2017

Cyclic Codes and Cyclic Lattices

Scott Maislin
Claremont McKenna College

Cyclic Codes and Cyclic Lattices

Scott Maislin
Advisor: Lenny Fukshansky

Senior Thesis in Mathematics
Submitted to Claremont McKenna College

Aprill 24, 2017
Department of Mathematical Sciences

Abstract

In this thesis, we review basic properties of linear codes and lattices with a certain focus on their interplay. In particular, we focus on the analogous constructions of cyclic codes and cyclic lattices. We start out with a brief overview of the basic theory and properties of linear codes. We then demonstrate the construction of cyclic codes and emphasize their importance in error-correcting coding theory. Next we survey properties of lattices, focusing on algorithmic lattice problems, exhibit the construction of cyclic lattices and discuss their applications in cryptography. We emphasize the similarity and common properties of the two cyclic constructions.

Acknowledgments

I would like to thank my wife Alaina for tolerating the wild mood swings brought on by this project. Also, thank you to my daughter Eloisa. As I watch you learn to crawl by dragging yourself across the floor I can see that all good things are learned through struggle. Finally, I must thank my thesis advisor Professor Lenny Fukshansky. I swear I learned something in the last year. A wise man once told me that something is better than nothing. Thank you for giving me the opportunity to challenge myself.

Contents

1. **Introduction**

The Microcomputer revolution of the 1980s changed the face of information exchange. Prior to this time the average individual came into contact with computers only in rare instances. The advent of the Microcomputer, a computer at least small enough to fit in a room, caused an exponential growth in the exchange of digital information. The benefits of the proliferation of digital communication are of course all around us. One example is the ability to conduct commerce and finance over an open channel, the Internet for example, on a wide scale.

While effects of digital communication and the proliferation of Microcomputers are evident in all aspects of modern life they are particularly evident in the aforementioned examples. In e-commerce, for example, there is an ever present need to exchange information in a clear manner and to maintain the security of a channel. Let us consider a hypothetical scenario. The reader Alice realizes that 'Fifty Shades of Blue' is available for sale as an ebook from vendor Bob, who owns www.booksforsale.com. Alice wishes to to order this book and sends her credit card information and email address to complete the order. In order to make such a transaction possible we must first ensure that the information sent is accurate across a channel which is possible noisy. Define noise to be any interference which could cause a different message from the intended one to arrive. This is important because we want to avoid the seller receiving an order for another similarly titled book. Also, for the payment to work the credit card number must be accurate. Noise might cause one digit in the credit card number to be incorrect or deleted.

We must also consider the need to speak securely. Alice lives in country X where 'Fifty Shades of Blue' is banned. Eve is an Internet criminal who sometimes works with the government of country X. Eve would like to know who is buying banned books in country X in order to sell that information to the government. Also, Eve, in the process of intercepting the transaction also obtains Alice's credit card number. Eve sells the information to country X and then uses Alice's credit card number to buy expensive electronics which she then resells for cash. Alice is arrested and jailed without trial. Her family, completely broke, becomes destitute. We see the need to make information secure over a clear channel. Alice could encrypt her order. Now Alice and Bob are able to send the order across the clear channel but when Eve intercepts the order she can only see a string of unintelligible letters and numbers. This makes the intercepted information useless to Eve.

We now define some key terms. Coding theory is the use of codes in transmitting information with error correcting capabilities, data compression, and cryptographic communication. A code is any method of representing information such as a word, number, gesture, or an image into a compressed and sometimes secret medium. Written language is in itself an example of an encoding of the spoken language. Morse code is a coding of the written language into sound for transmission over great distances. Error correcting codes are capable of detecting when an error occurs in transmission and then correcting said error without the need for retransmission. Data compression, which we will unfortunately not cover, is the stripping away all but the most essential elements in the information. This saves bandwidth. The information can then later be restored to its full form when needed for use. Finally, cryptography

is the scrambling of information with the intent that it is illegible to everyone except those who understand the method by which it was scrambled. By first scrambling a message, and then transmitting it via an error correcting code we can transmit a secure message and minimize the risk of sending and receiving incorrect information.

While multiple types of error correcting codes exist this paper will focus on linear codes. We define linear codes as a collection of linearly independent code words $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n$ closed under addition. Simply put any linear combination of words $\sum_{i=1}^{n} \boldsymbol{\alpha}\boldsymbol{c}_i$ where $\boldsymbol{c}_i$ is an element of the code and $\boldsymbol{\alpha}$ is any scalar will give the result of another element of the code.

Within the family of linear error correcting codes lies the subset of codes known as block codes. Important and famous linear block codes that will be covered are the Hamming $[7, 4, 3]$ codes and the Golay codes. Following this we will briefly overview the construction of a structured family of block codes known as cyclic codes.

Cyclic codes are in many ways analogous to cyclic lattices. We will first overview the lattice in general. This will segue into the topic of classic lattice problems such as closest vector problem and shortest vector problems. These problems will require an explanation of basic tenants of computational complexity. Finally we will cover the basic definitions of public key cryptography and how cyclic lattices can be used to construct encryption algorithms. One such algorithm, NTRU which is resistant to quantum computing attacks and requires less computational resources than another non lattice based algorithm called RSA encryption.

2. Theory of linear codes

2.1. **Finite fields and polynomial rings.** As stated in the introduction linear codes are an important family of error correcting codes. A convenient fact in regards to linear codes is that they have algebraic structure to them. Specifically they have the structure of what is called a finite field. Let us first define a field.

Definition 2.1(a): A field, referred to as $\mathbb{F}$, is a ring whose non-zero elements form a multiplicative Abelian Group. In short a non-zero commutative division ring. This creates an algebraic structure which conforms to the following rules.

(1) $\mathbb{F}$ is closed under $+$ and $\cdot$ such that the output of any elements in the set under these operations will be in the set. $+, \cdot$ are typically referred to as addition and multiplication.

(2) The afore-mentioned operations, $+, \cdot$ are commutative. For example, let there exist two elements in the set a, b. Then

$$a + b = b + a, a \cdot b = b \cdot a$$

(3) Associativity also holds. I.E.:

$$(a + b) + c = a + (b + c), (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

(4) The distributive law holds:

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

(5) Being that a field is a commutative non-zero division ring there must exist identity elements for the two operations $+, \cdot$. These are defined as:

$$a + 0 = a \text{ for all } a \text{ in } \mathbb{F}$$
$$a \cdot 1 = a \text{ for all } a \text{ in } \mathbb{F}$$

(6) Since $\mathbb{F}$ is a ring with unique properties it must of course contain the additive inverse such that for any a in $\mathbb{F}$:

$$a + (-a) = 0$$

(7) The unique property which distinguishes fields from rings is that for any non zero element of the field there exists a multiplicative inverse such that

$$a \cdot a^{-1} = 1$$

whereas examples of infinite fields are $\mathbb{Q}$ and $\mathbb{R}$. A field of a finite cardinality is called a finite field. This cardinality is referred to as the order of the field. The order of this field, q is denoted $\mathbb{F}_q$.

There are some basic properties in regards to finite fields. [15]

(1) The order, or number of elements in finite field must be a prime power. Let $p^n = q$ be the order of a finite field where p is some prime number and n is a positive integer. It is notable that all finite fields of the same order are isomorphic to one another. Finite fields of order p^n are denoted $\mathbb{F}_{p^n}$

(2) $(x + y)^p = x^p + y^p$ where x and y are elements in a field of order p.

(3) $x^p - x = \prod(x - a)$ where element a is any element of the field. More generally we can say $(x^p)^n - X = 0$ Without running through a complete proof we can see from this property that finite fields are cyclic.

It is important to understand that while the most basic version of a finite field is $\mathbb{Z}/p\mathbb{Z}$, i.e a finite field of integers modular some prime p, we are not limited to such examples. If we begin with a field $\mathbb{F}_q$ then we can create an extension of this field. The term extension is by definition an add on to our original field. A simple example is that $\mathbb{C}$ is an extension of $\mathbb{R}$. Conversely the rationals are subfield of the complex numbers. Formally, A field $\mathbb{K}$ is called an extension or extension field of a field $\mathbb{F}$, if $\mathbb{F}$ is contained in $\mathbb{K}$.

If we take our field $\mathbb{F}_q$ then $\mathbb{F}_q[X]$ is the ring of polynomials whose coefficients are between 0 and $q - 1$. Recall the field $\mathbb{F} = \mathbb{Z}/p\mathbb{Z}$. This is a quotient ring that is also a field by the fact that the ring is modular some integral domain, i.e. there are no zero divisors, and that the same element is maximal, i.e. the quotient ring is not modular some element whose entire multiplicative image is contained in the image of another single element. As a result a finite field of integers must be of a prime order, in other words modular some prime element. If we were instead to create a finite field from a quotient ring of polynomials $\mathbb{F}_q[X]/ < g >$. $\mathbb{F}_q[X]$ is now modular a polynomial, $< g >$ which meets specific requirements. The polynomial quotient ring is a field by virtue of the fact that $< g >$ is maximal. We will put this in formal terms but first must define the term ideal which up to this point we have been skirting around.

Definition 2.1(b): If R is a ring then the ideal I is a non empty subset of R which contain two properties.

(1) If $a, b \in I$ then $a + b \in I$

(2) If $a \in I$ and $r \in R$ then $ar \in I$ and $ra \in I$

[5]

Theorem 2.1(a). Let I contained in $\mathbb{F}[X]$ be an ideal Then there exists a polynomial $g(x)$ in $\mathbb{F}[x]$ such that $I =< g(x) >$ such that $\{g(x)f(x) : f(x) \in \mathbb{F}[x]\}$ Remark: This property means that every ideal in F[x] is principally generated. In other words. Generated by a single element. Rings with this property are called principal ideal rings.

[5]

Proof: To show that the ideal is generated by single polynomial g we simply consider Euclid's division algorithm. Let $g \in I$ be a non-zero polynomial of least degree where $g = I$. If $f \in I$ also then $f = qg + r$ where q, r are polynomials of degree less than g. We see that because g is of least degree and degree $r <$ degree g then the degree of r must be 0. This leads us to the fact $f = qg$. By definition of the ideal every multiple of $g \in I$. So $I \subseteq g$ and $g \subseteq I$ so $I = G$.

Furthermore we can say that the quotient of said ring $\mathbb{F}[x]$ is a field when $< g >$ is maximal. A maximal ideal is an ideal which not contained in a larger ideal of the same ring $\mathbb{F}[x]$. For example within the integers $\mathbb{Z}/6\mathbb{Z}$ we can see that 6 is an ideal meeting the requirements of definition 2.1(c). However, closer examination shows that the ideal generated by the integer 6 is contained entirely within an ideal generated by 2. In brief $< 2 > \subseteq < 6 >$

Definition 2.1(c): Let R be a commutative ring with identity. An ideal P contained in R is called prime if wherever ab in P for some a, b in R, then at least one of $a, b \in P$.
[5]

Remark: For the purposes of commutative rings a prime ideal P and maximal ideal are interchangeable terms. However in more abstract terms this is never the case.

Theorem 2.1(b):. Let P contained in $\mathbb{F}[x]$ be an ideal. Then P is prime iff there exists an irreducible polynomial $g(x)$ in P such that P is generated by this $g(x)$ [5]

Proof: Suppose the ideal P is maximal and that P is generated by polynomial $< g >$. Write $< g >= fq$ for some $f, q \in R$. Since $f| < g >, P$ must be a subset of the ideal generated by f. Were this inclusion to be proper, by the maximality of P, we would have R being generated by f. This make f a unit. By symmetry, f or q is a unit. We conclude that $< g >$ is irreducible.

Theorem 2.1(c): $\mathbb{F}[x]/ < g >$ is a field iff $< g >$ generates a prime ideal I.
[5]

Proof: $\Rightarrow$ Suppose I is a maximal ideal. Suppose $I + a \in R/I$ such that $I + a \neq I + 0$ $(a \notin I)$. Consider $J = I + Ra = \{i + ra | i \in I, r \in R\}$. Note that J is an ideal. Also, for all $i \in I, i = i + 0a \in j$, such that $I \subseteq J$. Since I is maximal, $J = I$ or $J = R$. Note $a = 0 + 1a \in J$, but $a \notin I$ so $J \neq I$. This forces $J = R$ such that, $I + Ra = R \supseteq 1$, hence $1 = i + ra$ for some

$i \in I, r \in R$. So $1 - ra = i \in I$, so $I + 1 = I + ra$. This means $(I + r)(I + a) = I + 1$. This shows that $(I + a)$ is a unit hence R/I is a field.

$\Leftarrow$ Suppose R/I is a field. Let J be an ideal $J \supset I$. Then there exists some element $x \in J$ but $x \notin I$ such that $I + x \neq I + 0$ Since R/I is a field, so there is some $I + y \in R/I$ such that

$$(I + x)(I + y) = I + 1$$

$$I + xy = I + 1.$$

hence $xy - 1 \in I \subseteq J$. Note $1 = xy - xy - 1 \in J$ Then for all $r \in R, r = r \cdot 1 \in J$, so $R = J$ Which proves I is maximal.

2.2. **Linear codes and Error Correction: definitions and basic properties.** We start with some definitions which will help us improve our understanding of what makes a code linear. We then discuss in very basic terms the idea behind error correcting codes. In initial laymen's terms we see that a code is linear when combinations of any elements in the code will remain in the code. Each word, or vector, in the code is a message which is designed to be sent over a noisy channel. When errors occur error correcting codes detect and correct errors up to certain limits. Many error correcting codes are linear and we will only discuss this subset of the family of error correcting codes. The following definitions, a through f, are found in [10].

Definition 2.2(a): A linear code of length n over the finite field $\mathbb{F}_q$ is a subspace of the finite field $\mathbb{F}_q$ where n is the characteristic, also known as

order, of the field. Any codeword in the code is a vector of the subspace $(\mathbb{F}_q)^n$. Commonly referred to as a code vector.

Henceforth we can refer to the subspace which contains the linear code by C. Any such code has an alphabet of size q since it is a subspace of the field $\mathbb{F}_q$. So C is q-ary code. Of course when C is a subspace of $\mathbb{F}_2$ or $\mathbb{F}_3$ it is a binary or ternary code respectively. Since C is a subspace it adheres to normal rules of sub spaces such as:

(1) For elements a, b in the linear code $\mathbf{C}$, $(a + b)$ is in $\mathbf{C}$ for any a, b in $\mathbf{C}$
(2) Any element a multiplied by a scalar remains in $\mathbf{C}$ as long as said scalar is in $\mathbb{F}_q$.

Since C is a subspace of length n it will have a basis $\{c_1, c_2 \ldots, c_k\}$ where k is the dimension of the subspace. The size of the codes can be expressed in the terms q^k. C. In other words C has q^k possible vectors. Any word or code vector in the code is a linear combination of the above basis. The vectors which form said basis are written in an $n \times k$ matrix commonly referred to as a generator matrix of the code C.

Definition 2.2(b) Generator Matrix: A matrix $G \in \mathbb{F}_q^{n \times k}$ is a generator matrix for C if its k rows span C.

Remark: Why various texts differ it is common for vectors in code, code vectors, to be rows in a matrix.

Definition 2.2(c) Hamming Distance: Let there exist two code vectors in C x, y where both x, y are length n and defined over the same alphabet of order q. Then distance between x and y is defined as $\Delta(x, y) = \sum_{i=0}^{n} x_i \neq y_i$.

Furthermore, we can define the fractional hamming distance as $\delta(x, y) = \frac{\Delta(x,y)}{n}$.

Definition 2.2(d) Hamming Weight: Defined as $wt(C_i)$ of a code vector C_i is the number of nonzero components of the code vector. The Hamming weight of the entire code $wt(C)$ is the minimum weight of all non-zero vectors in the code.

Remarks: Due to linearity the Hamming weight of the code is the weight of the smallest non-zero code vector in the generator matrix G. Additionally, for any $x, y \in C$ $wt(x - y) = \Delta(x - y)$.

Definition 2.2(e) Minimum Distance: For any any two code vectors $x, y \in C$ the minimum distance of the code, denoted $d(C) = min\{d(x, y)|x, y \in C, x \neq y\}$.

Definition 2.2(f) Code Rate: The rate of a code is defined as the the proportion of useful information sent over a channel out of the total amount of information sent. This is simply k/n. For ever k bits of information sent n total bits are then generated as part of the encoding process. $k - n$ bits is the redundant part of the message sent over the channel.

Theorem 2.2(a): Let C be a linear code and let $wt(C)$ be the smallest of the weights of the non-zero code vectors of C. Then $d(C) = wt(C)$. [12]
Proof: There exists code vectors x, y such that $d(C) = \delta(x, y)$. Then by the fact, $wt(x - y) = \Delta(x - y)$,

$$d(C) = wt(x - y) \geq wt(C)$$

since $(x - y)$ is a code vector of C. We can then apply this statement for the zero code vector and the smallest non-zero code vector, $x \in C$ such that.

$$wt(C) = wt(x) = \Delta(x, 0) \geq d(C)$$

Having defined some key terms such as hamming distance, weight, generator matrix, and size of the code we can now introduce some simple notation. The basic characteristics of a code. $[n, k, d]$ describes the length of the code vectors, the size of the code, and the minimum distance of said code. Often this abbreviated to $[n, k]$ and d is omitted.

Linear codes over the same field $\mathbb{F}_q^n$ can be considered equivalent if one can be obtained over two operations.

(1) Permutation of the positions of the code
(2) Multiplication of the symbols appearing in a fixed position by a non-zero scalar.

Theorem 2.2(b): Let G be a generator matrix of an $[n, k]$ code C. By performing permutations of positions and multiplication on fixed position symbols by non-zero scalars we can put a matrix into standard form.

$$[I_k|A]$$

where I_k is the $k \times k$ identity matrix and A is a $k \times (n - k)$ matrix.
[12]

Proof: For purposes of brevity the proof this theorem is available on page 51 of *A First Course in Coding Theory* by Raymond Hill.

Linear codes are primarily used in the construction of error correcting codes. The foundation upon which error correction and detection is built is the

Hamming distance. The Hamming distance is a well defined metric space, although this is not proved in this paper, and is the foundation of what is called nearest neighbor decoding.

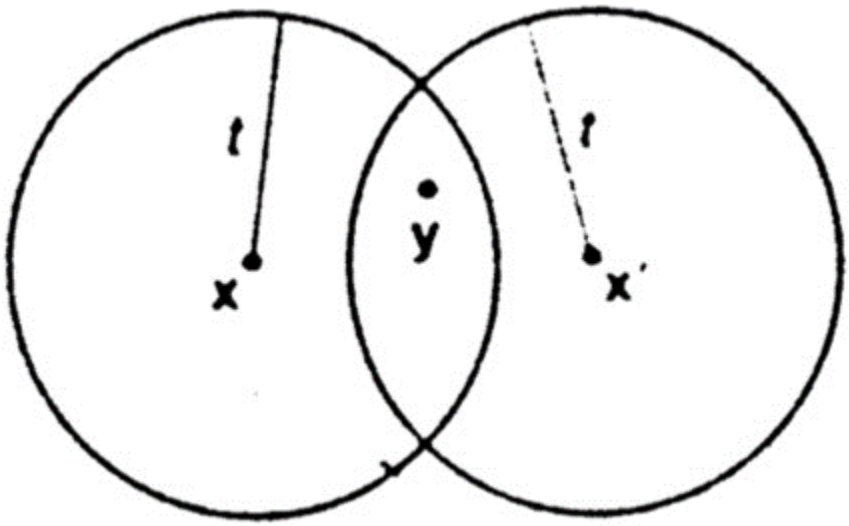
Theorem 2.2(b):

(1) A code C can detect up to s errors in any code vector if $d(C) \geq s+1$.
(2) A code C can correct up to t errors in any codeword if $d(C) \geq 2t+1$. [12]

Proof:

(1) Suppose $d(C) \geq s+1$. Suppose a code vector $\boldsymbol{X}$ is transmitted and s or fewer errors are introduced. Then the received vector cannot be a different code vector and so the rrors can be detected.
(2) Suppose $d(C) \geq 2t+1$. Suppose a code vector $/bX$ is transmitted and the code vector $\boldsymbol{Y}$ is received in which t or fewer errors have occurred, so that $d(\boldsymbol{X}, \boldsymbol{Y}) \leq t$. If $\boldsymbol{X'}$ is any code vector other than $\boldsymbol{X}$, then $d(\boldsymbol{X'}, \boldsymbol{Y}) \geq t+1$. For otherwise, $d(\boldsymbol{X'}, \boldsymbol{Y}) \leq t$, which implies, by the triangle inequality, that $d(\boldsymbol{X}, \boldsymbol{X'}) \leq d(\boldsymbol{X}, \boldsymbol{Y}) + d(\boldsymbol{X'}, \boldsymbol{Y}) \leq 2t$. This contradicts $d(C) \geq 2t+1$. So $\boldsymbol{X}$ is the nearest code vector to $\boldsymbol{Y}$ and nearest neighbor decoding corrects the errors.

Figure 2.2(a):



[12]

The process of encoding messages in code C is extremely straightforward. There exists q^k messages in C. Let $\mathbf{U}$ be code vector in C such that $\mathbf{U} = U_1U_2U_3 \ldots U_k$. We simply apply standard matrix multiplication on the generator matrix G where $G = c_1, c_2 \ldots, c_k$

$$\mathbf{U}G = \sum_{i=1}^{k} U_i C_i$$

If the generator matrix is in its standard form then

$$\mathbf{U}G = x_1x_2 \ldots x_kx_{k+1} \ldots x_n$$

and $x_i = u_i$, $1 \leq i \leq k$ are the bits containing the encoded message and from the $k+1$ bit and beyond we have the redundant bits previously discussed. The redundant check digits are expressed in the form.

$$x_{k+i} = \sum_{j=1}^{k} a_{ji}u_j, \qquad 1 \leq i \leq n-k$$

The process of decoding a code vector requires defining several terms first.

Definition 2.2(f) Dual of The Code: $C^{\perp}$ denotes the dual of a code. Let the dual of code be

$$C^{\perp} = \{x \in \mathbb{F}_q^n \mid \sum_{i=1}^{n} x_i c_i = 0 \; \forall \, C_i \in C\}$$

[12]

Definition 2.2(g) Parity Check Matrix: If we assume that C has a generator matrix in the form $G = [I_K|A]$ Then there exists a generator for for $C^{\perp}$ of the form

$$H = [-A^T|I_{n-k}].$$

Remark: The parity check matrix gives us the parity check equation. Where $(-A^T)^T$ represent the the digits in each code vector which add to zero under $(\mathbb{F}_q)^n$. For example

$$H = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

tells us that in the code containing H the digits $C_3 + C_4$ and $C_1 + C_2$ must add to zero.

[12]

Definition 2.2(g) Syndrome Decoding: The syndrome of a codeword is defined as

$$S(y) = yH^T$$

[12]

where y is some received message vector. Right multiplication of the message vector gives us the syndrome of the message vector received. Only message vectors with $S(y) = 0$ are in the code.

Individuals with knowledge of group theory will immediately recognize that that vectors with the same syndrome are in the same coset as each other.

The coset leader is the vector of minimum hamming weight in the coset. For example the coset leader of the coset represented by $S(y) = \overline{0}$ will be the zero vector of the generator matrix. When an error vector y is received we can conduct nearest neighbor decoding through the process of calculating $S(y)$ and finding the coset leader. We then subtract the coset leader from y, which within the limits maximum error detection and correction, will give us the intended code word sent.

2.3. **Construction of cyclic codes.** We start by defining a cyclic code.

Definition 2.3(a) Cyclic Codes: A linear code of length n is cyclic if it is invariant under a cyclic shift. So we see that

$$c = (c_0, c_1, c_2, \ldots, c_{n-2}, c_{n-1}) \in C$$

$$\Longleftrightarrow$$

$$\widetilde{c} = (c_{n-1}, c_0, c_1, c_2, \ldots, c_{n-2}) \in C$$

If $\mathbb{F}^n$ is a field such that $n \geq 3$ then there will always be trivial examples of a cyclic code present in the field.

(1) A length n zero dimensional code consisting of the all zero code word. This code lacks any information.
(2) A length n one dimensional code, known as the repetition code

(3) An n length code with dimension $(n-1)$ consisting of all vectors $(c_0, c_1, c_2, \ldots, c_{n-2}, c_{n-1})$ such that $\sum_i C_i = 0$. This is the single parity check code.

(4) A length n dimension n. This is the no parity code.

[2]

It may appear that appearances of cyclic codes are arbitrary with the exception of the trivial examples listed above. This is not true. In fact cyclic codes have algebraic structures.

Let $R = \mathbb{F}[x]/(x^n - 1)$ be the polynomial quotient ring over the finite field $\mathbb{F}_q^n$. If we have some polynomial in R such that the code vector $(c_0, \ldots, c_{n-1})$ maps to the polynomial $c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$.

For example in a code over $\mathbb{F}_2^3$

$$(0) = (000)$$

$$(1 + x) = (110)$$

$$(x + x^2 = (011)$$

$$(1 + x^2) = (101).$$

Note that over any field $(x^n - 1) = (x - 1)(x^{n-1} + x^{n-2} + \cdots + x + 1)$. Also, $(x^n - 1) = 0$ in the polynomial ring and is an ideal of the ring of polynomials. As discussed earlier when the ideal is generated by an irreducible polynomial then the quotient ring is a field. In this specific case the factor of $(x^n - 1)$

will be an irreducible polynomial which generates the ideal and is thus referred to as the generator of the cyclic code. [12]

Theorem 2.3(a): If we define the generator function as $c(x)$ then $c^r(x)$ is the right cyclic shift defined by $c^r(x) = x \cdot c(x)\ mod\ (x^n - 1)$

Proof:

$$\begin{aligned} c^r(x) &= x \cdot c(x)\ mod\ (x^n - 1) \\ &= x(c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}) \\ &= xc_0 + c_1 x^2 + \cdots + c_{n-2}x^{n-1} + c_{n-1}x^n \\ &= c_{n-1} + xc_0 + c_1 x^2 + \cdots c_{n-3}x^{n-2} + c_{n-2}x^{n-1} \end{aligned}$$

It is important to remember that in the polynomial ring modular $(x^n - 1)$ $x^n = 1$ such that the cyclic shift can now occur.

Theorem 2.3(b): Let the code $C \neq 0$ be a cyclic code and of length n over $\mathbb{F}$ then $g(x)$ is the generator function of minimal degree that is uniquely determined and $C = \{g(x)q(x)| \in \mathbb{F}_{n-d}\}$ where d denotes the degree of the generator function $g(x)$. [11]

Proof:

Let $c(x) = g(x)q(x) + r(x)$. Where $c(x)$ is any polynomial in the code. By equality $c(x) - g(x)q(x) = r(x)$. If $r(x)$ was non zero then it would also be in C and would have degree less than d. However, this contradicts the minimality of $g(x)$ and so $r(x) = 0$ This proves that $g(x)$ is the unique minimal generator in C.

Theorem 2.3(c): Let the check polynomial of a cyclic code be denoted by $h(x)$. Furthermore, $x^n - 1 = h(x) \cdot c(x) = 0$ under $\mathbb{F}_q^n$ for any $c(x)$ in the code. [11]

Proof: Again we refer to the division algorithm. For some $s(x)$ with degree less than d

$$x^n - 1 = h(x) \cdot g(x) + s(x)$$

$$-h(x) \cdot g(x) = s(x)\, mod(x^n - 1)$$

If $s(x)$ is not equal to zero then there exists a contradiction to the minimality of $g(x)$. So therefore $s(x) = 0$ and $h(x)$ is the polynomial which negates any code vector in C. The polynomial $h(x)$ is referred to as the check polynomial.

2.4. **Prominent examples of cyclic codes.** Having defined the basic construction of cyclic codes we will now look at some actual example. First we will consider the famous Hamming code. Hamming codes are most easily defined in terms of their parity check matrix.

Definition 2.4(a) Hamming Code: Let H denote the parity check matrix such that H is a $r \times (2^r - 1)$ size matrix. This matrix H is the parity check matrix for the binary Hamming code denoted $\mathrm{Ham}(r, 2)$. [12]

Remark: While we are only discussing the binary Hamming codes in this text there is no need to restrict Hamming codes to binary alphabets. To describe other codes we simply replace $\mathrm{Ham}(r, 2)$, with $\mathrm{Ham}(r, q)$ such that q is the size of the alphabet.

Additionally we note some other facts about $\mathrm{Ham}(r, 2)$ codes.

(1) $\mathrm{Ham}(r, 2)$ has length $n = 2^r - 1$ and dimension $k = n - r$.

(2) Hamming codes are perfect codes. This means that the code vectors with covering radius of t errors fill the entire vector space with no overlap. Alternatively the vector space is filled with the maximum number of code words without violating the minimum distance between codes.

(3) $\mathrm{Ham}(r, 2)$ has a minimum distance of three. This is easily proved by considering the case of a code vector whose $wt(x) = 1$. We quickly see that the parity check matrix would have an all zero vector. This would violate the definition of the parity check matrix. Showing that there are no code vectors of $wt(x) \leq 3$ is an extension of this proof.

(4) Decoding with $\mathrm{ham}(r, 2)$ is very simple. Since the code is perfect the coset leaders are the 2^r vectors of $wt \leq 1$ in the space. This means that $s(y)$ will, in binary representation, give the position of a single error digit in the code vector. This, in the case of single error detection and correction allows for fast and low bandwidth error correction.

It is not immediately apparent that the $\mathrm{Ham}(r, 2)$ codes are in fact cyclic. They, in fact, are cyclic. This property allows us to describe a hamming code with a single polynomial function.

Definition 2.4(b): Cyclotomic Coset Let n be relatively prime to q. The cyclotomic coset of q modulo n containing i is defined by:

$$C_i = \{i \cdot q^j (mod\, n)\mathbb{Z}_n \mid j = 0, 1, \dots .\}$$

A complete set of representatives of equivalence classes under $\mathbb{Z}_n$ is referred to as the complete set of representatives of cyclotomic cosets.

Theorem 2.4(a): Let α be a generator element of $\mathbb{F}_q^r$. Then the minimal polynomial of α^i with respect to $\mathbb{F}_q$ is.

$$M^{(i)}(x) = \prod_{j \in C_i} (x - \alpha^j)$$

where C_i is the unique cyclotomic coset of q modulo $q^r - 1$ containing i. [3]

Proof: Let

$$M^{(i)}(x) = a_0 + a_1 x^1 + \cdots + a_{r-1} x^{r-1} + a_r x^r$$

Then by raising each coefficient to the power of q we obtain

$$a_0^q + a_1^q x^1 + \cdots + a_{r-1}^q x^{r-1} + a_r^q x^r = \prod_{j \in C_i} (x - \alpha^{qj}) = \prod_{j \in C_{qi}} (x - \alpha^j) = \prod_{j \in C_i} (x - \alpha^i) = M^{(i)}(x)$$

Which proves $M^{(i)}(x)$ is a polynomial over $\mathbb{F}_q$.

Let us show $M^{(i)}(x)$ is minimal. First, since α is a generator element the $\alpha^k \neq \alpha^j$ for $k \neq j$. Next let $f(x) \in \mathbb{F}_q[X] \mid f(a^i) = 0$. If we then construct $f(x) = f_0 + f_1 x + \cdots + f_n x^n$ then for any $j \in C_i$ there exists an integer l such that $j \equiv iq^l \pmod{q^r - 1}$. Hence,

$$f(\alpha^j) = f(\alpha^{iq})^l = f(\alpha^i)^{q^l} = 0$$

Which shows that $M^{(i)}(x)$ is a divisor of $f(x)$.

Remark 1: $a_0^q + a_1^q x^1 + \cdots + a_{r-1}^q x^{r-1} + a_r^q x^r = \prod_{j \in C_i} (x - \alpha^{qj})$ by way of the freshman's dream property of finite fields which is referenced in section

2.1. Also, by definition of the cyclotomic coset $C_i = C_{qi}$. This will allow us to find the third product of the proof. Lastly, because $a_k = a_{q^k}$, for all $0 \leq k \leq r$, all a_k are elements of $\mathbb{F}_q$.

Thereom 2.4(b): The ham$(r, 2)$ code is equivalent to a cyclic code. [2]

Proof: Let α be the generator polynomial of the field $\mathbb{F}_2^r$. By the previous theorem the minimal polynomial of α is $f(x) = M^1(x) = \prod_{j \in C_i}(x - \alpha^i)$ where C_1 is the cyclotomic coset of 2 modulo $2^r - 1$. By our earlier definition of cyclotomic cosets $C_1 = \{1, 2, 3, 2^2, 2^3, \ldots, 2^{r-2}, 2^{r-1}\}$, such that the degree of $f(x) = M^1(x)$ is r, see theorem 2.4(a), and is irreducible. Furthermore,

$$\{0, 1, \alpha, \ldots, \alpha^{2^r-2}\} = \mathbb{F}_{2^r} = \mathbb{F}_2[x]/f(x) = \mathbb{F}_2[\alpha] = \{\alpha_0 + \alpha_1\alpha + \alpha_2\alpha^2 + \cdots + \alpha_{r-1}\alpha^{r-1} | \alpha_i \in \mathbb{F}_2\}$$

Let $\alpha_0 + \alpha_1\alpha + \alpha_2\alpha^2 + \cdots + \alpha_{r-1}\alpha^{r-1} \in \mathbb{F}_{2^r} = \{0, 1, \alpha, \ldots, \alpha^{2^r-2}\}$. Be associated to the column vector

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{r-1} \end{bmatrix} \in \mathbb{F}_2^r$$

Let $n = 2^R - 1$. The $r \times n$ matrix

$$H[1, \alpha, \alpha^2, \cdots, \alpha^{n-1}]$$

is the parity check matrix for $C = Ham(r, 2)$. since its columns are the distinct nonzero vectors of $\mathbb{F}_2^r$. Our code vector multiplied by the parity check matrix H must be equal to 0 so that.

$$\{\mathbf{C} = \mathbf{U}(x) \in \mathbb{R}_2^n | \mathbf{U}(\alpha) = 0 \in \mathbb{F}_2[x]/f(x)\} = \{\mathbf{x} \in \mathbb{F}_2^n | H\mathbf{x}^t = 0\}$$

Finally, if $\mathbf{U}(x) \in C$ and $r(x) \in \mathbb{R}_2^n$, then $r(\alpha)\mathbf{u}(\alpha) = r(\alpha)0 = 0 \in \mathbb{F}_2[x]/f(x)$, noting that $\alpha^n = 1$, so r(x)$\mathbf{u}$(x) is an element of C. The fact that we are working in a field implies C is cyclic.

We now define a 3 step process which allows us to find a cyclic code equivalent to the binary $[2^r - 1, 2^r - 1 - r]$ hamming code.

Definition 2.4(c): A polynomial g(x) is primitive if its coefficients are in $GF(p) = \mathbf{Z}/p\mathbf{Z}$, has degree m, and has a root α in $GF(p^m)$ such that $\{0, 1, \alpha, \alpha^2, \alpha^3, \cdots, \alpha^{p^m-2}\}$ is the entire finite field. A corollary to this is that α is a root of unity in the finite field. [2]

Step 1: Let $m = 2^r - 1$, and write the cyclotomic factorization $x^m - 1 = \prod_{d|n} \Phi_d(x)$, such that $0 < d < n$, in $\mathbb{Z}[x]$.

Step 2: Reduce this factorization mod 2, and factor into irreducibles in $\mathbb{F}_2[x]$. Let $g(x)$ be a primitive irreducible factor in this factorization. It will have degree n.

Step 3: Let $g(X)$ be the generator polynomial such that $< g(x) >$ Is the cyclic code in $\mathbb{F}_2[x]/ < x^m + 1 >$.

3. **Lattices and cryptographic applications**

In the following sections we will discuss properties of integer lattices, computationally complex problems related to lattices, and how lattices and cyclic codes combine to create a cryptographic algorithm called NTRU.

3.1. **Definitions and basic properties of lattices.**

First we will define lattices from two standpoints.

Definition 3.1(a): A lattice is a discrete additive subgroup of $\mathbb{R}^n$, i.e., it is a subset $\Lambda \subseteq \mathbb{R}^n$ which satisfies the following:

- Λ is closed under addition and subtraction. It is a subgroup.
- The lattice is discrete. There is an $\epsilon > 0$ such that for any two lattice points $\boldsymbol{x} \neq \boldsymbol{y} \in \Lambda$ are at least the distance $\|\boldsymbol{x} - \boldsymbol{y}\| \geq \epsilon$. [17]

So based off this initial definition of a lattice we see that the set $\mathbb{Z}^n$ is a lattice. This set is closed under addition and subtraction as well as having a minimum euclidean distance of at least 1 between any two vectors.

An alternative definition of a lattice is as follows.

Definition 3.1(b): Given n linearly independent vectors $b_1, b_2, \cdots, b_n \in \mathbb{R}^m$, the lattice generated by them is defined as

$$\mathcal{L}(b_1, b_2, \cdots, b_n) = \{\sum x_i b_i | x_i \in \mathbb{Z}^n\}$$

The vectors $b_1, b_2, \cdots, b_n$ are defined as the basis of the lattice.

If we define the matrix B as the $m \times n$ matrix whose columns are $b_1, b_2, \cdots, b_n$ then the lattice generated by B is

$$\mathcal{L}(B) = \mathcal{L}(b_1, b_2, \cdots, b_n) = \{Bx \mid x \in \mathbb{Z}^n\}$$

[19]

Definition 3.1(c): We can define the rank of the lattice as the n columns of the above matrix. If the matrix has dimension m and $n = m$ then the lattice is called a full rank lattice.

Definition 3.1(c): The span of a lattice $\mathcal{L}$ is the linear space spanned by its vectors.

$$span(\mathcal{L}(B)) = span(B) = \{By \mid y \in \mathbb{R}^n\}$$

[19]

Definition 3.1(d): The fundamental parallelepiped for any lattice of basis B is defined as

$$\mathcal{P}(B) = \{Bx \mid x \in [0,1)^n\}$$

[13]

Which essentially is the space spanned by its basis when restricting the scalar of the basis to a domain from $[0, 1)$.

An interesting property of the fundamental parallelepiped, $\mathcal{P}(B)$ is that if the lattice is full rank, i.e. the matrix B has $n = m$ columns and rows, is that $\mathcal{P}(B)$ will tile $\mathbb{R}^n$. More formally we write this as.

$$\mathbb{R}^n = \{\mathcal{P}(B) + x \mid x \in \mathcal{L}(B)\}$$

[13]

From this it becomes clear that translations of $\mathcal{P}(B)$ by all the vectors in $\mathcal{L}(B)$ will cover the space $\mathbb{R}^n$. Naturally, if the lattice $\mathcal{L}(B)$ is of full rank, B

is an $n \times n$ matrix, then the intersection of the lattice $\mathcal{L}(B)$ with $\mathcal{P}(B)$ is the empty set. The proof of this fact is on page two of [13].

Theorem 3.1(a): Let V be an r-dimensional subspace of $\mathbb{R}^n$, and let Γ be a discrete co-compact subgroup of V. Then Γ is a lattice of rank r in $\mathbb{R}^n$.

Remark: We will briefly define co-compactness as that if there exists some discrete subgroup $\boldsymbol{H}$ of $\boldsymbol{G}$ which is co-compact. Then there must also exists a compact subgroup $\boldsymbol{K}$ such that $\boldsymbol{HK} = \boldsymbol{G}$..

The previous theorem is proved on page 14 in [7]. For sake of brevity the full proof is omitted. However, it is worth explaining why this theorem is so important. Clearly we can construct a basis of linearly independent vectors $\{\alpha_1, \cdots, \alpha_r\}$ which span $\mathbb{R}$. We must construct our lattice basis, $\{b_1, b_2, \cdots, b_r\}$, in such a manner in such a way that it specifically spans $\mathbb{Z}$ in r dimensions. To do this we must first construct the linearly independent basis for Γ. Then the proof must show that the span of $\mathbb{Z}$ in r dimensions contains Γ, which is fairly obvious. Lastly, we show that Γ contains the span of $\mathbb{Z}$ in r dimensions. The main point of the above theorem is that an integer lattice is a unique object quite different from a full rank matrix due to it being a discreet subgroup of a vector space in $\mathbb{R}^n$.

Another interesting property of lattices is that a basis for a lattice is not necessarily unique. We now show why this is so. This fact is highly important in the use of lattices in cryptography.

Given two basis B_1, B_2, we must determine if they generate an equivalent lattice such that $\mathcal{L}(B_1) = \mathcal{L}(B_2)$. It is necessary to introduce more definitions to do so.

Definition 3.1(e): A matrix U in $\mathbb{Z}^{n\times n}$ is called unimodular if $det\, U = \pm 1$ [19]

Lemma 3.1(a): If U is unimodular, then U^{-1} is also unimodular, and $U^{-1} \in Z^{n\times n}$.

Proof: Suppose A is unimodular. Then by generalized Cramer's rule we have:

$$A^{-1}(I,J) = \pm\frac{A(J^c, I^c)}{detA}$$

,

where $A(J^c, I^c)$ is the cofactor matrix. Since A is unimodular by definition $detA = 1$. So we can restate the above as

$$A^{-1}(I,J) = \pm A(J^c, I^c)$$

Which essentially states that $A^{-1} = Adj(A)$. Where $Adj(A)$ is the adjugate of A. Finally, there exists a relationship, which we will not prove for brevity sake, that states

$$det(Adj(A)) = det(A)^{n-1}$$

. Since our matrix is unimodular it is clear $det(A)^{n-1} = 1$ Which shows that $det(Adj(A)) = 1$ and we are done.

Theorem 3.1(b): Let Λ be a lattice of rank n in $\mathbb{R}^n$, and let A be a basis matrix for Λ. Then B is another basis matrix for Λ if and only if there exists an $N \times N$ integral matrix U with $detU = 1$ such that

$$B = UA$$

[7]

Proof: First note that if

$$B = UA$$

then

$$\boldsymbol{b_i} = \sum_{j=1}^{N} u_{ij} a_j$$

and since B is also a basis matrix then there must exist some unimodular square integer matrix W such that

$$\boldsymbol{a_i} = \sum_{j=1}^{N} w_{ij} b_j$$

such that

$$B = UA = UWB$$

Which implies $UW = I_N$ which further implies that $W = U^{-1}$. Note that we already showed that if U is a unimodular square integer matrices then $det(U) = det(U^{-1}) = \pm 1$. From this we can see that

$$det(A)det(U) = det(U^{-1})det(B) \neq 0$$

which implies that the columns of B are linearly independent. Finally, if

$$B = U^{-1}A$$

the column vectors of A are in the span of the column vectors of B, which shows that

$$\Lambda = span_{\mathbb{Z}}\{b_1, b_2, \cdots, b_n\}$$

. Hence B is a basis for $Lambda$ and our proof is complete.

Corollary: From the above proof we see that

$$|det(a)| = |det(b)|$$

Remark: For the full version of this proof refer to page 15 of [7].

While there are many interesting properties and theorems concerning lattices we will conclude this section with a definition regarding the determinant.

Definition: Let $\Lambda = \mathcal{L}(B)$ be a lattice of rank n. Then $det(\Lambda) = volP(B)$, i.e. the determinant is the n dimensional volume of the fundamental parallelepiped. [19]

In the general case define $det(\Lambda) = \sqrt{B \cdot B^t}$. However since we are restricting ourselves to full rank lattices we state that $det(\Lambda) = |det(B)|$.

3.2. **Successive minima and Minkowski's theorems.** We start the following subsection with definitions which will be necessary for understanding upper and lower bounds on successive minima. We begin by defining the Gram-Schmidt Orthogonalization Process.

Definition 3.2(a): Given any n set of linearly independent vectors we can create a set of n orthogonal vectors through the following process.

(1) First we must define the projection operator as the algorithm for projecting the vector $\vec{v}$ onto a new orthogonal vector $\vec{u}$ this is defined

as

$$\text{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{v}, \mathbf{u} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u},$$

(2) Gram-Schmidt process on the k_{th} vector is as follows.

$$\mathbf{u}_k = \mathbf{v}_k - \sum_{j=1}^{k-1} \text{proj}_{\mathbf{u}_j}(\mathbf{v}_k), .$$

(3) Then we obtain an orthonormalized Gram-Schmidt basis when normalization is applied to each vector in the usual way.

$$\mathbf{e}_k = \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}$$

The span of the orthonormalized basis is equal to the span of the basis of the original lattice $\mathcal{L}(B)$. However, the orthonormal basis need not be a basis for $\mathcal{L}(B)$ and generally is not. We finish our discussion of the Gram-Schmidt process with a formal definition

Definition 3.2(b): Let $\boldsymbol{B} \in \mathbb{R}^{d \times n}$ be a basis then

$$det(\mathcal{L}(B) = vol(P(B)) = \prod_i \|\mathbf{b}_\mathbf{i}^*\|$$

[17]

Where $\|\boldsymbol{b}_\boldsymbol{i}^*\|$ is the i_{th} orthogonal Gram-Schmidt vector. From this we have a convenient way of calculating the determinant of the lattice.

Definition 3.2(c): For any lattice basis of $\boldsymbol{B} \in \mathbb{R}^{d \times n}$

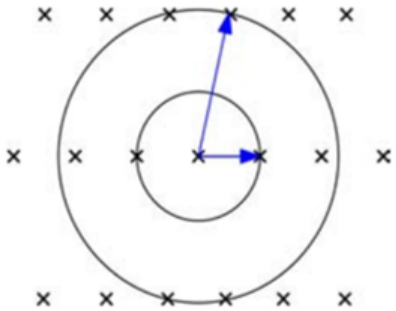
$$det(\mathcal{L}(B) = \sqrt{\boldsymbol{B}^{\boldsymbol{T}} \boldsymbol{B}}$$

[17]

Definition 3.2(d): The i_{th} successive minimum of lattice $\Lambda, \lambda_i(\Lambda)$, is defined to be $inf\ \{\ r \mid dim(span(\lambda \cap \overline{B}(0,r))) \geq i\}$ [13]

More colloquially we can say that λ_i is the following: it is the smallest r such that the lattice points inside a ball of radius r span a space of dimension i. Below we see an example in $\mathbb{R}^2$.

Figure 3.2(a):



[13]

However, defining the successive minima does little to help us understand what the upper and lower bounds on them are. More importantly there are key questions left unanswered regarding on how one obtains a basis of successive minima.

The question of the lower bound is the least complicated. We can use our Gram-Schmit process to produce an orthogonal basis. This will then provide us a lower bound for the successive minima.

Theorem 3.2(a): Let B be a rank-n lattice basis and let B^* be its Gram Schmidt orthogonalization. Then we have that

$$\lambda_1(\mathcal{L}(B)) \geq \min_{1,\cdots,n} \|B^*\| > 0$$

[22] [19]

Proof: Let $j \in \{1, \ldots, n\}$ be the largest index such that $x_j \neq 0$. Then,

$$|\langle Bx, b_j^* \rangle| = |\langle \sum_{i=1}^{n} x_i b_i, b_j^* \rangle| = \sum_{i=1}^{n} x_i |\langle b_i, b_j^* \rangle| = |x_j| \langle b_j^*, b_j^* \rangle = |x_j| \cdot \left\| b_j^* \right\|^2$$

where we used that for all $i < j, \langle b_i, b_j^* \rangle = 0$ and that $\langle b_j, b_j^* \rangle = \langle b_j^*, b_j^* \rangle$. On the other hand, $|\langle Bx, b_j^* \rangle| \leq \|Bx\| \cdot \left\| B_j^* \right\|$, concluding that

$$\|Bx\| \geq |x_j| \left\| b_j^* \right\| \geq \left\| b_j^* \right\| \geq \min \|b_i^*\|$$

We must now consider the upper bounds on successive minima. To do this however we must first present two theorems known as Blichfeld's Theorem and Minkowski's Convex Body Theorem. Let us begin with Blichfeldt's theorem.

Theorem 3.2(b): For any full-rank lattice $\Lambda \subseteq \mathbb{R}^n$ and set $S \subseteq \mathbb{R}^n$ with $vol(S) > det(\Lambda)$ there exists two nonequal points $z_1, z_2 \in S$ such that $z_1 - z_2 \in \Lambda$. [19]

Proof: Let B be a basis for the lattice $\mathcal{L}$. Define $f : \mathbb{R}^n \mapsto P(B)$ as follows: $f(\sum x_i b_i) = \sum (x_i - \lfloor x \rfloor) b_i$. First, note that $\sum x_i b_i - f(\sum x_i b_i) = \sum \lfloor x_i \rfloor b_i \in \mathcal{L}$. Now consider the following two cases:

Case 1: If $\exists\ x, y \in\ S\ s.t.\ f(x) = f(y)$. Then $x - y = (x - f(x)) - (y - f(y))$. But as noted, $x - f(x) \in \mathcal{L}$ and $y - f(y) \in \mathcal{L}$. Therefore $x - y \in \mathcal{L}$.

Case 2: Assume there are no collisions. Les $S = \cup_{x \in \mathcal{L}} S_x$. Define $S_x^* = S_x - x$. By definition, $S_x^* \subseteq P(B)$. Also, $vol(s) = \sum vol(s_x)$ and $vol(S_x^*) = vol(S_x)$. Therefore $vol(s) = \sum vol(s_x) = \sum vol(S_x^*)$. But since we assume that we do not have any collisions, then for $x, y, S_x^* \cap S_y^* = \emptyset$. And so,

$$vol(s) = \sum vol(S_x^*) = \sum vol(\underset{x \in \mathcal{L}}{\cup} S_x^*) \leq vol(P(B)) = det(\mathcal{L})$$

Therefore, $vol(s) \leq det(\mathcal{L})$ which contradict the assumption of case 2. [22].

Theorem 3.2(c): (Minkowski's Convex Body Theorem) For all full-rank lattice $\mathcal{L}$, and a convex centrally symmetric set S with $vol(s) > 2^n det(\mathcal{L}), S$ contains a non-zero lattice point. [22].

Proof: Let $S^* = x/2 : x \in S$. Then,

$$vol(s^*) = 2^{-n} vol(s) > det(\mathcal{L}).$$

[22]

Therefore, by the Blichfeldt theorem $\exists x, y \in S^* \mid x - y \in \mathcal{L}$. We will show that $x - y \in S$. Now, $2x \in S$ and $2y \in S$ by the construction of S^*.
Therefore, $-2y \in S$. and $x - y = \frac{2x-2y}{2} \in S$.

Finally, we have the building blocks for a theorem which gives us an upper bound on the first successive minima.

Theorem 3.2(d): For every full rank lattice $\mathcal{L}$:

$$\lambda_1 \mathcal{L} \leq \sqrt{n} \cdot det(\mathcal{L})^{1/n}$$

[22]

Proof: Let $S = (0, \lambda_1(\mathcal{L}))$, where $\mathcal{B}(x, r)$ is an n-dimensional open ball of radius r centered at x. This ball contains an n dimension cube of length $\frac{2r}{\sqrt{n}}$. Therefore,

$$vol(\mathcal{B}(0, r)) \geq (\frac{2r}{\sqrt{n}})^n$$

.

Therefore, we get $vol(\mathcal{B}(0, \lambda_1(\mathcal{L}))) \geq (\frac{2\lambda_1\mathcal{L}}{\sqrt{n}})^n$. But from theorem 3.2(c) (Minkowski's convex body theorem) and the fact that the ball is open and hence contains no non-zero lattice points, we get

$$\lambda_1(\mathcal{L})) \geq \sqrt{n} \cdot det(\mathcal{L})^{1/n}$$

We conclude this section by giving an upper bound for any successive minima. This is Minkowski's second theorem. From this theorem we now have a lower and upper bound for any λ_i. This is as follows:

Theorem 3.2(e): For all full rank lattices $\mathcal{L}$.

$$\prod_{i=1}^{n} \lambda_i(\mathcal{L})^{1/n} \leq \sqrt{n} \cdot (det(\mathcal{L}))^{1/n}$$

Remark: The proof of this theorem is simply too long to include in this work. However, for the full proof refer to Martin Henk's proof of Minkowski's second theorem. Minkowski's own theorem is widely viewed as overly long and difficult to understand.

To conclude this section we must note that while we have defined upper and lower bounds for successive minima we have not determined the successive

minima themselves. This is a uniquely hard problem. This leads us into our next two sections.

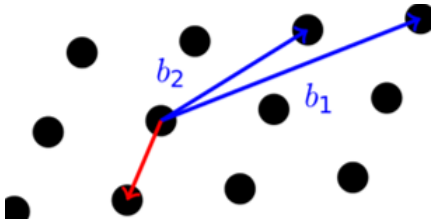
3.3. **Lattice problems.** In order to understand how lattices can be used to encrypt and decrypt messages we must explain several problems based off finding successive minima. We start off with the definition of shortest vector problem (SVP).

Definition 3.3(a): Given a lattice Λ find $\lambda_i \neq 0 \in \Lambda$ such that for any other lattice point $\lambda_k \in \Lambda$ where $0 < k \leq n$ and $k \neq i$ we have that

$$\|\lambda_k\| \geq \|\lambda_i\|$$

From this we can see the that $\lambda_i = \lambda_1$, the shortest non-zero vector in the lattice. Visually we see that

Figure 3.3(a):

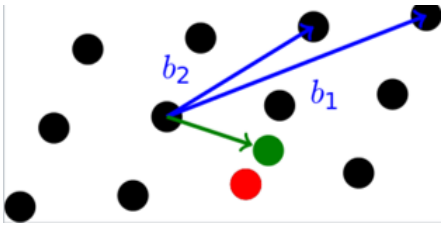


[20]

Where the red vector represents λ_i and the blue vectors represent the basis vectors of a lattice in two dimensions. In the following sections we will discuss why it is difficult to find this vector. We now discuss a similar problem called closest vector problem (CVP)

Definition 3.3(b): Given a lattice Λ and vector $\overline{y} \in \mathbb{R}^n$, find a $\lambda_k \in \Lambda$ such that $\|\overline{y} - \lambda_k\|$ is minimal.

Figure 3.3(b):



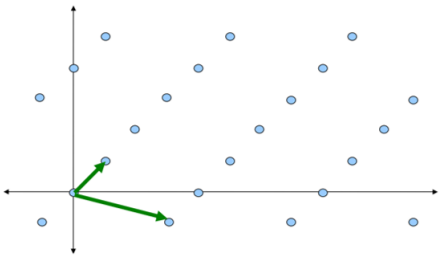
[20]

Where the green point is the vector $\overline{y}$ and the red point is λ_k.

Finally we define the Shortest Independent Vector Problem (SIVP) which is finding n independent short vectors. We now formally define this.

Figure 3.3(c): Given a basis $B \in \mathbb{Z}^{n \times n}$, find independent vectors $\{u_1, \cdots, u_n\}$ such that $\|u_i\| \leq \lambda_n$ for $i \in [n]$. [22]

Figure 3.3(c):



[16]

In the image above we see that we have found the shortest independent vectors up to $n = 2$. Over more dimension this problem becomes progressively more complex.

Although we have not adequately explained complexity at all, much less with rigor. It is not hard for the lay person to imagine how these problems would get progressively difficult as the dimension of your lattice increases.

3.4. **Basic computational complexity.** Alan Turing is viewed as the father of modern computer science. The Turing Machine, named after him, is a purely hypothetical construct which we will now define.

Definition 3.4(a): A Turing machine is specified by a finite alphabet Σ, a finite set of states K with a special element s, (the starting state), and a transition function $\delta : K \times \Sigma \mapsto (K \cup \{halt, yes, no\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$. It is assumed that Σ, K , $\{halt, yes, no\}$, and $\{\leftarrow, \rightarrow, -\}$ are disjoint sets, and that Σ contains two special elements $\triangleright, \sqcup$ representing the start and end of the tape, respectively. We require that for every $q \in K$, if $\delta(q, \triangleright) = (p, \sigma, d)$ then $\sigma = \triangleright$ and $d \neq \leftarrow$. In other words, the machine never tries to overwrite the leftmost symbol on its tape nor to move to the left of it. [14]

Quite simply the Turing machine reads a tape and proceeds left or right on the tape based off an instruction alphabet written on the tape. While this machine was hypothetical in its construction it is not far off from the function of modern computers.

Obviously we can use computers to solve certain problems faster than people. For example, calculators conduct multiplication for us and can do so at a much faster rate than people. However, speed, or efficiency, of the computer can depend greatly on the algorithm used. For example, we can tell a computer to multiply $a \cdot b$ by adding a to itself $b - 1$ times or through use of the usual grade school algorithm. We see that the grade school algorithm requires at most $2n^2$ steps, where n is the size of the input. The former method requires at least $n10^{n-1}$ steps. Using the more efficient grade school algorithm we would see common pocket calculators beating supercomputers

using the former repeat addition algorithm. This disparity of course holds only for sufficiently large input n.

It is important, when talking about efficiency, to try and rigorously define how much time an algorithm takes to solve some sort of problem. We define our first set of complexity class with two definitions.

Definition 3.4(b): Let $T : \mathbb{N} \mapsto \mathbb{N}$ be some function. We let $\boldsymbol{DTIME}(T(n))$ be the set of all Bolean (one bit output) functions that are computable in $c \cdot T(n)$-time for some constant $c > 0$. [1]

Definition 3.4(c): $\boldsymbol{P} = \cup_{c \geq 1} \boldsymbol{DTIME}(n^c)$ Where $\boldsymbol{P}$ the class $\boldsymbol{P}$. [1]

So we see that a problem solved in $\boldsymbol{DTIME}$, or deterministic polynomial time, might be solved in n^3, or n^5 step algorithm. From this we can say that the group of efficient algorithms for decision (boolean) problems are analogous to the class $\boldsymbol{P}$.

We now know that there are efficient algorithms for solving certain problems. Other problems, say a sudoko puzzle or the shortest route for a UPS delivery truck, have a verifiable solution. These problems with solutions may not have an efficient algorithm to solve the problem, only to verify the solution when given. The set of problems in the latter set are in what is called non-deterministic polynomial time, $\boldsymbol{NP}$ time.

Definition 3.4(d): A language $L \subseteq \{0,1\}^*$ is in $\boldsymbol{NP}$ if there exists a polynomial $p : \mathbb{N} \mapsto \mathbb{N}$ and a polynomial-time Turing Machine M such that for every $x \in \{0,1\}^*$,

$$x \in L \leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \; s.t. \; M(x,u) = 1$$

if $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$ then we call u a certificate for x (with respect to language L and machine M) [1]

Remark: We must define some terms from theoretical computer science. $\{0,1\}^n$ denotes strings of bits of length n. $\{0,1\}^*$ is the set of all strings. Finally, $\{0,1\}^{p(|x|)}$ is the string of length $p(|x|)$ This is a rigorous way of stating that if input x is in the language and we have a polynomial size solution that verifies x then the problem is in the $\boldsymbol{NP}$ class.

Theorem 3.4(a): $\boldsymbol{P} \subseteq \boldsymbol{NP} \subseteq \cup_{c \geq 1} \boldsymbol{DTIME}(2^{nc})$ [1]

Proof: First we show $\boldsymbol{P} \subseteq \boldsymbol{NP}$. Suppose $L \in \boldsymbol{P}$ is decided in polynomial time by a Turing Machine N. The $L \in \boldsymbol{NP}$ since we can take N as the machine M in definition 3.4(d) and $p(x)$ the zero polynomial such that u would be the empty string.

We now prove $\boldsymbol{NP} \subseteq \cup_{c \geq 1} \boldsymbol{DTIME}(2^{nc})$. If $L \in \boldsymbol{NP}$ and $M, p()$ are as in definition 3.4(d) then we L in time $2^{O(p(n))}$. by enumerating all possible u and and using M to check whether u is a valid certificate or witness fo rthe input x. The machine accepts if and only if such a u is ever found. Since $p(n) = O(n^c)$ for some $c > 1$ then this machine runs in $2^{O(p(n))}$ time. [1]

Remark: The above theorem is a fairly trivial result. It does not, for example, answer the most central question of complexity theory of whether $\boldsymbol{P} = \boldsymbol{NP}$. It is a commonly held belief that $\boldsymbol{P} \neq \boldsymbol{NP}$. Proving this fact is an extremely inefficient method of making one million dollars. Proving that $\boldsymbol{P} = \boldsymbol{NP}$ would probably cause the collapse of modern society.

We must now define the concept of a non-deterministic Turing Machine in order to define the class $\boldsymbol{NP}$ in a way which does not require $\boldsymbol{DTIME}$.

Definition 3.4(e): For every function $T : \mathbb{N} \mapsto \mathbb{N}$ and $L \subseteq \{0,1\}^*$, we say that $L \in \boldsymbol{NTIME}(T(n))$ if there is a constant $c > 0$ and a $cT(n)$ time Non Deterministic Turing Machine M such that for every $x \in \{0,1\}^*, x \in L \Leftrightarrow M(x) = 1$. [1]

Remark: The key difference between a deterministic Turing Machine and it's non-deterministic counterpart is that it has two transition functions δ_1, δ_2. The non-deterministic machine also has a special state, defined as q_{accept} in [1]. $M(x)$ for every input x in L will reach either q_{accept} or $M(x) = 0$

Theorem 3.4(b): $\boldsymbol{NP} = \cup_{c \in \mathbb{N}} \boldsymbol{NTIME}(n^c)$. [1] [1]

Proof: Suppose $p : \mathbb{N} \mapsto \mathbb{N}$ is a polynomial and L is decided by a non-deterministic Turing Machine N that runs in time $p(n)$. For every $x \in L$, there is a sequence of nondeterministic choices that makes N reach q_{accept} on input x. We can use this sequence as a certificate, or witness, for x. Note, this certificate has length $p(|x|)$ and can be verified in polynomial time by a deterministic machine, which checks that N would have entered q_{accept} after using these non-deterministic choices. Thus $L \in \boldsymbol{NP}$ According to definition 3.4(a)

In the other direction, if $L \in \boldsymbol{NP}$ according to definition 3.4(a), then we describe a polynomial time Non-Deterministic Turing Machine N that decides L. On input x, it uses the ability to make non-deterministic choices to write down a string u of length $p(|x|)$. The it runs the deterministic verifier M of definition 3.4(a) to verify u is a valid witness for x, and if so enters q_{accept}. Clearly, N enters q_{accept} on x if and only if a valid certificate

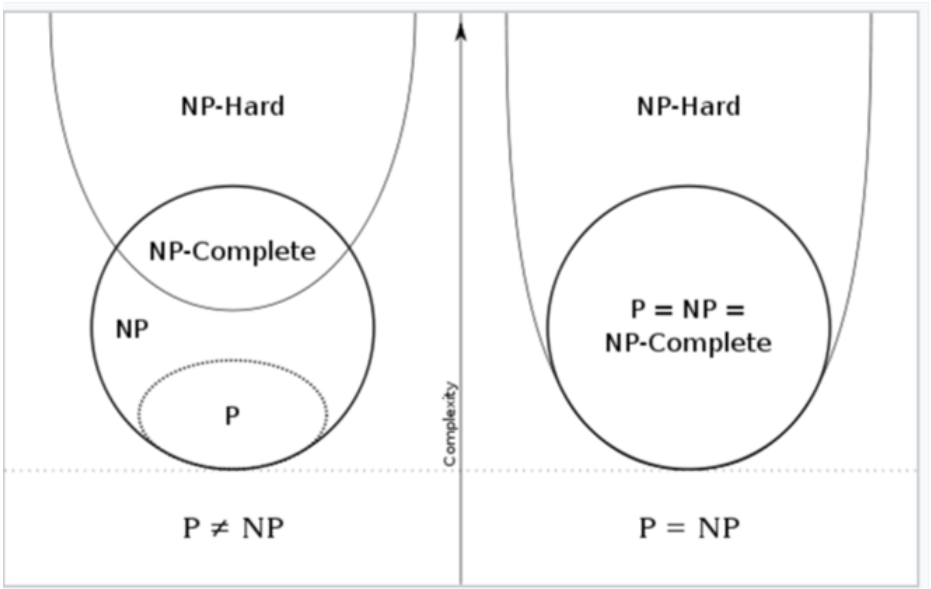
exists for x. Since $p(n) = O(n^c)$ for some $c > 1$ we must conclude that $L \in \boldsymbol{NTIME}(n^c)$

Definition 3.4(f): We say that some problem A is reducible to language B in polynomial time if there is some polynomial time function if every element $x \in \{0,1\}^*$, x is in A if and only if $f(x)$ is in B.

Definition 3.4(g): We say B is $\boldsymbol{NP}$-hard if A is reducible to B for every $A \in \boldsymbol{NP}$.

Definition 3.4(h): We say that B is $\boldsymbol{NP}$-complete if B is $\boldsymbol{NP}$-hard and $B \in \boldsymbol{NP}$

Figure 3.4(a):



[6]

Above we see figure 3.4(a). This is a visual representation of the above definitions for both $\boldsymbol{P} \neq \boldsymbol{NP}$, and $\boldsymbol{P} = \boldsymbol{NP}$. Note that $\boldsymbol{NP}$-hard problems

need not even be decidable at all. Hence we see that much of this class lives outside of $\boldsymbol{P}$ and $\boldsymbol{NP}$.

3.5. **Applications in cryptography.** We have, over the course of this writing, discussed a wide variety of seemingly disparate topics. In this section we begin to show how we can use lattices to encrypt and decrypt information in an efficient manner. Of course, we must first define what we mean by encryption as well as provide some background information.

If we define cryptography as the art of writing or solving codes then in modern cryptography we have two different types of encryption. First we see the symmetric type. This is best described as a system in which two parties communicate via one shared secret key. There are various algorithms but the key factor in symmetric encryption is that everyone must have the same key. In order for the message traffic to be secure the key must be transported in some secure fashion. Usually this means by courier or that all parties must meet and share the key prior to sending traffic. Another issue is that if one wishes to have private conversations with multiple individuals then one must generate a separate key for each individual. However, symmetric encryption enjoys a vast advantage in computational efficiency over asymmetric, or public key, cryptography. This is true for the following reasons.

- Symmetric keys generally require less CPU power to encrypt and decrypt messages. While each algorithm is different, symmetric encryption is generally faster than computationally heavy public key encryption. However, it has not been rigorously proven that all public key cryptography is more computationally demanding. This of course

is related to key length. For example a symmetric 128 bit AES encryption is at least as secure as a 3072 bit RSA encryption key. Symmetric key encryption is generally most vulnerable to brute force attacks because a randomly generated relatively short key creates a vast key space. This short key length greatly speeds up the encryption decryption process. Asymmetric encryption uses one way functions and thus require relatively longer keys to prevent attacks.

- Symmetric keys are by their nature private and shared amongst a select group. Since public keys are available to the masses the message traffic on a network using public key encryption could slow down network performance. A side benefit of symmetric key cryptography being private is that they are not generally vulnerable to plain text attacks used against public key cryptography.
- Symmetric keys encrypt messages and break up the plain text message into blocks fit the key length. For odd length messages something called padding, adding in random junk, is used to fill the unused space in the key. However the amount of padding is minimal. In public key cryptography, RSA specifically, a large amount of padding is necessary. This is due to the fact that the key in RSA, as previously stated, is based off a one way function. Careless generation of factors for the modulus are vulnerable to various methods of attacks. More importantly, without padding an attacker can use the public key to repeatedly encrypt messages. Through comparison of the encrypted messages the attacker can deduce the private key. The only defense is to pad the message with useless random junk. This

reduces available message space and is in contrast to the randomly generated symmetric key. As a result sending information via public key is much slower. In fact, most secure communication channels use a public key only for the transmission of a symmetric key, which is then used to send message traffic back and forth. This is referred to as hybrid encryption.

Despite having discussed all this the advantages of public key cryptography are obvious. The ability to establish a secure channel without couriers or meetings greatly enhances the availability of secure communication. However, even if one properly implements RSA, or some other public key system, the encryption could still be broken by a quantum computer. Integer factorization, discrete logarithms, and elliptic curves are no longer difficult problems when attacked by a quantum computer. The development of a such a computer is predicted within the next 30 years. While integer factorization, the basis of RSA, is a difficult problem it has not been proven to be in the $\boldsymbol{NP}$ class.

The usefulness of lattices is that many problems associated with them are proven to be $\boldsymbol{NP}$-hard. In fact, SVP is proven to be $\boldsymbol{NP}$-hard and the CVP is proven to be at least as hard as SVP. So given that these problems are in the $\boldsymbol{NP}$-hard class it is natural to ask the question if they can be used create public key encryption algorithms.

In 1996 the first semi-practical lattice based crypto-system was developed by Boldreich, Goldwasser, and Halevi. This system, GGH, was based on the fact that given the shortest bases for a lattice solving CVP is accomplishable in polynomial time. This is best demonstrated graphically.

Assume, there exists some lattice Λ of rank at most $n-1$. with GGH we must first generate a private key and then generate a public key. We do this as follows.

- This private key is the basis, B of Λ where B is "nearly orthogonal" and short. This is a "good" basis. If an orthogonal vector does exist one should not use this vector as algorithms to find such a vector exist and will complete in polynomial time. A good basis will allow us to solve instances of CVP where the vector $\overline{v}$ is very close to but not in Λ.
- The public key should be some "bad" basis H in Λ. By bad we mean long and not close to orthogonal. One possible use the Hermite Normal.

[16]

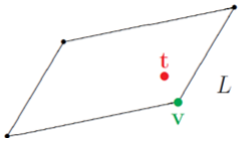
We must remember from earlier that assuming $B \cdot U = H$ for where $det(U) = 1$ then $\mathcal{L}(B) = \mathcal{L}(H)$. While CVP is easily solvable in $n = 2$, we will present two figures of a lattice in $\mathbb{Z}^2$ to demonstrate how a good basis can allow us to solve CVP. In figure 3.5(a) we see the good and the bad basis overlaid on the origin.

Figure 3.5(a):

[21]

and below this we see in figure 3.5(b) the target vector, or target point of the in the closest vector problem surrounded by the a translation of $\mathcal{P}(B)$ or the fundamental parallelepiped of the good basis.

Figure 3.5(b):



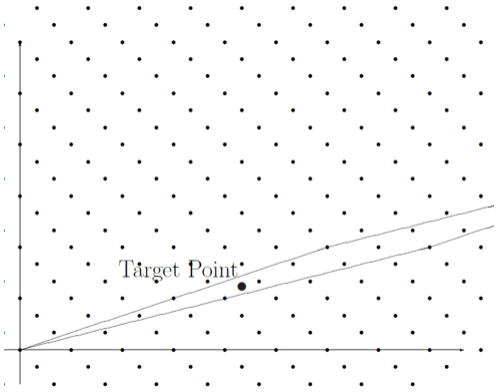
[21]

It is clear with a short and nearly orthogonal basis the solution to CVP is the vertex of the translated $\mathcal{P}(B)$ closest to the target vector. The designers of the private key, having selected such a basis, B, would be able to quick find a solution to the closest vector problem.

Finally we see in figure 3.5(c) that an individual trying to use the public key, basis H, would not be able to solve the closest vector problem in polynomial

time. The translated $\mathcal{P}(B)$ of the bad basis, H, would contain so many discrete lattice points that it would not be practical to select the solution to CVP in polynomial time.

Figure 3.5(c):



[21]

We now understand that GGH is cryptosystem which, with some pre-existing knowledge of the semi-orhogonal basis B, allows us to come to an approximate solution to CVP. This would allow someone to determine, with reasonable certainty what the solution to the closest vector problem was. We will now show how GGH can be used to encode and decode a message.

(1) We select some message $(m_1, ..., m_n)$ consisting of integer values which exists in the same space as the lattice Λ

(2) We then take $m = (m_1, ..., m_n)$ and encode via the bad basis, H.

$$\overline{v} = \sum m_i H_i$$

such that our message, since H is a basis for Λ, is now a point in Λ.

(3) In order to encrypt the message we now introduce a small error vector, $\overline{e}$, such that the ciphertext is.

$$c = \overline{v} + \overline{e} = m \cdot H + \overline{e}$$

Now that we have the cipher text c we can send the message to the recipient who must decrypt it.

(4) To decrypt we must use the private key, B, as well as its inverse B^{-1}. The decryption method is as follows.

$$c \cdot B^{-1} = (m \cdot H + e)B^{-1} = m \cdot U \cdot B \cdot B^{-1} + e \cdot B^{-1} = m \cdot U + e \cdot B^{-1}$$

(5) We are left, after right multiplication by the inverse of B with

$$m \cdot U + e \cdot B^{-1}$$

Note that $m \cdot U$ is in our lattice. To remove the term $e \cdot B^{-1}$ We apply an algorithm not discussed in this writing called Babai's algorithm. If we have a sufficiently short and orthogonal basis B there will remain.

$$m \cdot U$$

.

(6) Finally we multiple the above statement $m \cdot U$ by the inverse of U

$$m = m \cdot U \cdot U^{-1}$$

and we retrieve the original message.

While GGH was the first theoretical lattice based crypto-system which could function it was not practical from an efficiency standpoint. Since algorithms such LLL are capable of finding a nearly orthogonal basis for lattices up to $n = 100$ then size of the public key would be unmanageably large, say $n = 500$. If $n = 500$ then the size of our public key is now 500^2. In order to solve these problems a new system, N-th Degree Truncated Polynomial Ring, or NTRU.

3.6. **Cyclic lattices and NTRU.** NTRU is based off the premise that one can generate a cyclic lattice which has good properties for use in a public key encryption scheme. We must start by again defining the cyclic shift operator. We do this now in the context of a cyclic lattice. This is

Definition 3.6(a): The cyclic shift operator is defined as

$$rot(x_1, x_2, \cdots, x_{N-1}, X_N) = (x_N, x_1, X_2 \cdots, x_{N-1})$$

for every $(x_1, x_2, \cdots, x_{n-1}, X_n) \in \mathbb{R}^n$. The number of cyclic shifts applied, or iterations, is signified by k such that rot^k is k shifts.

Having defined the cyclic shift we can say that a sublattice Γ of $\mathbb{Z}^n$ is cyclic if $rot(\Gamma) = \Gamma$.

Cyclic lattices themselves are derived from ideals in the quotient polynomial ring $\mathbb{Z}[x]/(x^n - 1)$. If we have some polynomial $p(x) \in \mathbb{Z}[x]/(x^n - 1)$ then

$$p(x) = \sum_{n=0}^{n-1} a_n x^n$$

for some $a_0, \cdots, a_{n-1} \in \mathbb{Z}^n$.

Let us now define the isomorphism

$$\rho(p(x)) = (a_0, \cdots, a_{n-1}) \in \mathbb{Z}^n$$

then for any ideal $I \subseteq \mathbb{Z}[x]/(x^n - 1), \Gamma_I$ is equivalent to $\rho(I)$, and will be a sub-lattice of $\mathbb{Z}^n$. So we see that for every $p(x) = \sum_{n=0}^{n-1} a_n x^n \in I$

$$xp(x) = a_{n-1} + a_0 x + a_1 x^2 + \cdots + a_{n-2} x^{n-1} \in I$$

So that if we again apply our isomorphism ρ again we will see that

$$\rho(xp(x)) = (a_{n-1}, a_0, a_1, \cdots, a_{n-2}) = rot(\rho(p(x))) \in \Gamma_I$$

We also see that for integer vector $(a_0, \cdots, a_{n-1}) \in \Gamma_I$,

$$rot(a_0, \cdots, a_{n-1}) = \rho\left(x \sum_{n=0}^{n-1} a_n x^n \right) \in \Gamma_I$$

since $x \cdot p(x) \in I$. In summation, $\Gamma \subseteq \mathbb{Z}^n$ is a cyclic lattice if and only if $\Gamma = \Gamma_I$ for some some ideal $I \subseteq \mathbb{Z}[x]/(x^n - 1)$.

Additionally, we can say that an integer sub-lattice Γ is cyclic if $\Gamma(B) \subseteq \mathbb{Z}^n$ such that $B = \{p(x)\, mod\, f(x) \,:\, p(x) \in I\}$ where $f(x)$ is some monic polynomial of degree n and the ideal $I \subseteq Z[x]/(x^n - 1)$.

Remark: The above construction of cyclic lattices was taken from [8].

Having defined cyclic lattices we can now explain the basics of the NTRU encryption algorithm. We will approach this in a step by step fashion.

(1) Key Creation: Fix n, p, q with n prime and that p, q and $q > p$. Then choose two random polynomials $f, g \in \mathbb{R}$ such that both polynomials

have coefficients in $\{-1, 0, 1\}$. Also the degree of f, g is at most $n-1$. Then compute the inverses.

$$F_q \equiv f^{-1} \, (mod \, q) \text{ and } F_p \equiv f^{-1} \, (mod \, p)$$

$$\text{set } h = pF_q \cdot g \, (mod \, q)$$

So we have Public Key $= h$ and Private key $= f$ and (F_p)

(2) Encryption: The encryption process is similar to GGH and is based on creating a closest vector problem, but with a cyclic lattice. We take our message m and map it to a polynomial with $mod \, p$ coefficients. We then randomly generate a very small polynomial r with coefficient. Our ciphertext is generated by the following formula.

$$c = e \equiv p \cdot r \cdot h + m \, (mod \, q)$$

such that the ciphertext is an error vector created by adding the message to random shift from a convolution lattice.

(3) Decryption: To decrypt we utilize the private key f.

$$\begin{aligned} a &\equiv f \cdot e \, (mod \, q) \\ &\equiv f \cdot (r \cdot h + m) \, (mod \, q) \\ &\equiv f \cdot (r \cdot pF_q \cdot g + m) \, (mod \, q) \\ &\equiv pr \cdot g + f \cdot m \, (mod \, q) \end{aligned}$$

we must now recover the plaintext message m from a. Of note we must set the coefficients for a in the interval $[\frac{-q}{2}, \frac{q}{2}]$ to avoid potential problems with recovery of the message. To do this we set

$$b \equiv a \,(mod\, p) = f \cdot m \,(mod\, p)$$

lastly we use F_p to recover the original message.

$$F_p \cdot f \cdot m \,(mod\, p) = m$$

and the decryption process is complete.

To understand NTRU in terms of lattices we must first define the convolution modular lattice.

Definition 3.6(b): The convolution modular lattice L_h associated to the vector h and modulus q is the $2n$ dimensional lattice with basis given by the rows of the matrix:

$$L_h = RowSpan \left[\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} \\ 0 & 1 & \cdots & 0 & h_{n-1} & h_0 & \cdots & h_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right]$$

or alternatively we can say that

$$L_h = \{(a, b) \in \mathbb{Z}^{2n} \ : \ a * h \equiv b \, (mod \, q)\}$$

[21]

In the context of NTRU our lattice has the properties:

$$f(x) \cdot h(x) \equiv g(x) \, (mod \, q)$$

for f, g with small coefficients. This relation would imply the fact that L_h contains the vector

$$[f, g] = [f_0, f_1, \cdots, f_{n-1}, g_0, g_1, \cdots, g_{n-1}]$$

To show that $[f, g] \in L_h$ let

$$u\Big(x\Big) = \frac{-f(x) \cdot h(x) + g(x)}{q} \ \in \ \mathbb{Z}[x]$$

[21]

Then we have that:

$$[f_0, f_1, \cdots, f_{n-1}, u_0, u_1, \cdots, u_{n-1}] \left[\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} \\ 0 & 1 & \cdots & 0 & h_{n-1} & h_0 & \cdots & h_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right] = [f_0, f_1, \cdots, f_{n-1}, g_0, g_1, \cdots, g_{n-1}]$$

[21]

Let L_h be the 2 dimensional module over the over polynomial ring $R = \mathbb{Z}[x]/(x^n - 1)$. Then $L_h = \{[u, v] \in \mathbb{R}^2 \ : \ u \cdot \ h \equiv \ v \, (mod \, q)\}$. From this we can see that the lattice L_h contains the short vectors $[f, g]$ and the long vectors $[1, h]$, and $[0, q]$. So that we have two basis of L_h. The span of $[f, g]$ is the short basis which allows us to solve the CVP problem of NTRU. The basis formed by $[1, h]$ and $[0, q]$ is the bad long basis.

4. **Conclusion**

As computer processing speeds increase the mathematical community must continue to come up with cryptosystems based off hard problems. Of course if we want to be totally secure then we should never send a message in the first place. If that is not a viable option then we must encrypt our information somehow. While understanding of how these systems work is not easy the relevancy of encryption is plain to see in current events. Of course non of these encryption schemes work if the channel is too noisy. Over a noisy channel the cipher is junk to both the eavesdropper and the receiver. For this reason we must utilize linear codes so that an encrypted message sent will arrive without errors, ready for decryption. While not totally related the algebraic structures for cyclic lattices and cyclic codes are almost identical. These two complex yet fascinating subjects can be used in concert to deliver information and messages in a secure and accurate manner.

The last 60 pages are but a brief foray into the world of coding theory and cryptography. For a more in depth resource on lattice based cryptography please refer to An Introduction to the Theory of Lattices and Applications to

Cryptography by Dong Pyo Chi. For an in depth introduction to error correction codes written at a beginner level refer to [12].

References

[1] S. Arora and B. Arak Computational Complexity: A Modern Approach (draft) Princeton University, January 2007

[2] J.C. Bowman Math 422 Coding Theory and Cryptography University of Alberta, Edmonton, Canada, October 15, 2015

[3] M. Calderbank AN INTRODUCTION TO LINEAR AND CYCLIC CODES University of Chicago, VIGRE, 2008

[4] J. W. S. Cassels. *An Introduction to the Geometry of Numbers*. Springer-Verlag, 1959.

[5] D.R. Finston and P.J. Morandi An Introduction to Abstract Algebra via Applications Department of Mathematical Sciences New Mexico State University, Las Cruces NM 88003-8001, September 25, 2007

[6] B. Esfahbod Illustrations of P-Time, NP-Time, NP-Complete, and NP-Hard ven-diagrams http://behnam.es/

[7] L. Fukshansky. Discrete geometry lecture notes. *CMC Math 149, Fall 2013*. 2013.

[8] L. Fukshansky X. Sun On the Geometry of Cyclic Lattices Department of Mathematics, Claremont McKenna College School of Mathematical Sciences, Claremont Graduate University

[9] P. M. Gruber and C. G. Lekkerkerker. *Geometry of Numbers*. North-Holland Publishing Co., 1987.

[10] V. Guruswami Notes 1: Introduction, linear codes Introduction to Coding Theory, Carnegie Mellon University, 2010

[11] J.I. Hall Notes on Coding Theory chapter 6,7,8 Michigan State University, 2015

[12] R. Hill *A First Course in Coding Theory*. Oxford Applied Mathematics and Computing Science Series, Clarendon Press, 1990.

[13] J. Kelner 18.409 An Algorithmists Toolkit, Lecture 1 Massachusetts Institute of Technology, 2009

[14] B. Kleinberg Introduction to Algorithms CS 4820, Spring 2013 Notes on Turing Machines Cornell University

[15] R. Lidl and H. Niederreiter Introduction to Finite Fields and Their Applications, Chapter 2 Cambridge University Press, 1994.

[16] D. Micciancio and O. Regev Lattice-based Cryptography University of Southern California and Tel Aviv University. 22 July 2008

[17] D. Micciancio. CSE 206A, Lattice Algorithms and Applications Lecture 1. University of California San Diego. 2010

[18] J. Hoffstein and J. Pipher and J. H. Silverman NTRU: A Ring Based Public Key Cryptosystem In Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, J.P. Buhler (ed.), Lecture Notes in Computer Science 1423, Springer-Verlag, Berlin, 1998, 267-288.

[19] O. Regev Lattices in Computer Science, Lecture 1 University of Tel Aviv, 2004

[20] S. Schmittner Images of Closest Vector and Shortest Vector Problem https://research.schmittner.pw/

[21] J.H. Silverman An Introduction to theTheory of Lattices and Applications to Cryptography, online lecture files. University of Wyoming Summer School on Cryptography, 2006

[22] V. Vaikuntanathan 6.876 Advanced Topics in Cryptography: Lattices, lecture 3 Massachusetts Institute of Technology, 2015