, and k_{pr} hidden.

Alice can then encrypt a message m by computing $c = m^a \bmod n$. To decrypt, Bob calculates $c^b \bmod n$. We can see that decryption works, because

$$c^b = (m^a)^b \bmod n = m^{a \cdot a^{-1}} \bmod n = m \bmod n$$

Homomorphism Furthermore, given two plaintexts m_1 and m_2 and two corresponding ciphertexts $c_1 = \texttt{Encrypt}(m_1)$ and $c_2 = \texttt{Encrypt}(m_1)$ we can compute

$$c_1 \cdot\ c_2 \quad = \quad m_1^a \cdot\ m_2^a \bmod n = (m_1 \cdot\ m_2)^a \bmod n = \texttt{Encrypt}(m_1 \cdot\ m_2)$$

This yields an encryption of the product of our original plaintexts. Thus, the RSA scheme is multiplicatively homomorphic.

This scheme is also outlined in Section 2.4. The homomorphic property is showcased in an example at the end of the section.

2.4 Algorithms and Examples

This section is comprised of algorithms and examples intended to supplement Sections 2.2.1 and 2.3.1. Below are algorithms detailing key generation, encryption, and decryption for the additive ElGamal encryption scheme as well as the unpadded RSA encryption scheme.

We conclude with two examples: First, a spaceman tells a computer to count the votes deciding its own demise using our additively homomorphic scheme. Second, a client outsources his computation to a remote server using our multiplicatively homomorphic scheme.

ElGamal Encryption Scheme

Algorithm 1 Additive ElGamal Encryption

```
Output: public key $k_{pub}$ and private key $k_{pr}$
1: function KeyGen
2:     Choose a large prime $p$
3:     Choose a primitive element $\alpha \in \mathbb{Z}_p^*$
4:     Choose an integer $a \in \{0, \dots, p-2\}$
5:     $\beta = \alpha^a$
6:     return $k_{pub} = (p, \alpha, \beta), k_{pr} = a$
7: end function

Input: public key $k_{pub} = (p, \alpha, \beta)$ and message $m$
Output: ciphertext $c$
1: function Encrypt($m$)
2:     Choose $k \in \{2, \dots, p-2\}$
3:     $x = \alpha^k \bmod p$
4:     $y = \alpha^m \cdot \beta^k \bmod p$
5:     return $c = (x, y)$
6: end function

Input: private key $k_{pr} = a$ and ciphertext $c = (x, y)$
Output: message $m$
1: function Decrypt($c$)
2:     $m^* = x^{-a} \cdot y \bmod p$
3:     Recover $m$ from $m^* = \alpha^m$
4:     return m
5: end function
```

RSA Encryption Scheme

Algorithm 2 Multiplicative RSA Encryption

```
Output: public key $k_{pub}$ and private key $k_{pr}$
1: function KeyGen
2:     Choose two large primes $p$ and $q$
3:     $n = p \cdot q$
4:     $\phi(n) = (p-1) \cdot (q-1)$          ▷ Found using Euler's totient function
5:     Choose an integer $a \in \{2, \dots, \phi(n)-1\}$
6:     Find $b = a^{-1} \pmod{\phi(n)}$
7:     return $k_{pub} = (n, a), k_{pr} = b$
8: end function

Input: public key $k_{pub} = (n, a)$ and message $m$
Output: ciphertext $c$
1: function Encrypt($m$)
2:     $c = m^a \bmod n$
3:     return $c$
4: end function

Input: private key $k_{pr} = b$ and ciphertext $c$
Output: message $m$
1: function Decrypt($c$)
2:     $m = c^b \bmod n$
3:     return $m$
4: end function
```

ElGamal Example: Dave and HAL

Example 1. *To see this in action, we turn to Arthur C. Clarke's 2001, a Space Odyssey. Dave is collecting votes from his crew, and would like HAL to tally the computationally expensive sum. The result of the vote will determine whether or not to ram their ship into the nearest gravitational body, destroying HAL in the process. Dave, therefore, doesn't want HAL to know the true outcome of the vote. He sends her encrypted versions of two sub-tallies for HAL to add.*

Dave		*HAL*
Selects a large prime $p = 13$		
Chooses the primitive element $\alpha = 2$ *in* $\mathbb{Z}_{13}^*$		
Chooses $a = 4$ *from* $\{0,\ldots,11\}$		
Computes $\beta = 2^4 \bmod 13 = 3 \bmod 13$		
Gather the votes as plaintexts $m_1 = 5$ *and* $m_2 = 7$		
Choose corresponding random integers $k_1 = 6$ *and* $k_2 = 8$		
Compute $x_1 = 2^6 \bmod 13$ $= 12 \bmod 13$		
Compute $y_1 = 2^5 \cdot 3^6 \bmod 13$ $= 6 \bmod 13$		
Compute $x_2 = 2^8 \bmod 13$ $= 9 \bmod 13$		
Compute $y_2 = 2^7 \cdot 3^8 \bmod 13$ $= 8 \bmod 13$		
Send $c_1 = (12, 6),\ c_2 = (9, 8)$	$\longrightarrow$	
		Compute $x_3 = x_1 \cdot x_2$ $= 12 \cdot\ 9 \bmod 13 = 4 \bmod 13$
		Compute $y_3 = y_1 \cdot y_2$ $= 9 \cdot\ 8 \bmod 13 = 9 \bmod 13$
	$\longleftarrow$	*Send* $c_3 = (4, 9)$
Decrypt partially $x_3^{-a} \cdot\ y_3$ $= 4^{-4} \cdot 9 \bmod 13 = (4^4)^{-1} \cdot 9 \bmod 13$		

$= 9^{-1} \cdot\ 9 \bmod 13 = 1 \bmod 13$

Solve $\alpha^m = 1 \bmod 13$ *for* m

While this last step requires an efficient implementation, we can verify that $\alpha^{m_1+m_2} = 1 \pmod{13}$ *by*

$$\begin{aligned} 2^{5+7} \bmod 13 &= 2^{12} \bmod 13 = (2^4)^3 \bmod 13 \\ &= 3^3 \bmod 13 = 27 \bmod 13 \\ &\equiv 1 \pmod{13} \end{aligned}$$

Once Dave can recover m from the above partial decryption, he will have the desired plaintext tally of votes. Note that while most of the above computation seems to fall to Dave, the encryption of data is merely an overhead cost of the scheme. It may be performed once to allow for endless large sums to be computed by HAL.

RSA Example: Clyde and Sergei

Example 2. *To illustrate the utility of RSA's multiplicative property, we construct a more generic scenario than in Example 1. Instead, consider a client, Clyde, who wishes to pay a Computational Service Provider, Sergei, to compute the product of his data.*

Clyde	*Sergei*
Selects two large primes $p = 3$ *and* $q = 7$	
Computes $n = p \cdot\ q = 21$	
Computes $\phi(n) = (p-1) \cdot\ (q-1) = 2 \cdot\ 6 = 12$	
Chooses $a = 3$ *from* $\{2, \ldots, 7\}$	
Compute $b = a^{-1} \bmod \phi(n)$ $= 3^{-1} \bmod 12 = 8 \bmod 12$	
Gather the data $m_1 = 4$ *and* $m_2 = 6$	
Encrypt $c_1 = m_1^a \bmod n$ $= 4^3 \bmod 21 = 1 \bmod 21$	
Encrypt $c_2 = m_2^a \bmod n$ $= 6^3 \bmod 21 = 6 \bmod 21$	

Send $c_1 = 1,\ c_2 = 6$ $\longrightarrow$

Compute $c_3 = c_1 \cdot\ c_2$

$= 1 \cdot\ 6 \bmod 21 = 6 \pmod{15}$

$\longleftarrow$ *Send* $c_3 = 6$

Decrypt $c_3^b \bmod n = 6^8 \bmod 21$

$= 15 \bmod 21$

Clyde recovers the plaintext answer to his query by the standard decryption algorithm.

3 Basic Ring Theory

Our ultimate goal is to understand how and why the encryption scheme outlined in Section 6 is indeed fully homomorphic. We will show that one can go between logic and ring operations in order to describe certain functions. To identify the specific homomorphic properties of the encryption scheme, we then examine the structure of polynomial rings. Our goal is to provide a framework to understand the properties and boundaries of encryption and decryption. Once we have a standard language with which to describe our operations, we can take a closer look at the scheme itself. For the following definitions, theorems, and examples, we assume our ring R is commutative, nontrivial and has unity 1. Furthermore, for our formulations below we assume that our ring R is of characteristic greater than 2.

3.1 From Circuits to Rings

To understand our scheme, we first must decide what language to use to describe it. In Section 6 our scheme is described in terms of algebraic operations. If one prefers to work strictly in Boolean logic, a conversion to the standard logical operators may be used. However, as our functions grow in complexity, some may find it simpler to work with plaintexts and ciphertexts as ring elements. We show below that one may work with ring operations without loss of functionality. The translation from logic to ring algebra is given for the basic logic gates. We then show that the universality of the `NAND` gate is preserved in the algebraic translation.

To get full functionality of a digital circuit, we first need to verify that binary operations can be expressed in terms of basic ring operations $+$ and $\cdot$.

We start with some well-known Boolean logic. First, the `NOT` operation on only one variable $X \in \{0, 1\}$. For these values of X, we can compute `NOT` X with $1 - X$.

X	`NOT` X	$1 - X$
0	1	1
1	0	0

Given two inputs $X,\ Y \in \{0, 1\}$, the operation X `AND` Y can be calculated with the product $X \cdot Y$.

X	Y	X `AND` Y	$X \cdot Y$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

We can similarly compute X `OR` Y using $X + Y - X \cdot Y$ This subtraction in the ring can be alternately stated as $(-1)X \cdot Y$, where -1 is the additive inverse guaranteed in R.

X	Y	X `OR` Y	$X+Y-X\cdot Y$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

It is also helpful to verify that the `XOR` and `NAND` operations can be expressed. For X `XOR` Y, often abbreviated $X \oplus Y$, we compute $X+Y-2X\cdot Y$ or rather $X+Y-X\cdot Y-X\cdot Y$. Lastly, X `NAND` Y is simply $1-X\cdot Y$.

X	Y	X `XOR` Y	$X+Y-2X\cdot Y$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

X	Y	X `NAND` Y	$1-X\cdot Y$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Using certain subsets of these binary operations (for example, `AND`, `OR`, and `NOT`) we can express any computation on our binary inputs. This universal computation can also be achieved with a subset containing a single operation, `NAND`. This is useful to note because it allows one to consider a simpler analysis in terms of logical quantities.

In particular, any of the above operations can be constructed using varying combinations of `NAND` operations. Given a single input $X \in \{0,1\}$, we can compute `NOT` X using only `NAND`s as X `NAND` X. Using the above translations to ring operations, this amounts to $1-X\cdot X = 1-X^2$, which since $X \in \{0,1\}$ is equivalent to $1-X$.

Similarly, we can construct the equivalent of an `AND` operation using only `NAND`s as

$$X \texttt{ AND } Y = (X \texttt{ NAND } Y) \texttt{ NAND } (X \texttt{ NAND } Y)$$

Again we verify the ring operative equivalent.

$$X \texttt{ AND } Y = 1-(1-XY)(1-XY) = 1-(1-2XY+X^2Y^2) = -2XY+XY = XY$$

The equivalent of the `OR` operator can be made with

$$X \texttt{ OR } Y = (X \texttt{ NAND } X) \texttt{ NAND } (X \texttt{ NAND } X)$$

Using our expressions in terms of the ring operations, we have

$$1-(1-X^2)(1-Y^2) = 1-(1-X^2-Y^2+X^2Y^2) = X^2+Y^2-X^2Y^2 = X+Y-XY$$

Finally, we construct the equivalent of an `XOR` operation using only `NAND` operations by

$$X \texttt{ XOR } Y = ((X \texttt{ NAND } X) \texttt{ NAND } Y) \texttt{ NAND } (X \texttt{ NAND } (Y \texttt{ NAND } Y))$$

We verify this algebraically with

$$
\begin{aligned}
&1-(1-(1-X^2)(Y))(1-(X)(1-Y^2))\\
&=1-[1-X(1-Y^2)-Y(1-X^2)+XY(1-Y^2)(1-X^2)]\\
&=1-[1-X+XY^2-Y+X^2Y+XY(1-X^2-Y^2+X^2Y^2)]\\
&=1-[1-X+XY^2-Y+X^2Y+XY-X^3Y-XY^3+X^3Y^3]\\
&=1-[1-X+XY-Y+XY+XY-XY-XY+XY]\\
&=1-[1-X-Y+4XY-2XY]\\
&=1-[1-X-Y+2XY]\\
&=X+Y-2XY
\end{aligned}
$$

In our proceeding analysis, we will mainly consider the algebraic operations $+$ and $\cdot$ with regards to polynomials. One could translate these to their logical equivalents if they do not wish to continue in algebraic notation.

3.2 Definitions

In Section 3.1 we showed that one may use ring operations to describe logical functions. Continuing this notion, [Gal10] provides some algebraic definitions that we will later make use of.

By using ideals, we obtain not only an additive, but multiplicative structure for our ciphertexts. In short, there are two properties that we wish to obtain. For a subring $I \subseteq R$, and two elements $a, b \in R$,

$$
\begin{aligned}
(a+I)+(b+I) &= a+b+I\\
(a+I)\cdot(b+I) &= a\cdot b+I
\end{aligned}
$$

A two-sided ideal suits these needs. We proceed with the formal definition followed by a small example.

Definition 2. *A subring I of a ring R is* ***ideal*** *(or two-sided ideal) in R if for all $r \in R$ and for all $i \in I$, $ir \in I$ and $ri \in I$.*

Example 3. *A simple example of this is the subring $I = 7\mathbb{Z} = \{\dots, -14, -7, 0, 7, 14, \dots\}$ of the integers $R = \mathbb{Z}$. Note that for any element $i \in 7\mathbb{Z}$, we know that $i = 7j$ for some $j \in \mathbb{Z}$. So for any $r \in \mathbb{Z}$, $ir = 7jr \in 7\mathbb{Z}$. Similarly because $\mathbb{Z}$ is commutative under multiplication, $ri \in 7\mathbb{Z}$. Thus, $7\mathbb{Z}$ is an ideal subring of $\mathbb{Z}$ (or I is an ideal subring of R.)*

Definition 3. *Two ideals I and J of a ring R are* ***comaximal*** *if $I + J = R$. That is, for any element $r \in R$, $r = i + j$, for some $i \in I$ and $j \in J$. If R contains a multiplicative identity 1_R, then this definition is equivalent to saying that $1_R = i + j$ for some $i \in I$ and $j \in J$.*

Example 4. *Using the ideal subring from above, note that $J = 5\mathbb{Z}$ is comaximal to $I = 7\mathbb{Z}$ in $R = \mathbb{Z}$ because $15 - 14 = 1$, where $15 \in 5\mathbb{Z}$ and $-14 \in 7\mathbb{Z}$.*

3.3 Necessary Theorems

In Section 3.1 we explored the motivation for using algebraic structures to represent logical quantities. Now that we have a basic idea of what these algebraic structures look like, we can continue to modify the form of these ideals to suit our needs. We will need a few theorems from [Gal10] to help our construction along.

Our goal will be to take ideals from $\mathbb{Z}[x]$ and restrict the elements to a manageable size. This is done by using a principle ideal $(f(x))$ generated by a single polynomial $f(x)$.

Theorem 1 (First Isomorphism Theorem for Rings)**.** *If $\varphi : R \to S$ is a ring homomorphism, then the kernel (denoted $\ker \varphi$) is an ideal of R, the image (denoted $\varphi(R)$) is a subring of S, and $R/\ker\varphi$ is isomorphic to $\varphi(R)$.*

Theorem 2 (Third Isomorphism Theorem for Rings)**.** *Let I and J be ideals of R with $I \subseteq J$. Then J/I is an ideal of R/I and $(R/I)/(J/I)$ is isomorphic to R/J.*

Theorem 3 (Fourth Isomorphism Theorem for Rings)**.** *Let I be an ideal of R. The correspondence given by*

$$A \longleftrightarrow A/I$$

is an inclusion preserving bijection between the set of subrings A of R that contain I and the set of subrings of R/I. Furthermore, A (a subring containing I) is an ideal of R if and only if A/I is an ideal of R/I.

As a result of these, we can take ideals I and J from $\mathbb{Z}[x]$ and restrict them to their corresponding ideals in $\mathbb{Z}[x]/(f(x))$. The ring $\mathbb{Z}[x]/(f(x))$ is often referred to as a *truncated polynomial ring* (typically with $f(x) = x^n - 1$).

3.4 Examples

We consider examples below of a very specific kind. We begin with ideals that are generated by a pair $(p, g(x))$, where $p \in \mathbb{Z}$ and $g(x) \in \mathbb{Z}[x]$. Using Theorem 3, we then find the corresponding ideals in $\mathbb{Z}[x]/(f(x))$ for a polynomial $f(x)$ in both ideals.

We first want to find two comaximal ideals of R, I and J, such that $I + J = R$. If R has unity, we can verify this property by finding $x \in I, y \in J$ such that $x + y = 1 \in R$. We can then restrict these ideals by a shared element. The procedure is as follows:

1. Pick the ideals I and J.
    - Choose the generators $p_1,\ g_1(x);\ p_2,\ g_2(x)$.
    - Describe I and J in terms of restrictors.

2. Choose $f(x) \in I + J$, such that
   - $f(x) = h_1(x) \cdot p_1 + l_1(x) \cdot g_1(x) + h_2(x) \cdot p_2 + l_2(x) \cdot g_2(x)$
3. Transform I and J into $\hat{I} = I/(f(x))$ and $\hat{J} = J/(f(x))$.
4. Check for comaximality.
   - Solve for $x \in \hat{I}, y \in \hat{J}$ such that $x + y = 1$.

From now on we consider the following convention, where $I = (p, g(x))$ and $J = (q, h(x))$ are ideals of the ring $R = \mathbb{Z}[x]$, which we note is commutative and contains the multiplicative identity $1_R = 1$. We first proceed with examples using only steps 1 and 4 by construct comaximal ideals in R. Next, we use steps 2 and 3 in addition to steps 1 and 4 as outlined above.

Example 5. *Let $I = (p, x - a)$, that is, the ideal formed with generators p and $g(x) = x - a$, where $p, a \in \mathbb{Z}$. Then*

$$I = \{c(x) = c_m x_m + c_{m-1}x^{m-1} + \cdots + c_1 x + c_0 \mid c(x) = ir;\ i \in I,\ r \in R\}$$

So for any element $c(x) \in I$, $c(x) = s(x) \cdot p + t(x) \cdot (x - a)$. Note then that

$$c(a) = s(x) \cdot p + t(x) \cdot (a - a) = s(x) \cdot p + 0 = s(x) \cdot p$$

so $c(a) \equiv 0 \pmod p$. Thus, every element $c(x) \in I$ has the property that $c(a) \equiv 0 \pmod p$).

Now instead take any element of the form $c(x) = c_m x_m + c_{m-1}x^{m-1} + \cdots + c_1 x + c_0$ with the property that $c(a) \equiv 0 \pmod p$ for some $p \in \mathbb{Z}$. By the Chinese Remainder Theorem, $c(a) = k \cdot p$ for $k \in \mathbb{Z}$. So then $c(a) - k \cdot p = 0$ (Note: if this contained another factor of p, simply increase k to $k + 1$.) Then we know that either $p \mid c(a)$ or $c(a) = 0$. Thus, these elements are exactly those from I as described originally.

To examine comaximality, we begin with some simple constructions. It is convenient to note that given $I = (p, g(x))$, the following ideals are comaximal to I:

1. $J = (p \pm 1, h(x))$
2. $J = (q, g(x) \pm 1)$
3. $J = (r, h(x))$, where $gcd(p, r) = 1$

Case 1 ($q = p \pm 1$)**.** *Without loss of generality, assume $q = p + 1$. Then, assuming I and J proper ideals of R, we can take $x = p \in I$ and $y = -q \in J$ such that*

$$x + y = p - q = q + 1 - q = 1$$

making I and J comaximal.

Case 2 ($h(x) = g(x) \pm 1$)**.** *Similarly, if $h(x) = g(x) \pm 1$, we use the same construction as in Case 1 to show that I and J are comaximal. That is,*

$$h(x) \mp g(x) = 1$$

Case 3 ($\gcd p, q = 1$)**.** *Assume p and q are relatively prime. Then $\gcd(p, q) = 1$, so by the Extended Euclidean Algorithm there exist $s, t \in \mathbb{Z}$ such that $s \cdot p + t \cdot q = 1$. We define $R = Z[x]$, so $s, t \in R$. Recall that I and J, being ideals of R, absorb external multiplication from R. So $s \cdot p \in I$ and $t \cdot q \in J$. Thus, I and J are comaximal.*

We do not extend Case 3 to $g(x)$ and $h(x)$ relatively prime, because the greatest common divisor of polynomials with integer coefficients is made more difficult with 1 being the only unit of R.

Example 6. *Let $I = (5, x - 1)$ and $J = (4, x + 3)$. That is, based on our previous example, we know that we can express I and J as follows:*

$$I = \{c(x) \mid c(1) \equiv 0 \pmod 5\}$$
$$J = \{c(x) \mid c(-3) \equiv 0 \pmod 4\}$$

Then we construct

$$f(x) = (x - 1) \cdot (x + 3) = x^2 + 2x - 3$$

We take $\hat{I} = I/(f(x))$ and $\hat{J} = J/(f(x))$.

$$\hat{I} = \{ax + b \mid x^2 = -2x + 3; a \equiv 4b \pmod 5\}$$
$$\hat{J} = \{ax + b \mid x^2 = -2x + 3; a \equiv 3b \pmod 4\}$$

To check comaximality, we can simply take $-5 = (-1)5 + (0)(x - 1) \in \hat{I}$ and $6 = (1)6 + (0)(x + 3) \in \hat{J}$. It is clear that $-5 + 6 = 1$, so that $\hat{I}$ and $\hat{J}$ are comaximal.

Example 7. *Let $I = (5, x - 1)$ and $J = (5, 2x + 4)$. Then*

$$I = \{c(x) \mid c(1) \equiv 0 \pmod 5\}$$
$$J = \{c(x) \mid c(-2) \equiv 0 \pmod 5\}$$

We use the simple construction of

$$f(x) = g(x) \cdot h(x) = 2x^2 + 2x - 4$$

giving us our reduction rule: $x^2 = -x + 2$.

$$\hat{I} = \{ax + b \mid a \equiv 4b \pmod 5\}$$
$$\hat{J} = \{ax + b \mid 3a \equiv 4b \pmod 5\}$$

Finally, comaximality is verified by $(-4x + 9) + (4x + 8) = 1$, where $-4x + 9 \in \hat{I}$ and $4x + 8 \in \hat{J}$.

Example 8. *Consider the case where* $g_1(x) = g_2(x) = g(x)$. *For now, let* $I = (2, x+1)$ *and* $J = (5, x+1)$. *Then*

$$I = \{c(x) \mid c(-1) \equiv 0 \pmod 2\}$$
$$J = \{c(x) \mid c(-1) \equiv 0 \pmod 5\}$$

We construct $f(x)$ *similarly as before.*

$$f(x) = g_1(x) \cdot g_2(x) = (g(x)) =^2 x^2 + 2x + 1$$

Thus, our reduction rule is $x^2 = -2x - 1$.

$$\hat{I} = \{ax + b \mid a \equiv b \pmod 2\}$$
$$\hat{J} = \{ax + b \mid a \equiv b \pmod 5\}$$

Comaximality is checked by $-4 + 5 = 1$, *where* $-4 \in \hat{I}$ *and* $5 \in \hat{J}$.

4 Lattices

We saw in Section 3 that we can think of our encrypted objects as polynomials. As we continue, it becomes necessary to introduce the concept of length. The length of a polynomial can be thought of simply as the norm of the coefficient vector. For a given polynomial $p(x) = p_n x^n + \cdots + p_1 x + p_0$, we first form the coefficient vector $\mathbf{p} = [p_0, p_1, \ldots, p_n]$. We then compute the Euclidean norm of $\mathbf{p}$. The result is both the length of the coefficient vector $\mathbf{p}$ and the polynomial $p(x)$. We denote this as follows:

$$\|p(x)\| = \|\mathbf{p}\| = \sqrt{\sum_{i=0}^{n} p_i^2}$$

This notation is presented for the reader's convenience in Appendix A along with other conventions used throughout this paper.

Graphically, these vectors allow us to take a more geometric approach when considering bounds. More specifically, we will consider polynomial rings of limited degree. This is done by restricting the elements of our ideal to smaller order polynomials, which we saw how to do in Section 3.4. Once our polynomials have a maximum of n coefficients, we can think of them as vectors in an n-dimensional space. When these vector components belong to a subring of $S \subseteq R = \mathbb{Z}[x]/(f(x))$, as they do in our case, we may consider them as elements of a lattice.

It should also be noted that from here on we refer to the lattice corresponding to the ideal $I/(f(x))$ as simply I, and similarly the lattice corresponding to $J/(f(x))$ as J.

Formally, a lattice is defined as a discrete additive subgroup of $\mathbb{R}^n$. For our purposes it is useful to include the basis of the lattice in the definition.

Definition 4. *A* ***lattice*** *is the set of all integer linear combinations of a set of linearly independent vectors $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbb{R}^m$,*

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} c_i \mathbf{b}_i \;\middle|\; c_i \in \mathbb{Z},\ \forall i \right\}$$

The collection of vectors $\mathcal{B}$ is called the ***basis*** *of $\mathcal{L}$. When we take B to be a matrix whose columns are the basis vectors from $\mathcal{B}$, we can also denote*

$$\mathcal{L} = \{B\boldsymbol{c} \mid \boldsymbol{c} \in \mathbb{Z}^n\}$$

.

Definition 5. *A set of vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ are* ***linearly independent*** *if $\sum_{i=1}^{n} c_i \mathbf{b}_i = 0$ implies $c_i = 0$ for all i.*

We note that a basis is by definition linearly independent, but question whether a lattice must be thought of as being generated by a set of linearly independent vectors. That is, can a lattice $\mathcal{L}$ be generated from a set of linearly *dependent* vectors?

Lemma 1. *Let $\mathbf{b}_{n+1} = \sum_{i=1}^{n} d_i \mathbf{b}_i$, where $d_i \in \mathbb{Z}\ \forall i$ and at least one $d_i \neq 0$. The lattice generated by $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is equivalent to the lattice generated by $\{\mathbf{b}_1, \ldots, \mathbf{b}_n, \mathbf{b}_{n+1}\}$.*

Proof. Let $\mathcal{L}_1$ be the lattice generated by $B_1 = [\mathbf{b}_1 \cdots \mathbf{b}_n]$, and $\mathcal{L}_2$ be the lattice generated by $B_2 = [\mathbf{b}_1 \cdots \mathbf{b}_n \mathbf{b}_{n+1}]$. We will show that for any lattice point x, $x \in \mathcal{L}_1$ if and only if $x \in \mathcal{L}_2$.

$\Rightarrow$ First consider $x \in \mathcal{L}_1$. We know $x = \sum_{i=1}^{n} c_i \mathbf{b}_i$, where $c_i \in \mathbb{Z}\ \forall i$. So

$$x = \sum_{i=1}^{n} c_i \mathbf{b}_i + 0 \cdot \mathbf{b}_{n+1} = \sum_{i=1}^{n+1} c_i \mathbf{b}_i \qquad \text{where } c_i \in \mathbb{Z}\ \forall i$$

Thus, $x \in \mathcal{L}_2$.

$\Leftarrow$ Now consider $y \in \mathcal{L}_2$. We know

$$x = \sum_{i=1}^{n+1} c_i \mathbf{b}_i = \sum_{i=1}^{n} c_i \mathbf{b}_i + c_{n+1} \mathbf{b}_{n+1} \qquad \text{where } c_i \in \mathbb{Z}\ \forall i$$

We also know that $\mathbf{b}_{n+1} = \sum_{i=1}^{n} d_i \mathbf{b}_i$, where $d_i \in \mathbb{Z}\ \forall i$. Let $a_i = c_i + c_{n+i} d_i$ and note that because $c_i, d_i \in \mathbb{Z}$, a_i is also in $\mathbb{Z}$. Then

$$x = \sum_{i=1}^{n} c_i \mathbf{b}_i + c_{n+1} \sum_{i=1}^{n} d_i \mathbf{b}_i = \sum_{i=1}^{n} a_i \mathbf{b}_i \qquad \text{where } a_i \in \mathbb{Z}\ \forall i$$

Thus, $x \in \mathcal{L}_1$. □

Definition 6. *We say that a basis matrix $B = [\mathbf{b}_1 \cdots \mathbf{b}_n] \in \mathbb{R}^{m \times n}$ has **full rank** if the rank equals the dimension, $m = n$. A lattice is similarly said to be of full rank if its basis vectors form a full rank matrix.*

4.1 Multiple Bases

A lattice of dimension 2 or more can have many different possible bases. What decides whether a given basis is *good* comes down to several criteria concerning the basis vectors. As we will see later in Section 6, proper decryption in Gentry's scheme relies on our ciphertext vectors staying relatively small. For this reason, our criteria for a good basis are centered around this idea as well.

- **Length** The sum or product of short vectors is generally smaller in length than the sum or product of long vectors. Keeping the basis vectors short results in shorter computed vectors at each step of computation.

- **Orthogonality** Operations concerning orthogonal, or nearly orthogonal, vectors result in vectors that lie closer to the origin when compared to operations on similar-length vectors whose inclination from the origin is very close (i.e. not very orthogonal.)

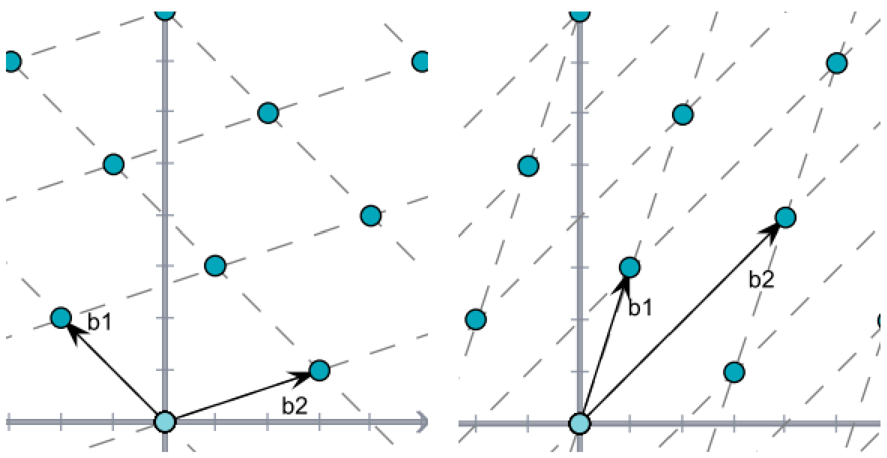

Figure 3: Good (left) and bad (right) bases for the same lattice

We see in Figure 3 an example of both a good and bad basis for the same lattice based on this criteria. Simply put, the best bases will be easy to perform computations with, because the vectors will be relatively small. A bad basis should be considerably harder to work with. Encryption will make use of two bases for a lattice J — one will be kept secret, while the other is made public. Specifically, we keep the good basis secret to prevent attackers from gaining any additional advantages.

4.2 Basis Reduction Algorithms

Given a basis for a lattice, we can sometimes construct a *better* basis with respect to the criteria in Section 4.1. We call the procedures used to construct these bases *basis reduction algorithms.*

One such algorithm is the Lenstra-Lenstra-Lovász (LLL) algorithm. A detailed description can be found in Chapter 2.6 of [Coh93]. Given an $m \times n$ basis B, it produces a new set of orthogonal (but not orthonormal) basis vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_m^*$ such that

$$\|\mathbf{b}_i^* + \mu_{i,i-1}\mathbf{b}_{i-1}^*\|^2 \geq \frac{3}{4}\|\mathbf{b}_{i-1}^*\|^2 \qquad \text{for } 1 < i \leq n$$
$$|\mu_{i,j}| \leq \frac{1}{2} \qquad \text{for } 1 \leq j < i < n$$

in polynomial time, specifically $O(m^5 n \log^3(\max \|\mathbf{b}_i\|))$.

However, given a basis B it is considered NP-hard to find a vector of shortest nonzero length in the lattice it spans. That is, the element of the lattice closest to the origin. This is known as the *Shortest Vector Problem.* More can be found on this topic in [Mic01b]. Although the LLL algorithm will yield a better basis, the shortest member $\mathbf{b}_i^*$ of this basis is not guaranteed to be a shortest vector.

A related problem is the *Closest Vector Problem.* With a basis B and a vector $\mathbf{c}$, it is considered NP-hard to find the closest lattice point. Given B and $\mathbf{v} \in \mathbb{R}^m$, find $\mathbf{c} \in \mathbb{Z}^n$ which will minimize

$\|\mathbf{v} - B\mathbf{c}\|$ over all $\mathbf{c} \in \mathbb{Z}^n$. A simple proof of the hardness of this problem can be found in [Mic01a], which also includes an account of hardness even when preprocessing is allowed. In [Gen09a] and [Gen09b], Gentry another problem as the *Ideal Coset Problem*. It is solved by the Closest Vector Problem.

Sometimes these algorithms produce a basis that is worse than the original. This can be useful to ensure that the public basis is indeed worse than the private basis, and does not provide a potential attacker with any additional information. In particular, the Hermite normal form of a matrix, when computed from a given basis matrix B, operates as a generic basis for which the corresponding lattice is typically useless for problems such as this. We will now present two algorithms that efficiently compute the Hermite normal form of a matrix.

4.2.1 Hermite Normal Form

Similar to the reduced echelon form of a matrix with entries in $\mathbb{R}$, a matrix A with integer entries in Hermite normal form is upper triangular ($a_{ij} = 0$ for $i > j$) and has positive, weighty diagonal entries ($a_{ij} > 0$ for $i = j$ and $0 \leq a_{ij} < a_{ii}$ for $i < j$.)

There are two main ways to compute the Hermite normal form (HNF) of a matrix. The first is similar to the Gram-Schmidt orthogonalization process, whereas the second makes use of the greatest common divisor to reduce computation time. The resulting matrix of this reduction has somewhat large, non-orthogonal columns. If we give an adversary only a basis, even a particularly bad one, they may compute the HNF in polynomial time. Because it is efficiently computable, we waste nothing in making the HNF of our basis public. Furthermore, it tends to yield a particularly useless lattice as a result, which benefits us as well. For this reason, these algorithms may be helpful in producing a basis matrix for the lattice J which is sufficiently bad according to the criteria in Section 4.1.

The algorithms below are taken from [Coh93] and reformulated for readability. In each case, a formal presentation is given. Code written in Sage for each algorithm can be found in Appendix B.

Hermite Normal Form Without GCD

To initialize the algorithm let $i = m$ be the number of rows of A, and let $k = n$ be the number of columns of A. We will count down from $k = n$ to $k = 1$. If A is wide, that is if $m \leq n$, set $l = 1$. Otherwise, if A is tall, set $l = m - n + 1$. We proceed by referring to the columns of A as $\mathbf{a}_i$ denoting the i^{th} column of A.

Pick a row starting from the last row $i = m$ and, moving up, and count down the entries along the columns from the left to right starting with column $k = n$.

Check if all the elements in this (the i^{th}) row to the left of a_{ik} are 0. If there are any nonzero

Algorithm 3 Algorithm 2.4.4 [Coh93]

Input: nonsingular square matrix A with entries $a_{ij} \in \mathbb{Z}$

Output: Hermite normal form of A

```
 1: function HNF($A$)
 2:     while $i \neq 1$ do
 3:         if $a_{ij} = 0$ for all $j < k$ then
 4:             if $a_{ik} < 0$ then
 5:                 $\mathbf{a}_k = -\mathbf{a}_k$
 6:             end if
 7:             if $a_{ik} = 0$ then
 8:                 $k++$
 9:             else for $j < k$
10:                 $\mathbf{a}_j = \mathbf{a}_j - \lfloor (a_{ij}/a_{ik}) \rfloor \mathbf{a}_k$
11:             end if
12:             $i--$
13:             $k--$
14:         else
15:             if $a_{ik} < 0$ then
16:                 $\mathbf{a}_k = -\mathbf{a}_k$
17:                 $a_{min} = \min a_{ij}$ for $j < k$
18:                 $j_{min} = j$
19:                 swap $\mathbf{a}_{j_{min}}$ and $\mathbf{a}_k$
20:                 if $a_{ik} < 0$ then
21:                     $\mathbf{a}_k = -\mathbf{a}_k$
22:                 end if
23:                 $\mathbf{a}_j = \mathbf{a}_j - \lfloor a_{ij}/a_{ik} \rfloor \mathbf{a}_k$
24:             end if
25:         end if
26:     end while
27: end function
```

elements, find the minimum nonzero element in the row, say in the j^{th} column, and exchange columns j and k. If the entry $a_{ik} < 0$, replace $\mathbf{a_k}$ with $-\mathbf{a_k}$.

For all the columns j to the left of k, set $q = \lfloor a_{ij}/a_{ik} \rfloor$, and redefine column $\mathbf{a_j} = \mathbf{a_j} - q\mathbf{a_k}$. So, for each column j to the left of k, subtract a multiple of the k^{th} column to make $a_{ij} = 0$ for all elements in the i^{th} row to the left of column k.

Now check again if all the elements in the i^{th} row to the left of the k^{th} column are 0. If they are not, repeat the above steps.

Once all of the elements to the left of the k^{th} column are 0, set $i = i - 1$ and $k = k - 1$, until $i = l = 1$ or $m - n + 1$, and $k = n - m + 1$ or 1 respectively. Stop when either the rows or columns of A run out. Do the above for the entries along the diagonal, beginning with the last entry in A, a_{mn}.

If A is not square, simply construct a new matrix W to be the transformed nonzero columns of A.

Hermite Normal Form With GCD

Using an efficient implementation of the extended Euclidean algorithm may improve this procedure. This leads to our next algorithm, which uses this notion to eliminate unnecessary iterations.

Algorithm 4 Algorithm 2.4.5

Input: nonsingular square matrix A with entries $a_{ij} \in \mathbb{Z}$

Output: Hermite normal form of A

```
function HNF($A$)
    $k = n$
    $j = m$
    for $i \in \{n, \dots, 1\}$ do
        $d = \gcd(a_{ik}, a_{ij})$
        Find minimal $(s, t)$ such that $sa_{ik} + ta_{ij} = d$ ▷ Use the Extended Euclidean Algorithm
        for $j \in \{m, \dots, 1\}$ do
            if $a_{ij} \neq 0$ then
                $\mathbf{b} = s \cdot \mathbf{a_k} + t \cdot \mathbf{a_j}$
                $\mathbf{a_j} = (a_{ik}/d)\mathbf{a_j} - (a_{ij}/d)\mathbf{a_k}$
                $\mathbf{a_k} = \mathbf{b}$
            end if
        end for
    end for
    for $i \in \{n, \dots, 1\}$ do
        if $a_{ik} < 0$ then
            $\mathbf{a_k} = -\mathbf{a_k}$
        end if
        for $j \in \{k+1, \dots, n\}$ do
            $q = \lfloor a_{ij}/a_{ik} \rfloor$
            $\mathbf{a_j} = \mathbf{a_j} - q\mathbf{a_k}$
        end for
    end for
end function
```

This second algorithm works similarly to the first, with the difference of using an efficiently implemented extended Euclidean algorithm to reduce each row (although we refer only to column vectors within the algorithm.) Rather than step through the reduction at each row and column, it efficiently reduces the entries by the maximum amount the first time using the gcd of its entries.

4.3 Fundamental Region

Now that we are familiar with the bases of a lattice, we can examine another crucial piece of its structure. Associated with a given basis B for a lattice $\mathcal{L}$ is the *fundamental region* formed by the basis vectors. We will make use of this concept when describing the encryption scheme in Section 6.

Definition 7. *Given a basis B for a lattice $\mathcal{L}$, the **fundamental region** associated with B consists of all the points within the box formed by the basis vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$.*

$$\mathcal{F}_B = \left\{ B\boldsymbol{c} \;\middle|\; -\frac{1}{2} \leq c_i < \frac{1}{2} \;\forall i \right\}$$

Notice that $\mathcal{F}_B \not\subset \mathcal{L}$. That is, the fundamental region is not made up of lattice points. In fact, the only point they have in common will the the one at the origin. Also note that although the length of the basis vectors may vary, the volume of the fundamental region will be the same regardless of which basis for a given lattice is used. This volume is referred to as the *determinant* of the lattice and is equal to $|\det(B)|$ for any basis B of the lattice. If a lattice $\mathcal{L}$ is spanned by a basis B, we may refer to $\mathcal{F}_\mathcal{L}$ to mean $\mathcal{F}_B$ if the basis B is easily inferred from the text. In fact, we refer to $\mathcal{F}_J^{sk}$ as the fundamental region formed by the secret basis for lattice J and $\mathcal{F}_J^{pk}$ as the fundamental region formed by the public basis for J.

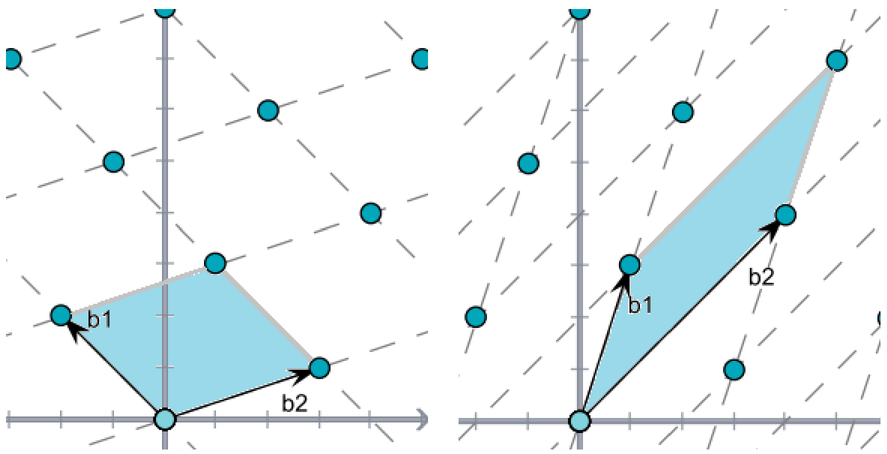

Figure 4: Fundamental regions formed by different bases

We will consider bases with fundamental regions centered at the origin. In addition, we will often refer to the operation $\mathbf{c} \bmod B$ to refer to finding the equivalent point $\mathbf{c}'$ within $\mathcal{F}_B$. A problem arises when one cannot differentiate between two equivalent points $\mathbf{c}' \equiv \mathbf{c} \pmod{B}$, where $\mathbf{c}' \in \mathcal{F}_B$ and $\mathbf{c} \notin \mathcal{F}_B$.

5 Sampling from a Lattice

Recall that encryption is meant to be a randomized function. Given an input π, we will therefore need to provide random input by means of sampling from the coset $\pi + I$ according to a distribution $\mathcal{D}$. This involves choosing an element $i \in I$, but we will see that $\mathcal{D}$ must satisfy the following condition in order for decryption to work:

- The element $\pi + i$ chosen from $\pi + I$ must be unique inside $\pi + J$ with respect to the secret basis B_J^{sk}.

More simply, we prefer to take our random element i such that $\pi + i \in \mathcal{F}_J^{sk}$. We refer to [GPV07] to see just how to sample from a lattice. First we define the continuous Gaussian function over $\mathbb{R}^n$ with center $\mathbf{c} \in \mathbb{R}^n$ and deviation $s \in \mathbb{R}$ as

$$\rho_{s,\mathbf{c}}(\mathbf{x}) = \exp(-\pi \|\mathbf{x} \text{ - } \mathbf{c}\|^2 / s^2)$$

for all $\mathbf{x} \in \mathbb{R}$. We then define the corresponding discrete Gaussian function over a lattice $\mathcal{L}$ as

$$D_{\mathcal{L},s,\mathbf{c}}(\mathbf{x}) = \frac{\rho_{s,\mathbf{c}}(\mathbf{x})}{\rho_{s,\mathbf{c}}(\mathcal{L})}$$

for all $\mathbf{x} \in \mathcal{L}$. The denominator is defined as $\rho_{s,\mathbf{c}}(\mathcal{L}) = \sum_{\mathbf{x}\in\mathcal{L}} \rho_{s,\mathbf{c}}(\mathbf{x})$, and acts as a normalizing factor on the probability of the elements of $\mathcal{L}$, similar to the implied denominator in the continuous function, $\rho_{s,\mathbf{c}}(\mathbb{R}^n) = \int_{-\infty}^{\infty} \rho_{s,\mathbf{c}}(\mathbf{x}) d^n x = 1$.

To sample randomly from a lattice, we first sample randomly from the integers, and incorporate this into our procedure. This method is outlined in [GPV07] and reproduced in part here.

We allow the use of a subroutine to choose a random element from a 1-dimensional integer lattice (i.e. $\mathbb{Z}$.) One common method of sampling according to a probability distribution function uses a pseudorandom number generator (PRNG) to produce a number $\xi \in [0, 1)$. The sample is then the value η that solves the cumulative density function $\xi = \int_{-\infty}^{\eta} \rho_{s,\mathbf{c}}(\mathbf{x}) d^n x$. In the discrete case it is often easier to implement rejection sampling, whereby a value η is chosen uniformly at random from the set $\mathbb{Z} \cap [\mathbf{c} - s \cdot t(n), \mathbf{c} + s \cdot t(n)]$ and accepted with probability $\rho_{s,\mathbf{c}}(\eta) \in (0, 1]$.

Our function will efficiently sample a lattice according to a given distribution $\mathcal{D}$. It does this by recursively sampling from the integers over the given distribution and computing the orthogonal components of its original input basis vectors. It is helpful to note that each time the recursive step is taken, one basis vector is deleted from the matrix B to form B'. The recursion will be allowed to unravel once $m = 0$, or the last basis vector has been deleted from B (or by that time B'.) The steps are then revisited as the recursion unravels. The output will be a vector sampled from the lattice I spanned by B, centered at $\mathbf{c}$, with spread parameter s.

Algorithm 5 SampleD

```
Input: basis $B \in \mathbb{Z}^{n\times m}$, spread parameter $s$, center $\mathbf{c}$
Output: a randomly chosen vector $i \in I$
 1: function SampleD($B, s, \mathbf{c}$)
 2:     if m=0 then
 3:         return 0
 4:     end if
 5:     $\widetilde{\mathbf{b}}_k = \mathbf{b}_k - \sum_{i=1}^{m-1} \mathbf{b}_i \frac{\langle \mathbf{b}_i, \mathbf{b}_k \rangle}{\langle \mathbf{b}_k, \mathbf{b}_k \rangle}$
 6:     $\mathbf{t} = \sum_{i=1}^{m} \mathbf{t} \frac{\langle \mathbf{t}, \mathbf{b}_i \rangle}{\langle \mathbf{b}_i, \mathbf{b}_i \rangle}$
 7:     $t = \frac{\langle \mathbf{t}, \widetilde{\mathbf{b}}_k \rangle}{\langle \widetilde{\mathbf{b}}_k, \widetilde{\mathbf{b}}_k \rangle}$
 8:     Choose $z \in \mathbb{Z}$ from distribution $D_{\mathbb{Z}, s/\|\widetilde{\mathbf{b}}_k\|, t}$        ▷ Subroutine mentioned above
 9:     $B' = [\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}]$
10:     $\mathbf{d} = \text{SampleD}(B', s, \mathbf{t} - z\mathbf{b}_k)$        ▷ Recursive step
11:     return $z\mathbf{b}_k + \mathbf{d}$
12: end function
```

6 Encryption and Decryption

For encryption, we combine the topics from all previous sections. Our plaintexts and ciphertexts are now viewed in terms of lattice elements. We will make heavy use of the fundamental regions associated with the different bases for our lattices, and continue to do so when assessing the growth of ciphertext vector length in Section 7.

We begin encryption with a plaintext $\pi \in \mathcal{F}_I$. To encrypt π, we make use of the property that the coset representative $\pi + i + J$ is distinguished with respect to the secret basis B_J^{sk} when choosing $i \in I$. That is, we do not choose $i \in I$ that has a corresponding $i' \in I$ such that $\pi + i \equiv \pi + i' \pmod{B_J^{sk}}$. In other words, we require $\pi + i \in \mathcal{F}_J^{sk}$. This is taken care of by our sampling distribution $\mathcal{D}$.

Also recall that encryption will use the public basis for the lattice J, denoted B_J^{pk}. Similarly, decryption will use the secret basis for J, denoted B_J^{sk}. Both encryption and decryption use the basis for I, denoted B_I, and in fact it is expected that this basis is commonly known for the purpose of computation by third parties.

6.1 Encryption with B_J^{pk}

We begin the examination of our scheme by considering encryption. A valid plaintext must first be taken from the region of valid plaintexts, $\pi \in \mathcal{P}$. To encrypt, first choose an element i of the lattice I according to a probability distribution $\mathcal{D}$ such that $\pi + i \in \mathcal{F}_J^{sk}$. Label our intermediate ciphertext as $\sigma = \pi + i$. Our ciphertext ψ will then be the reduction of σ by B_J^{pk}, $\psi = \sigma \bmod B_J^{pk}$. This operation amounts to adding an element j of J that brings the ciphertext back within the fundamental region $\mathcal{F}_{pk}$.

$$\texttt{Encrypt}(\pi) = \pi + i \bmod B_J^{pk} = \pi + i + j \qquad\qquad i \in I, j \in J$$

A graphical representation in 2 dimensions of this procedure can be seen in Figure 6, where π is referred to as p for convenience. The bases for the lattices I and J are seen first in Figure 5. In addition, the formal statement of `Encrypt` is given.

Algorithm 6 Encryption

```
Input: public basis $B_J^{pk}$ and plaintext $\pi$
Output: ciphertext $\psi$
1: function ENCRYPT($\pi$)
2:     Choose a random $i$ from $I$                    ▷ According to distribution $\mathcal{D}$
3:     $\psi = \pi + i \bmod B_J^{pk}$
4:     return $\psi$
5: end function
```

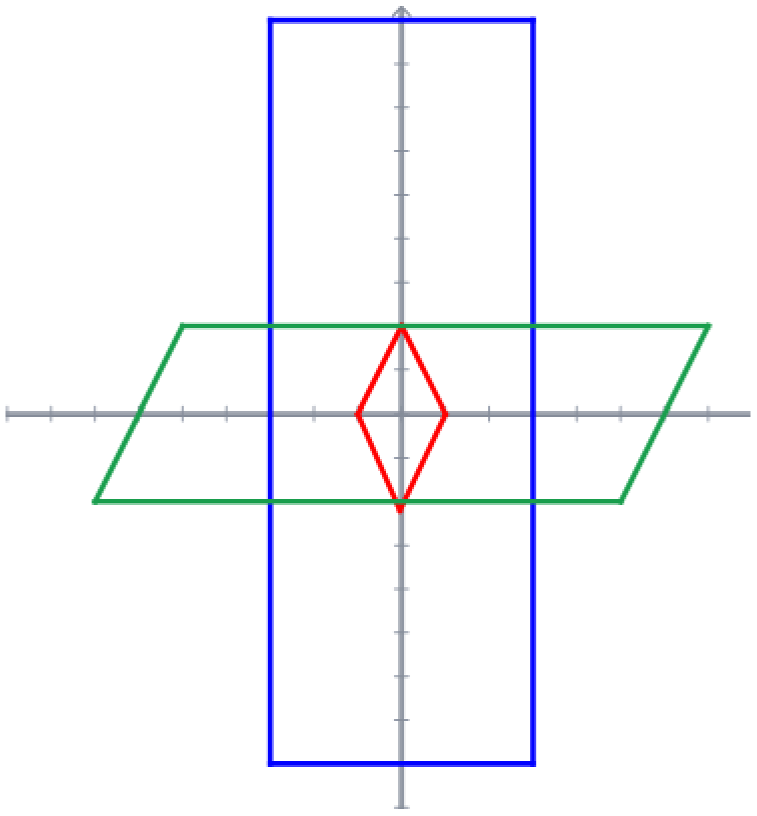

Figure 5: Basis for I (red), Secret (blue) and public (green) bases for J

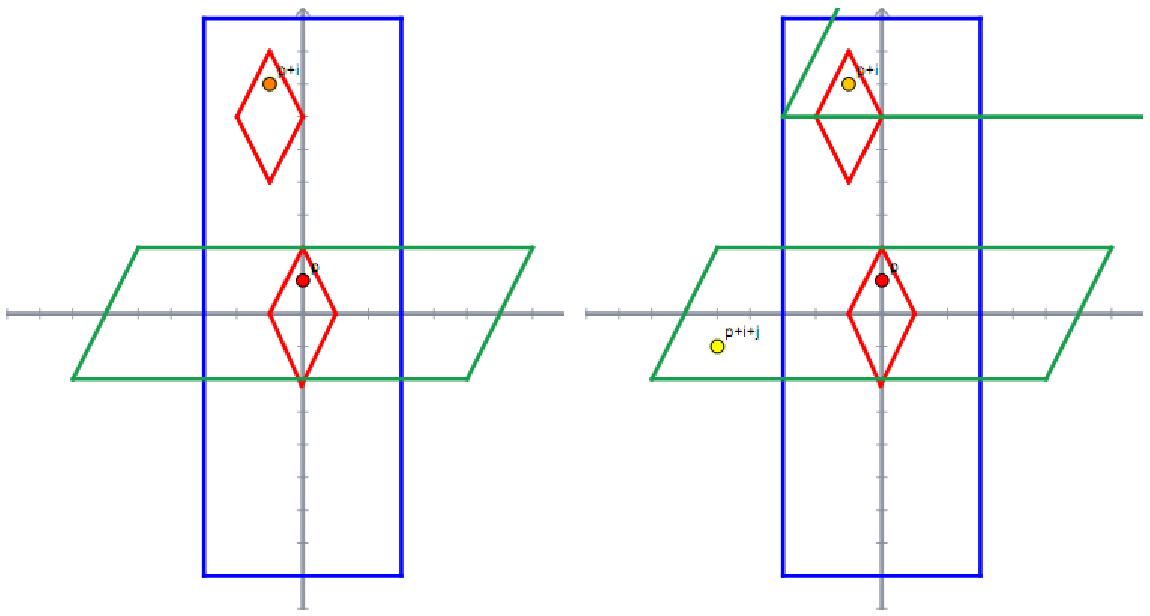

Figure 6: Encrypt: Locate the plaintext p and shift by a random element $i \in I$
Then compute $p + i \bmod B_J^{pk} = p + i + j$

6.2 Decryption with B_J^{sk}

Now that we have seen how to encrypt a plaintext π, we must see how to recover it. We are given a ciphertext ψ, and now make active use of the secret basis. To decrypt, first reduce the ciphertext by the secret basis of J. We write this as $\phi = \psi \bmod B_J^{pk}$, but recall that this action results from finding the equivalent element within $\phi \in \mathcal{F}_J^{sk}$. Furthermore, to find ϕ we are subtracting an element from J. So we may say that $\phi = \psi - j'$ for some $j' \in J$. What remains is, by definition, a distinguished representative of $\pi + i + J$, from which the message is easily recovered by reducing modulo B_I. This gives us a final element $\phi - i' = \psi - j' - i'$, where $i' \in I$. Note that correctness of decryption is nontrivial and that a proof is provided below.

$$\begin{aligned}
\texttt{Decrypt}(\psi) &= (\psi \bmod B_J^{sk}) \bmod B_I \\
&= ((\pi + i \bmod B_J^{pk}) \bmod B_J^{sk}) \bmod B_I \\
&= (\pi + i + j \bmod B_J^{sk}) \bmod B_I \\
&= \pi + i \bmod B_I = \pi.
\end{aligned}$$

Algorithm 7 Decryption

Input: private basis B_J^{sk} and ciphertext ψ
Output: plaintext π

```
1: function DECRYPT($\pi$)
2:     $\phi = \psi \bmod B_J^{sk}$
3:     $\pi = \phi \bmod B_I$
4:     return $\pi$
5: end function
```

Again, we see these steps graphically in Figure 7, where ψ is referred to as c for convenience. We also present the formal statement of `Decrypt`.

6.2.1 Correctness

We claim that decryption of a valid ciphertext correctly yields the original plaintext. Recall that to encrypt, we first take our message from the region of valid plaintexts, $\pi \in \mathcal{P} \subseteq \mathcal{F}_{\mathcal{I}}$. We choose an element i from the lattice I according to probability distribution $\mathcal{D}$, which has the property that $\pi + i$ is distinguished in $\pi + i + J$. Therefore we have that $\pi + i \in \mathcal{F}_J^{sk}$, by our hypothesis on $\mathcal{D}$.

We write $\sigma = \pi + i$, and set our ciphertext ψ to be the reduction by B_J^{pk}, $\psi = \sigma \bmod B_J^{pk}$. We express this as $\psi = \pi + i + j$ for some $j \in J$ that forces ψ to be inside $\mathcal{F}_J^{pk}$.

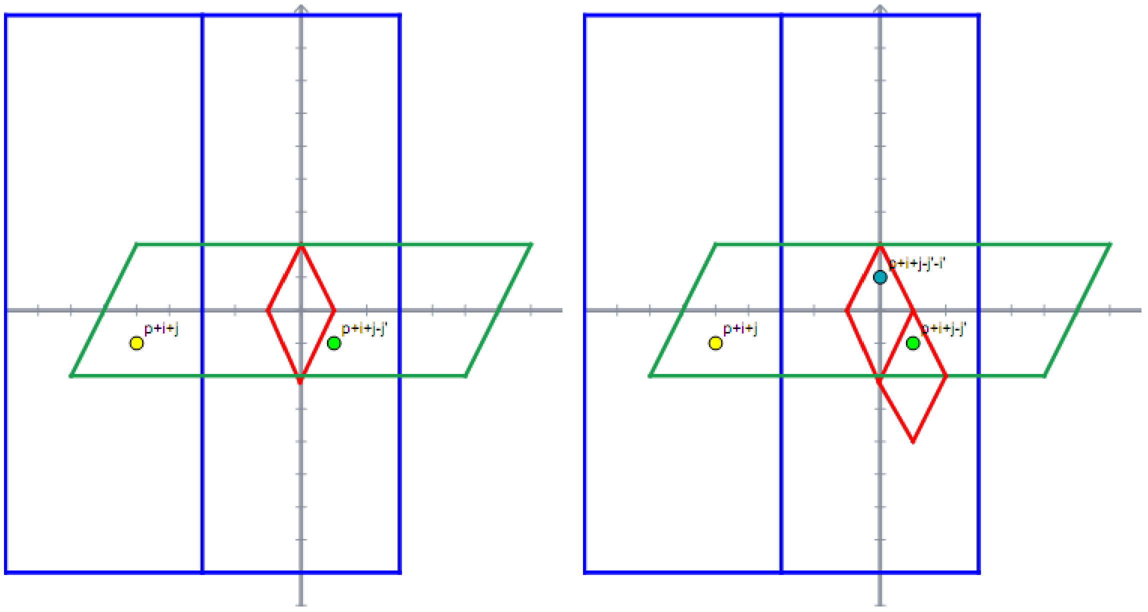

Figure 7: Decrypt: Locate c and compute compute $c \bmod B_J^{sk} = p + i + j - j'$
Then compute $(c \bmod B_J^{sk}) \bmod B_I = p + i + j - j' - i'$

Now decryption gives $\pi' = (\psi \bmod B_J^{sk}) \bmod B_I$. We then introduce $j' \in J$ and the transitional ciphertext ϕ such that $\phi = \psi \bmod B_J^{sk} = \psi - j'$, where $\phi \in \mathcal{F}_J^{sk}$. We also introduce $i' \in I$ such that $\pi' = \phi \bmod B_I = \phi - i' = \psi - j' - i'$. So $\pi' \in \mathcal{F}_I$. We must now show that $\pi' = \pi$.

Consider for a moment a more abstract scenario. Given a basis B for a lattice $\mathcal{L}$, we can consider the cosets formed by $k + \mathcal{L}$, where $k \notin \mathcal{L}$. Now note that each coset mimics the fundamental region $\mathcal{F}_B$ in a way. That is, it contains a copy of each vector within the fundamental region that is then shifted by the same element. Furthermore, we see that $k + \ell_1$ and $k + \ell_2$ in the same coset $k + \mathcal{L}$ will be equivalent modulo B. Thus, two distinct elements in the fundamental region $\mathcal{F}_B$ that we know to correspond to two elements of the coset modulo B must correspond to two distinct such elements. So if the elements within $\mathcal{F}_B$ are equivalent, the coset elements must be as well.

Recall then that $\phi \in \mathcal{F}_J^{sk}$, and by our hypothesis on $\mathcal{D}$, $\sigma \in \mathcal{F}_J^{sk}$ as well. We then have that $\phi = \sigma + (j - j')$, since both are contained in the same coset modulo J, and both ϕ and σ belong to $\mathcal{F}_J^{sk}$. Thus, $\phi = \sigma$. So $j - j' = 0$, and $j = j'$. This tells us a little more about π', namely that $\pi' = \sigma \bmod B_I$.

Now we have that π and π' are both contained in $\mathcal{F}_I$, and $\pi' = \pi + (i - i')$. Similarly as above, $\pi' = \pi$, and $i' = i$.

7 Error Growth

Now given a set of ciphertexts $\psi_1, \ldots, \psi_n \in \mathcal{F}_J^{pk}$, we must determine just how much computation we can perform before we are no longer able to decrypt. We refer to a set of these operations as a *circuit*. We also use ther terms *noise* or *error* to refer to the growth of the length of the ciphertext vector. Although we would like it to include a bit of this noise in order to be semantically secure, recall from Section 6.2.1 that a correct decryption is possible when the ciphertext lies within $\mathcal{F}_I$. That is, we must determine the maximum depth of an arbitrary arithmetic circuit C such that $\psi = C(\psi_1, \ldots, \psi_n)$ still lies within $\mathcal{F}_I$. If at each level of computation in a circuit, the length of the ciphertext grows with noise, we can establish a maximum depth d for a computable circuit that will produce a decryptable ciphertext.

We define positive real numbers r_{enc} and r_{dec} with the following properties:

- Every encryption ψ_i has length at most r_{enc}.
- Any vector of length at most r_{dec} may be properly decrypted.

For example, we may take r_{enc} to be the smallest radius of a ball centered at the origin that contains $\mathcal{F}_J^{pk}$. Likewise, r_{dec} may be taken as the largest radius of a ball centered at the origin and contained in $\mathcal{F}_I$.

7.1 Addition

We are given bounds on the lengths of our ciphertext vectors to begin with. This places a limitation on the arithmetic circuits we can evaluate. We will examine this more closely by considering individual gates within the circuit using linear algebra to describe the growth of the noise in the ciphertext ψ. For now, consider two arbitrary euclidean vectors $\mathbf{u}$ and $\mathbf{v}$, each of length at most r_{enc}. Notice that the domain of $\mathbf{u}$ and $\mathbf{v}$ is not specified. If, in our case, we consider the polynomials $u(x)$ and $v(x)$ in $\mathbb{Z}[x]$, then $\mathbf{u}$ and $\mathbf{v}$ will consist of the coefficients of $u(x)$ and $v(x)$. Furthermore, the vector $\mathbf{u}+\mathbf{v}$ represents the vector made up of the coefficients of the sum of these polynomials, and has length $\|\mathbf{u}+\mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\| \leq 2r_{enc}$ by the triangle inequality.

In fact, we can use this to bound the length of an arbitrary summation of such vectors. Let $\gamma > 0$, and consider the summation of $\mathbf{u_1}, \ldots, \mathbf{u}_\gamma$. We see that

$$\|\mathbf{u_1} + \cdots + \mathbf{u}_\gamma\| \leq \|\mathbf{u_1}\| + \cdots + \|\mathbf{u}_\gamma\| \leq \gamma \cdot r_{enc}$$

In the context of a circuit, we can quantify the *fan-in* of a single gate; that is, the number of operands. For example, a single addition gate as above would have a fan-in of γ, and could therefore sum up to γ operands.

Now that we've established a bound on the growth of noise as a result of addition, however, we will see that controlling the length of the output of a multiplication gate is not as trivial.

Consider again the euclidean vectors $\mathbf{u}$ and $\mathbf{v}$ that correspond to the polynomials $u(x)$ and $v(x)$ in the quotient ring $R = \mathbb{Z}/(f(x))$. We let $\mathbf{r}$ be the vector whose components correspond to the coefficients of $r(x) = u(x) \cdot v(x)$ in the quotient ring R. We will show that there exists $\gamma > 0$ that depends only on the quotient ring R such that the length of $\|\mathbf{r}\| \leq \gamma \cdot \|\mathbf{u}\| \cdot \|\mathbf{v}\|$. Once we obtain this bound, we will be able to estimate the growth of vector lengths for arbitrary circuits, and thereby obtain our depth bound, d.

7.2 Multiplication

Again, we consider ciphertext inputs of a bounded length and attempt to classify the growth of the product of these vectors. We can first consider the fan-in of a single multiplication gate to be 2. That is, we consider the growth of the noise or error when computing the product of two ring elements $u(x)$ and $v(x)$ in R. As before, we can then use this bound to quantify the growth of noise when computing the product of many ring elements.

Lemma 2. *Let $f(x)$ be a monic integer polynomial of degree n. Let $F(x) = x^n \cdot f(x^{-1})$ and $g(x) = F(x)^{-1} \bmod x^{n-1}$. If $u(x)$, $v(x) \in \mathbb{Z}[x]$, then the product of $u(x)$ and $v(x)$ in $\mathbb{Z}[x]/(f(x))$ has length at most $\gamma \cdot \|\mathbf{u}\| \cdot \|\mathbf{v}\|$, for some*

$$\gamma \leq \sqrt{2n} \cdot (1 + 2n \cdot \|\mathbf{g}\| \cdot \|\mathbf{f}\|).$$

Proof. Let $t(x)$ be the product $u(x) \cdot v(x)$ in $\mathbb{Z}[x]$, and $r(x)$ be the product $u(x) \cdot v(x)$ in the ring $R = \mathbb{Z}[x]/(f(x))$. We will make use of the reciprocal of $t(x)$, and so it behooves us to investigate in general the properties of $x^{n-k} \cdot h(x^{-1})$ for an arbitrary polynomial $h(x) \in R$. We will see that this process shifts the coefficients of $h(x)$, so that $\|x^{n-k} \cdot h(x^{-1})\| = \|h(x)\|$ while $k < \deg h(x)$. When convenient, we will refer to the length of the vector $\mathbf{h}$, which we write as $\|\mathbf{h}\|$. This is equivalent to the length of the polynomial $h(x)$, which we write similarly as $\|h(x)\|$. After a few examples, we will return to this proof and investigate other properties of these reciprocals to obtain our bound.

Lemma 3. *Let $h(x) \in \mathbb{Z}[x]$, and $\deg h(x) = \ell$. Let $H(x) = x^{n-k} \cdot h(x^{-1})$, where $k < \ell$. If $\boldsymbol{h}$ and $\boldsymbol{H}$ are the coefficient vectors of $h(x)$ and $H(x)$ respectively, then $\|\mathbf{h}\| = \|\mathbf{H}\|$.*

Proof. To see this, we simply apply the multiplication as follows:

$$\begin{aligned} H(x) &= x^{n-k} \cdot h(x^{-1}) \\ &= x^{n-k} \cdot (h_\ell x^{-\ell} + h_{\ell-1} x^{-(\ell-1)} + \cdots + h_1 x^{-1} + h_0) \\ &= h_\ell x^{n-k-\ell} + h_{\ell-1} x^{n-k-(\ell-1)} + \cdots + h_1 x^{n-k-1} + h_0 x^{n-k} \\ &= h_0 x^{n-k} + h_1 x^{n-k-1} + \cdots + h_{\ell-1} x^{n-k-(\ell-1)} + h_\ell x^{n-k-\ell} \end{aligned}$$

So if $h(x)$ has the coefficient vector

$$\mathbf{h} = \begin{bmatrix} h_0 & h_1 & \cdots & h_{\ell-1} & h_\ell & 0 & \cdots & 0 \end{bmatrix}$$

then $H(x)$ has coefficient vector

$$\mathbf{H} = \begin{bmatrix} 0 & \cdots & 0 & h_\ell & \cdots & h_0 & 0 & \cdots & 0 \end{bmatrix}$$

where $x^{n-k} \cdot h_i x^{-i} = h_i x^{n-k-i} = H_{n-k-i} x^{n-k-i}$. Thus, $h_i = H_{n-k-i}$ for $0 \leq i \leq \ell$. Since $\mathbf{H}$ contains the same values as $\mathbf{h}$, simply permuted, it follows that $\|\mathbf{H}\| = \|\mathbf{h}\|$. $\square$

Example 9. *To see a short example of this, consider $h(x) = 6x^4 + 3x^3 + x + 1$, so $\deg h(x) = \ell = 4$. Let $n = 6$ and $k = 2$. Then $H(x) = x^{n-k} \cdot h(x^{-1}) = x^4 \cdot (6x^{-4} + 3x^{-3} + x^{-1} + 1) = 6 + 3x + x^3 + x^4$. Constructing $\boldsymbol{h}$ and $\boldsymbol{H}$, we then have*

$$\begin{aligned} \mathbf{h} &= \begin{bmatrix} 1 & 1 & 0 & 3 & 6 & 0 & 0 \end{bmatrix} \\ \mathbf{H} &= \begin{bmatrix} 6 & 3 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{aligned}$$

Notice that if $n - k = \ell$, the entries h_0 through h_ℓ are reversed in place in $\boldsymbol{H}$.

Example 10. *Let $h(x) = 3x^3 + 2x^2 + x + 5$. Then $\deg h(x) = \ell = 3$. Let $n = 7$, and $k = 2$. Then $H(x) = x^{n-k} \cdot h(x^{-1}) = x^5 \cdot (3x^{-3} + 2x^{-2} + x^{-1} + 5) = 3x^2 + 2x^3 + x^4 + 5x^5$. So then*

$$\begin{aligned} \mathbf{h} &= \begin{bmatrix} 5 & 0 & 2 & 3 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{H} &= \begin{bmatrix} 0 & 0 & 3 & 2 & 1 & 5 & 0 & 0 \end{bmatrix} \end{aligned}$$

For $n - k > \ell$ it is possible to shift the coefficients to the right within the coefficient vector. Notice that although the entries of $\boldsymbol{h}$ are both reversed and shifted in $\boldsymbol{H}$, they are not shifted so greatly as to incur wraparound (as $h(x)$ is an element of $\mathbb{Z}/(f(x))$). This satisfies the conditions of the lemma, as $k < \deg h(x)$.

Now we return to the proof of Lemma 2. Recall that $\deg u(x), \deg v(x) < n$. So $\deg t(x) = \deg u(x) + \deg v(x) \leq 2n - 2$. Now let $T(x) = x^{2n-2} \cdot t(x^{-1})$, or rather,

$$\begin{aligned} T(x) &= x^{2n-2} \cdot t(x^{-1}) \\ &= x^{2n-2} \cdot (t_0 + t_1 x^{-1} + \ldots t_{2n-3} x^{-(2n-3)} + t_{2n-2} x^{-(2n-2)}) \\ &= t_0 x^{2n-2} + t_1 x^{2n-3} + \ldots t_{2n-3} x + t_{2n-2} \end{aligned}$$

To find the degree of $T(x)$, we must instead consider the lowest-order term of $t(x)$, since $t_i = T_{2n-2-i}$. Suffice it to say, however, that $\deg T(x) \leq 2n - 2$, where equality is possible only if both $u(x)$ and $v(x)$ have nonzero constant terms u_0 and v_0. We extend this notion to the unreduced representation of $t(x) = q(x) \cdot f(x) + r(x)$, and introduce a small lemma to help our proof along. The proof of Lemma 2 continues after Example 11.

Lemma 4. *Consider $t(x) = q(x) \cdot f(x) + r(x)$. Let $T(x)$ and $F(x)$ be as above. Let $Q(x) = x^{n-2} \cdot q(x^{-1})$ and $R(x) = x^{2n-2} \cdot r(x^{-1})$. If $t(x) = q(x) \cdot f(x) + r(x)$, then $T(x) = Q(x) \cdot F(x) + R(x)$.*

Proof. First we note that $t(x) = q(x) \cdot f(x) + r(x)$ in $\mathbb{Z}[x]$, which is contained in $\mathbb{Q}[x]$. We know then from [Gal10] that if this equation holds true in $\mathbb{Z}[x]$, then it will hold true for in $\mathbb{Q}[x]$. Consider then the quotient field $\mathbb{Q}(x)$ of $\mathbb{Q}[x]$. We know our equation still holds in $\mathbb{Q}(x)$, and furthermore that the rational function $t(x^{-1})$ is an element of $\mathbb{Q}(x)$. So the following operations are valid in $\mathbb{Q}(x)$.

$$\begin{aligned}
T(x) &= x^{2n-2} \cdot t(x^{-1}) \\
&= x^{2n-2} \cdot (q(x^{-1}) \cdot f(x^{-1}) + r(x^{-1})) \\
&= x^{n-2} \cdot q(x^{-1}) \cdot x^{n} \cdot f(x^{-1}) + x^{2n-2} \cdot r(x^{-1}) \\
T(x) &= Q(x) \cdot F(x) + R(x).
\end{aligned}$$

Since both the left-hand side and right-hand side belong to $\mathbb{Z}[x]$, the above holds true in $\mathbb{Z}[x]$. □

Example 11. *Let $n = 3$, and $t(x) = 2x^4 + x^2 + 2x + 2 = (2x) \cdot (x^3 + 1) + (x^2 + 2) = q(x) \cdot f(x) + r(x)$. Then we compute the following:*

$$\begin{aligned}
T(x) &= x^{2n-2} \cdot t(x^{-1}) \\
&= x^4 \cdot (2x^{-4} + x^{-2} + 2x^{-1} + 2) \\
&= 2x^4 + 2x^3 + x^2 + 2x
\end{aligned}$$

$$\begin{aligned}
Q(x) &= x^{n-2} \cdot q(x^{-1}) \\
&= x \cdot (2x^{-1}) \\
&= 2
\end{aligned}$$

$$\begin{aligned}
F(x) &= x^{n} \cdot f(x^{-1}) \\
&= x^3 \cdot (x^{-3} + 1) \\
&= x^3 + 1
\end{aligned}$$

$$\begin{aligned}
R(x) &= x^{2n-2} \cdot r(x^{-1}) \\
&= x^4 \cdot (x^{-2} + 2) \\
&= 2x^4 + x^2
\end{aligned}$$

And to verify, we check the following:

$$\begin{aligned} Q(x)\cdot F(x)+R(x) &= (2)\cdot(x^3+1)+(2x^4+x^2) \\ &= 2x^4+2x^3+x^2+2x \\ &= T(x) \end{aligned}$$

Returning to our original proof, recall that by our division algorithm $\deg r(x) \leq n-1$, and thus the lowest-order term of $R(x)$ is of the form $R_i x^i$, where $i \geq 2n-2-(n-1) = n-1$. So $R(x) \equiv 0 \pmod{x^{n-1}}$, and $T(x) \equiv Q(x)\cdot F(x) \pmod{x^{n-1}}$. Furthermore, we can see that $F(x)$ is guaranteed to have a nonzero constant term, since $\deg f(x) = n$ and $f_i = F_{n-k-i}$, and thus $f_n = F_0$. So $F(x)$ is also guaranteed a multiplicative inverse in $\mathbb{Q}(x)$. Let $g(x) = F(x)^{-1} \bmod x^{n-1}$, or rather the multiplicative inverse of $F(x)$ in $\mathbb{Z}[x]/(x^{n-1})$. Then $T(x)\cdot g(x) \equiv Q(x) \pmod{x^{n-1}}$.

Finally, to complete our proof we will make use of the Cauchy-Schwarz Inequality.

Lemma 5 (Cauchy-Schwarz Inequality)**.** *Let* $\mathbf{u},\mathbf{v} \in \mathbb{R}^n$*. Then* $|\mathbf{u}\cdot\mathbf{v}| \leq \|\mathbf{u}\|\cdot\|\mathbf{v}\|$*.[Rud79]*

A proof of this can also be found in [Rud79]. Continuing to look at the bound on the noise as a result of multiplication, we examine the length of the unreduced product $t(x) = u(x)\cdot v(x)$ in $\mathbb{Z}[x]$. Since $t(x) = u(x)\cdot v(x)$, each t_i is the dot product of some subset of coefficients from $u(x)$ and $v(x)$.

$$t_i = \sum_{\ell=0}^{i} u_i v_{i-\ell} = \begin{bmatrix} u_0 & u_1 & \dots & u_i \end{bmatrix} \cdot \begin{bmatrix} v_i & v_{i-1} & \dots & v_0 \end{bmatrix}$$

If we let

$$\dot{\mathbf{u}} = \begin{bmatrix} u_0 & u_1 & \dots & u_i \end{bmatrix}$$

$$\dot{\mathbf{v}} = \begin{bmatrix} v_i & v_{i-1} & \dots & v_0 \end{bmatrix}$$

then $\|t_i\| \leq \|\dot{\mathbf{u}}\|\cdot\|\dot{\mathbf{v}}\| \leq \|\mathbf{u}\|\cdot\|\mathbf{v}\|$. This allows us to bound the length of $\mathbf{t}$ as follows.

$$\begin{aligned} \|\mathbf{t}\| &= \sqrt{\sum_{i=0}^{2n-2} t_i^2} \\ &\leq \sqrt{\sum_{i=0}^{2n-2} (\|\mathbf{u}\|\cdot\|\mathbf{v}\|)^2} \\ &= \sqrt{(2n-2)\cdot(\|\mathbf{u}\|\cdot\|\mathbf{v}\|)^2} \\ &= \sqrt{2n-2}\cdot\|\mathbf{u}\|\cdot\|\mathbf{v}\| \\ &\leq \sqrt{2n}\cdot\|\mathbf{u}\|\cdot\|\mathbf{v}\| \end{aligned}$$

Note that the bound on the individual t_i, while simple in appearance, can be rather poor for particular t_i. Take, for example, $t_0 = u_0 v_0$, whose absolute value is (according to our above result) bounded by $\|\mathbf{u}\| \cdot \|\mathbf{v}\|$. This estimate is therefore quite generous. Similarly we can apply this to $Q(x)$ to obtain $\|\mathbf{Q}\| \leq \sqrt{2n} \cdot \|\mathbf{T}\| \cdot \|\mathbf{g}\|$. Then we have that the length of the product $u(x) \cdot v(x) \in \mathbb{Z}[x]/(f(x))$ is

$$\begin{aligned} \|\mathbf{r}\| &= \|\mathbf{R}\| && \text{Lemma 3} \\ &\leq \|\mathbf{T}\| + \|Q(x) \cdot F(x)\| && \text{Triangle Inequality} \end{aligned}$$

since $R(x) = T(x) - Q(x) \cdot F(x)$. Furthermore,

$$\begin{aligned} \|\mathbf{r}\| &\leq \|\mathbf{T}\| + \sqrt{2n} \cdot \|\mathbf{Q}\| \cdot \|\mathbf{F}\| && \text{Cauchy-Schwarz Inequality} \\ &= \|\mathbf{T}\| + \sqrt{2n} \cdot (\|T(x) \cdot g(x)\|) \cdot \|\mathbf{F}\| \\ &\leq \|\mathbf{T}\| + \sqrt{2n} \cdot (\sqrt{2n} \cdot \|\mathbf{T}\| \cdot \|\mathbf{g}\|) \cdot \|\mathbf{F}\| && \text{Cauchy-Schwarz Inequality} \\ &= \|\mathbf{T}\| + 2n \cdot \|\mathbf{T}\| \cdot \|\mathbf{g}\| \cdot \|\mathbf{F}\| \\ &= \|\mathbf{T}\| \cdot (1 + 2n \cdot \|\mathbf{g}\| \cdot \|\mathbf{F}\|) \\ &= \|\mathbf{t}\| \cdot (1 + 2n \cdot \|\mathbf{g}\| \cdot \|\mathbf{f}\|) && \text{Lemma 3} \\ &\leq \sqrt{2n} \cdot \|\mathbf{u}\| \cdot \|\mathbf{v}\| \cdot (1 + 2n \cdot \|\mathbf{g}\| \cdot \|\mathbf{f}\|) && \text{Cauchy-Schwarz Inequality} \end{aligned}$$

Letting $\gamma \leq \sqrt{2n} \cdot (1 + 2n \cdot \|\mathbf{g}\| \cdot \|\mathbf{f}\|)$, we then have that the product $r(x) = u(x) \cdot v(x)$ in $\mathbb{Z}[x]/(f(x))$ has length

$$\|\mathbf{r}\| \leq \gamma \|\mathbf{u}\| \cdot \|\mathbf{v}\| \leq \sqrt{2n} \cdot \|\mathbf{u}\| \cdot \|\mathbf{v}\| \cdot (1 + 2n \cdot \|\mathbf{g}\| \cdot \|\mathbf{f}\|)$$

□

Now that we have an idea of how much ciphertext will grow with each type of operation, we can quantify the depth of a valid circuit. In other words, we can find a bound on the maximum depth of a circuit so that the ciphertext output of that circuit does not grow beyond r_{dec}.

7.2.1 Maximum Circuit Depth

Recall that in general for valid encryption, we need the ciphertext to lie within the ball of radius r_{dec}. For this to occur, the length of the ciphertext ψ can be no larger than r_{dec}. Note that although we refer to ψ in neither vector, nor polynomial notation, it is an element of the ring $R = \mathbb{Z}/(f(x))$, and as such has a coefficient vector ψ. It is the length of this coefficient vector, denoted $\|\psi\|$, that we are bounding by r_{dec}, and not the number of bits in the ciphertext ψ. We will continue the use of arbitrary vectors $\mathbf{u}$ and $\mathbf{v}$ in place of these to avoid confusion.

We will proceed by considering the ciphertext at each level of the circuit. By using r_{dec} as a bound on the length of the vector ψ, we can work backwards to obtain a depth d for which the length of ψ does not exceed this bound.

Theorem 4. *Let $r_{enc} \geq 1$. For a circuit C, let the additive fan-in of C be γ, the multiplicative fan-in of C be 2, and the depth of C be at most*

$$loglog(r_{dec}) - loglog(\gamma \cdot r_{enc}).$$

Then $\mathbf{w} = C(\mathbf{u_1}, \ldots, \mathbf{u_n})$ has length at most r_{dec}, for $\|\mathbf{u_i}\| \leq r_{enc}$.

Proof. We begin by considering an upper bound on the size of the vectors at level i. Call this bound r_i. The input ciphertexts to our circuit will therefore have length at most r_0, and the final output ciphertext will have length at most r_d. Our claim is then that $r_d \leq r_{dec}$ for a circuit of depth d. We have already seen that the output $\mathbf{u}$ of an addition operation with fan-in γ has length $\|\mathbf{u}\| \leq \gamma \cdot r_i$, and the output $\mathbf{v}$ of a multiplication operation has length $\|\mathbf{v}\| \leq \gamma \cdot r_i^2$, for operands of length at most r_i.

As the length of the ciphertext produced by a multiplication operation has a larger bound than that of an addition operation, we can bound the length of the ciphertext produced by an addition operation by $\gamma \cdot r_i^2$ as well. Intuitively this amounts to thinking of each operation as a multiplication operation, whether or not it actually is. In that sense, the length of its output is bounded by $r_{i+1} \leq \gamma \cdot r_i^2$. Subsequently at level $i+2$ the length is bounded by $r_{i+2} \leq \gamma \cdot r_{i+1}^2 \leq \gamma \cdot (\gamma \cdot r_i^2)^2 = \gamma^3 \cdot r_i^4$.

Starting at level 0, the input to the the circuit, we then see that the length of the vectors at level 2 are bounded by $r_2 \leq \gamma^3 \cdot r_0^4$, and the length of the vectors at level 3 are bounded by $r_3 \leq \gamma^7 \cdot r_0^8$, and so on. So at level i, we have

$$r_i \leq \gamma r_{i-1}^2 \leq \gamma^3 r_{i-2}^4 \leq \cdots \leq \gamma^{2^i - 1} \cdot r_0^{2^i}$$

We can extend this to the end of our circuit at level d to obtain $r_d \leq \gamma^{2^d-1} \cdot r_0^{2^d}$.

Furthermore, if we can assume that our input ciphertexts are freshly encrypted, they will have length $r_0 \leq r_{enc}$. So $r_d \leq \gamma^{2^d-1} \cdot r_{enc}^{2^d}$.

Our goal then becomes to solve the inequality $r_{dec} \leq \gamma^{2^d-1} \cdot r_{enc}^{2^d}$ for d. We proceed in the following way:

$$\begin{aligned}
r_{dec} &\leq \gamma^{2^d-1} \cdot r_{enc}^{2^d} \\
log(r_{dec}) &\leq log(\gamma^{2^d-1} \cdot r_{enc}^{2^d}) = 2^d \cdot log(\gamma \cdot r_{enc}) \\
log(log(r_{dec})) &\leq log(2^d \cdot log(\gamma \cdot r_{enc})) = d \cdot log(2) + log(log(\gamma \cdot r_{enc})) \\
log(log(r_{dec})) - log(log(\gamma \cdot r_{enc})) &\leq d
\end{aligned}$$

This gives us the desired bound on d. □

This bound dictates the number of *levels* a computable circuit can have. If two operations can be performed in parallel, then we need only count them both as a single level.

8 Bootstrapping

By the limitations outlined above, our scheme is still only *somewhat* homomorphic (i.e. it can only perform a certain number of operations before it can no longer properly decrypt the ciphertext.) But we will see that periodically refreshing the ciphertext allows us to perform an unlimited number of operations. The method we use is called *bootstrapping.*

Definition 8. *We call a scheme* ***bootstrappable*** *if it can homomorphically evaluate its own decryption circuit.*

It is important to keep in mind two key concepts:

1. **Operations** The scheme must be homomorphic with respect to any gates that appear in the decryption circuit. Our scheme is homomorphic (although limited depth-wise) with respect to ring addition and multiplication, which together allow for universal computation.

2. **Noise** The scheme must be able to evaluate the entire decryption circuit without creating too much noise. We saw in Section 7 how operations will introduce noise into our ciphertexts by causing them to grow in length.

8.1 Recrypt

For our calculations in Section 7, we assumed that a freshly encrypted ciphertext has length at most r_{enc}. At this point our goal is then to take a long ciphertext ψ_1 (the result of computation), and create another ciphertext ψ_2 that is shorter in length.

To do this, the encrypter (below, Clyde) first chooses another pair of public and secret bases for the lattice J. We call these $B_J^{pk_2}$ and $B_J^{sk_2}$ respectively, with the original pair of bases denoted $B_J^{pk_1}$ and $B_J^{sk_1}$. The encrypter then sets $\beta_J^{sk_1}$ to be the encryption of the first secret basis $B_J^{sk_1}$ under the second public basis $B_J^{pk_2}$.

$$\beta_J^{sk_1} = \texttt{Encrypt}_{B_J^{pk_2}}(B_J^{sk_1})$$

He then sends the newly encrypted secret key $\beta_J^{sk_1}$ and the new public key $B_J^{pk_2}$ to the computational service provider (below, Sergei), who then *homomorphically* re-encrypts the ciphertext before continuing his computations.

More concretely, this re-encryption is encapsulated in the function `Recrypt`, which first computes the encryption of ψ_1 with respect to the second public basis $B_J^{pk_2}$

$$\sigma = \texttt{Encrypt}_{B_J^{pk_2}}(\psi_1)$$

and then the decryption of this with respect to the encrypted secret basis $\beta_J^{sk_1}$.

$$\psi_2 = \texttt{Decrypt}_{\beta_J^{sk_1}}(\sigma)$$

Since both σ and $\beta_J^{sk_1}$ are encrypted under the same public basis, the decryption circuit can be homomorphically evaluated using both as input. The result of `Recrypt` is a new ciphertext ψ_2 of shorter length than ψ_1, whose proper decryption yields the same plaintext. That is,

$$\texttt{Decrypt}_{B_J^{sk_2}}(\psi_2) = \texttt{Decrypt}_{B_J^{sk_1}}(\psi_1)$$

The re-encryption algorithm `Recrypt` is provided below, with stages of encryption and decryption specified.

Algorithm 8 Re-encryption

```
Input: encrypted private basis $\beta_J^{sk_1}$, public basis $B_J^{pk_2}$, and ciphertext $\psi_1$
Output: ciphertext $\psi_2$
1: function RECRYPT($\psi_1$)
2:     Encryption
3:     Choose a random $i$ from $I$                ▷ According to distribution $\mathcal{D}$
4:     $\sigma = \psi_1 + i \bmod B_J^{pk_2}$
5:     Decryption
6:     $\phi = \sigma \bmod \beta_J^{sk_1}$
7:     $\psi_2 = \phi \bmod B_I$
8:     return $\psi_2$
9: end function
```

Although there are some applications where it may be beneficial for the client to provide his keys at an earlier or later date than specified, we assume that the number of keys necessary is computed beforehand. This way, the client sends his keys along with his data and computation instructions.

Below we see how Clyde prepares to send his data to Sergei. Clyde must first generate his public and private bases for J. He may then use the first public basis to encrypt his plaintexts. Next he must successively encrypt each private basis (except $B_J^{sk_\ell}$) with the next pair's public basis. Clyde may then send all of his encrypted data to Sergei, and Sergei may begin computation.

Once Sergei's ciphertexts reach a threshold of length close to r_{dec}, he must re-encrypt them using the `Recrypt` algorithm. After he re-encrypts his data, however, he is free to proceed with computation. When his computations are complete, Sergei sends the encrypted result back to Clyde (also specifying which public basis it is encrypted under if Clyde does not already know.) Clyde may then decrypt the ciphertext to obtain the desired result.

Clyde	Sergei

Generate keys
$\{B_J^{pk_1}, \ldots, B_J^{pk_\ell}\}$ and $\{B_J^{sk_1}, \ldots, B_J^{sk_\ell}\}$

Encrypt plaintext arguments under a single key
$\psi_j = \texttt{Encrypt}_{B_J^{pk_1}}(\pi_j)$

Encrypt secret keys successively
$\beta_J^{sk_i} = \texttt{Encrypt}_{B_J^{pk_{i+1}}}(B_J^{sk_i})$

Send ciphertext and keys to Sergei
$\{\psi_1, \ldots, \psi_n\}$
$\left\{\beta_J^{sk_1}, \ldots, \beta_J^{sk_{\ell-1}}\right\}, \left\{B_J^{pk_1}, \ldots, B_J^{pk_\ell}\right\}$ $\longrightarrow$ Begin computation

$\vdots$

Re-encrypts the ciphertext
$\psi_{j+1} = \texttt{Recrypt}(\psi_j)$
Repeat as necessary

$\vdots$

Return ciphertext ψ_m,
Decrypt ciphertext $\longleftarrow$ encrypted under $B_J^{pk_m}$
$\pi_m = \texttt{Decrypt}_{B_J^{pk_m}}(\psi_m)$

Clyde must know which public basis the ciphertext is encrypted under so that he can decrypt it with the proper secret basis. It might seem that Sergei could attempt to decrypt one of the encrypted secret bases, but in order to do this he will need to decrypt all of the secret bases. In fact, he will need the unencrypted secret basis $B_J^{sk_\ell}$, which Clyde does not send him.

8.2 Usability

To ensure that the noise stays at a manageable level, we re-encrypt the data before the noise reaches the threshold that prevents proper decryption. A valid circuit is one that produces a ciphertext that can be properly decrypted. If in addition we can evaluate this circuit *and* re-encrypt the data without going over the threshold of noise, we call this pair an *augmented* circuit.

If we do not reuse keys, a new (B_J^{pk}, B_J^{sk}) pair will be needed for each augmented circuit. In the worst case, a new key pair will be needed for every level of the circuit. In Section 3.1, however, we showed that it is sufficient if we can evaluate a single `NAND` gate before re-encrypting. Through the use of only `NAND` gates, we can evaluate any arithmetic function. Recall that X `NAND` Y translates to $1 - X \cdot Y$, which uses a single multiplication and a subtraction.

The above assumes that the decryption circuit can be evaluated without creating a ciphertext whose length exceeds that of r_{dec}. To achieve this, however, some modifications needs to be made to the scheme to make the decryption circuit sufficiently shallow. The modifications proposed in [Gen09a] are twofold. First, the size of the secret basis is decreased by using fractional ideals. Second, the size of r_{enc} is decreased, mainly outlined in Chapter 9 of [Gen09a]. This change restricts our set of vectors closer the origin within $\mathcal{F}_J^{sk}$, allowing us to use fewer bits of precision in our computation of the decryption circuit.

References

[bgn] Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 325–341.

[Cal09] Lewis Call. Michael l. dertouzos, April 2009. http://www.kurzweilai.net/michael-l-dertouzos.

[Cal12] Lewis Call. History of network technology, April 2012. http://cla.calpoly.edu/ lcall/354/.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag, second edition, 1993.

[Gal10] Joseph A. Gallian. *Contemporary Abstract Algebra.* Brooks/Cole, seventh edition, 2010.

[Gen09a] Craig Gentry. *A Fully Homomorphic Encryption Scheme.* PhD thesis, Stanfard university, 2009.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. STOC '09, pages 169–178. ACM, 2009.

[GPV07] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. Cryptology ePrint Archive, Report 2007/432, 2007.

[Gre09] Andy Greenberg. IBM's blindfolded calculator. *Forbes Magazine*, 2009.

[Gre11] Andy Greenberg. An MIT magic trick: Computing on encrypted databases without ever decrypting them. *Forbes Magazine*, 2011.

[Mic01a] Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, 2001.

[Mic01b] Daniele Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, March 2001.

[PP10] Christof Paar and Jan Pelzl. *Understanding Cryptography.* Springer-Verlag, second edition, 2010.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169,179, 1978.

[Rud79] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, Inc., third edition, 1979.

[vDJ10] Marten van Dijk and Ari Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. Cryptology ePrint Archive, Report 2010/305, 2010.

9 Appendix A: Notation

The following notation is used throughout the text when discussing or describing algebraic quantities:

- $u(x)$ A polynomial in either $\mathbb{Z}[x]$ or $R = \mathbb{Z}[x]/f(x)$ with the coefficients $u_0, \ldots, u_n$.
- $\mathbf{u}$ The vector whose components are the coefficients of $u(x)$ in order from least to most significant power.
- $\|\mathbf{u}\|$ The Euclidean norm (2-norm) of the vector $\mathbf{u}$, evaluated by $\|\mathbf{u}\| = \sqrt{\sum_{i=1}^{n} u_i^2}$.
- $\|u(x)\|$ The length of the polynomial $u(x)$, quantified by the Euclidean norm of the vector $\mathbf{u}$.
- u_i The i^{th} coefficient of $u(x)$ corresponding to the term $u_i x^i$; also the i^{th} component of $\mathbf{u}$.
- $u(x) \cdot v(x)$ The polynomial corresponding to the product $(u_n x^n + \cdots + u_1 x + u_0) \cdot (v_n x^n + \cdots + v_1 x + v_0)$.
- $\mathbf{u} \cdot \mathbf{v}$ The inner (dot) product of the two vectors $\mathbf{u}$ and $\mathbf{v}$, $u_n v_n + \cdots + u_1 v_1 + u_0 v_0$.

10 Appendix B: Sage Code

10.1 Basis Reduction Functions

```
def MakeNonNegative(A,i,k):
    if A[i][k] < 0:
        for i in range(0,m):
            A[i][k] = (-1)*A[i][k]
    return A

def Swap(A, j0, k):
    for i in range(0,m):
        x = A[i][j0]
        A[i][j0] = A[i][k]
        A[i][k] = x
    return A



def RowFinished(A, i, k):
    finished = 1
    for j in range(0,k-1):
        if A[i][j] != 0:
            finished = 0
    return finished



def Reduce(A, i, k):
    b = A[i][k]
    for j in range(0,k):
        q = A[i][j]*(1/b)
        for x in range(0,m):
            A[x][j] = A[x][j] - q*A[x][k]
    return A

def FinalReduce(A,i,k):
    b = A[i][k]
    for j in range(k+1,n):
        q = A[i][j]*(1/b)
        for x in range(0,m):
            A[x][j] = A[x][j] - q*A[x][k]
    return A

def gcd_reduce(A,i,j,k):
    euclid = Integer._xgcd(A[i][k],A[i][j])
    for row in range(0,m+1):
        B[row] = euclid[1]*A[row][k] + euclid[2]*A[row][j]
    for row in range(0,m+1):
        A[row][j] = float(A[i][k]/euclid[0])*A[row][j] - float(A[i][j]/euclid[0])*A[row][k]
    for row in range(0,m+1):
        A[row][k] = B[row]
    return A
```

10.2 Hermite Normal Form Without GCD

```
while i != l:

        if RowFinished(A, i, k):
            A = MakeNonNegative(A,i,k)
            if A[i][k] == 0:
                k = k + 1
            else:
                A = FinalReduce(A, i, k)
            i = i - 1
            k = k - 1
        else:
            A = MakeNonNegative(A,i,k)
            min_a = A[i][k]
            min_j = k
            for j in range(0,k-1):
                if A[i][j] < min_a and A[i][j] != 0:
                    min_a = A[i][j]
                    min_j = j

            A = Swap(A,min_j,k)
            A = MakeNonNegative(A,i,k)

            A = Reduce(A, i, k)
```

10.3 Hermite Normal Form with GCD

```
while i != l:
    while j != 0:
        j = j-1
        if A[i][j] == 0:
            A = gcd_reduce(A,i,j,k)

    b = A[i][k]
```

```
        if b > 0:
            A = MakeNonNegative(A,i,k)
            b = -1*b
        if b == 0:
            k = k+1
        else:
            for col in range(0,k+1):
                q = floor(A[i][col]/b)
                for i in range(0,m+1):
                    A[i][col] = A[i][col] - q*A[i][k]
        i = i-1
        k = k-1
        j = k

W = []
for j in range(0,n-k):
    for i in range(0,m+1):
        W[i][j] = A[i][j+k-1]
```