12-2023

A Survey of Code-Based Cryptography

Antsa Rakotondrafara
arakoto@clemson.edu

A survey of code-based post-quantum cryptosystems

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science
Mathematical Science

---

by
Antsa Tantely Fandresena Rakotondrafara
December 2023

Accepted by:
Dr. Ryann Cartor, Committee Chair
Dr. Shuhong Gao
Dr. Jim Coykendall
Dr. Rafael D'Oliveira
Dr. Felice Manganiello

Abstract

One of the biggest concerns of the digital era is to keep data secure. With the rise of quantum computers, some of the problems that were considered hard will be solved in polynomial time. Therefore, efforts are taken to design practical cryptosystems that will resist attacks from both classical computers and quantum computers. In this thesis, we analyze three quantum-resistant code-based cryptosystems. We propose a new type of attack based on mixed-integer non-convex programming for the NIST submission FuLeeca.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Whitfield Diffie, one of the pioneers of public key cryptography once said, "without strong encryption, you will be spied on systematically by lots of people." Cryptography has been used to protect information for thousands of years. From the use of hieroglyphs by the Egyptians[1], to the Enigma machine of World War II [2], symmetric encryption was used in important communication, oftentimes of military value so that the messages could only be read by the intended receiver. In symmetric encryption, both the sender and the receiver need to agree on a shared secret. This shared secret, called *the secret key* is used in order to *encrypt* the message, that is to transform the message into a form that is not readable by an outsider. It is also used to *decrypt* the message, meaning to transform the message back to the original form. In the last few decades, the need for a secure way to send messages is no longer reserved for military use. Every day companies and individuals send and receive a lot of classified or sensitive information, for example, social security numbers can be used to apply for a loan in our names. We can also mention our credit card information and our health data. It is clear that encryption should be used on such pieces of information. Modern-day symmetric-key cryptosystems like AES [3] provide us with a way to encrypt and decrypt data with high speed, and AES has proven to be computationally impractical to break. One question then needs to be asked: how do we exchange the secret key in order to communicate securely?

Another concern for the digital era is the ownership and repudiation of a message. Let us suppose that a person named Alice communicates with another person named Bob. How could Bob be sure that the message he received is from Alice and not from an evil Eve who tries to

[1] See `http://all.net/edu/curr/ip/Chap2-1.html`

impersonate Alice? Also, how can Bob hold Alice accountable for the message that she sent to Bob? Lastly, how could Bob know if the message he receives does not contain errors? Nowadays, the term *cryptography* is used as an umbrella for the answers to all the previous questions. It refers to the use of mathematical algorithms in order to hide sensitive information (encryption schemes), verify identities (digital signature scheme), ensure private communication (key exchange), and prevent messages from being tampered with (cryptographic hash function).

In his paper [4], Diffie presented us the idea of public key cryptography. Public key cryptography was proposed as an answer to the problem of key distribution and identity verification. In a public key cryptosystem, Alice has a public key that she publishes for the whole world to use. Anyone can then encrypt a message to be sent to Alice but only Alice can decrypt the ciphertext using her private key. In order to construct such a cryptosystem, cryptographers use a one-way function with a trapdoor. That is a function such that the preimage is difficult to compute except for someone in possession of a secret piece of information (the trapdoor, a.k.a the secret key). The RSA encryption scheme [5] and the Diffie-Hellman key exchange [6] are two of the earliest public key cryptosystems that are still widely in use and most importantly, provide security to many of our internet transactions.

The RSA cryptosystem is based on the fact that the factorization of a product of two large primes is difficult unless one already has the factors. The private key is a pair of large primes p and q and the public key is $N = pq$. The Diffie-Hellman key exchange [6] is built upon the difficulty of solving the discrete logarithm. That is given a power of prime q, a primitive element $\alpha \in \mathbb{F}_q$. It is very easy to compute $y = \alpha^x \mod q$ for any $x \in \mathbb{F}_q$, and solving for x in $y = \alpha^x \mod q$ is computationally expensive. With the computational infrastructures that we have now, both the RSA cryptosystem and the Diffie-Hellman key exchange algorithm are secure. What is concerning is when quantum computers of large-scale will be available.

There is no known polynomial time algorithm to solve the factorization problem and the discrete logarithm problem with a classical computer. In 1994, while no useful quantum computers were built yet, Shor [7] proposed algorithms that use the property of quantum superposition and quantum Fourier transformation in order to factor a large number and get the discrete logarithm of a number efficiently using a quantum computer. As of now, the most powerful quantum computer is the Osprey chip by IBM [8], and has 433 qubits. Many other companies are also in the race to build larger quantum computers. This tells us that the large-scale quantum computers will arrive

and we need to be prepared.

The goal of *post-quantum cryptography* is to develop public key cryptosystems that can be used in our classical architectures and are secure against cryptanalysis performed using both classical computers and quantum computers. This mostly means, using hard problems in maths other than factorization and the discrete logarithm to create cryptosystems. We also would like to note that post-quantum cryptography is different from quantum cryptography. Quantum cryptography utilizes quantum mechanics in its encryption and decryption process so it requires a quantum facility to run. On the other hand, Post-Quantum Cryptography does not need any quantum facility.

To mitigate this threat posed by quantum computers, in 2016, the National Institute of Standards and Technology (NIST) started a project to standardize quantum-resistant public key cryptosystems (encryption, key encapsulations, and signature) [9]. The first formal call for submission was announced in 2016 and 82 submissions were received for this first round, one of them being the Classic McEliece cryptosystem [10]. This standardization process is not a competition in the sense that there is no 'winner' to be selected. A variation of algorithms will be selected to be standardized and some algorithms, even those not chosen to be standardized will still be considered as "good choice". All algorithms that are submitted are evaluated by both NIST and the public. The candidates for the NIST standardization can be classified by the mathematical structures they are based on. First, we have the cryptosystems based on lattices. Their security relies on the difficulty of either finding short integer solutions or a closest vector in a lattice. Second, we have multivariate cryptosystems which are based on the difficulty of solving a system of multivariate quadratic polynomial equations over a finite field. Third, we have hash-based schemes which are taking advantage of the properties of a cryptographic hash function. And lastly, we have the code-based cryptosystems on which we are focusing in this thesis. As of now, NIST has selected four algorithms for standardization[2]: the public key encryption CRYSTALS-KYBER (based on lattice) and the three signature schemes CRYSTALS-Dilithium (based on lattice), FALCON (based on lattice) and SPHINCS+ (hash-based signature scheme) and the cryptosystems BIKE [3], Classic McEliece [4], HQC [5] and SIKE [6] are on the fourth round of evaluation.

Among the finalists of this standardization process, the Classic McEliece cryptosystem de-

[2] `https://csrc.nist.gov/news/2022/pqc-candidates-to-be-standardized-and-round-4`
[3] `https://bikesuite.org/`
[4] `https://classic.mceliece.org/`
[5] `https://pqc-hqc.org/`
[6] `https://sike.org/`

serves a closer look. This NIST candidate is considered secure and is still under consideration to be standardized. The only drawback is that it has a large public key and therefore, may not be widely used in practice. The Classic McEliece cryptosystem is based on the original version of the McEliece cryptosystem which was proposed in [11] one year after the RSA cryptosystem was introduced. The McEliece cryptosystem, and hence the Classic McEliece NIST candidate, is based upon Goppa codes. Goppa codes are a type of linear code that is already well-studied. From the apparition of the McEliece cryptosystem to now, there are several papers that study both the linear code and the cryptosystem itself. A selected list can be viewed on the official website of Classic McEliece[7]. We will give the basic background of linear codes in Chapter 2 and an overview of the Classical McEliece in detail in Chapter 3.

The only standardized digital signatures are based on lattices and hash functions. In an effort to diversify the signature portfolio, NIST called for additional signature schemes for the post-quantum cryptosystem standardization [12]. This call for Additional Signatures received 4 submissions whose security is based on hard problems in coding theory·These cryptosystems were Enhanced pqsigRM, FuLeeca, LESS, MEDS, and Wave. We will focus on the FuLeeca cryptosystem and its cryptanalysis in Chapter 4.

Our main contribution is presenting a mixed integer programming key recovery attack against the NIST candidate FuLeeca.

[7] `https://classic.mceliece.org/papers.html`

Chapter 2

Background on Linear codes

2.1 Notation and basic concepts

Throughout this thesis, we use bold letters to denote vectors and matrices. Let q be a prime number and m, n, and k be positive integers such that $n \geq k \geq 1$. We denote by $\mathbb{F}_q$ the finite field of order q and $\mathbb{F}_{q^m}$ an extension field of $\mathbb{F}_q$ of degree m. We will use I_k to denote the identity matrix of size k. Let$\mathcal{M}_{k,n}(\mathbb{F}_{q^m})$ denote the space of all $k \times n$ matrices over $\mathbb{F}_{q^m}$, and $GL_n(\mathbb{F}_{q^m})$ denotes the set of all invertible matrices in $\mathcal{M}_{n,n}(\mathbb{F}_{q^m})$. For a matrix $\mathbf{M} \in \mathcal{M}_{k,n}(\mathbb{F}_{q^m})$, $\langle \mathbf{M} \rangle_{q^m}$ denotes the vector space spanned by the rows of $\mathbf{M}$ over $\mathbb{F}_{q^m}$. The element in the j-th row and $k-$th column of $\mathbf{M}$ is denoted by m_{jk}.

2.2 Linear codes

Definition 1 (Linear codes)**.** An $[n, k]$ *linear code* over $\mathbb{F}_{q^m}$ is a subspace of $\mathbb{F}_{q^m}^n$ of dimension k.

An $[n, k]$ linear code has *length* n and *rate* k/n. An element of a code is called a *codeword*. A $[n, k]$ linear code $\mathcal{C}$ can be defined by giving a matrix $\mathbf{G} \in \mathcal{M}_{k,n}(\mathbb{F}_{q^m})$ of full rank, called a *generator matrix* of $\mathcal{C}$, such that each codeword can be uniquely written as $\mathbf{y} = \mathbf{xG}$. Using the notation described above, $\mathcal{C} = \langle G \rangle_{\mathbb{F}_{q^m}}$. We can also define $\mathcal{C}$ by giving a matrix $\mathbf{H} \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and defining $\mathcal{C} = \{\mathbf{y} \in \mathbb{F}_{q^m}^n | \mathbf{y}\mathbf{H}^\top = \mathbf{0}\}$. The matrix $\mathbf{H}$ is called a *parity check matrix* of $\mathcal{C}$. For any vector $\mathbf{v} \in \mathbb{F}_{q^m}^n$, the syndrome of $\mathbf{v}$ is the vector $\mathbf{v}\mathbf{H}^\top$. Note that $\mathbf{v}\mathbf{H}^\top = \mathbf{0} \Leftrightarrow \mathbf{v} \in \mathcal{C}$.

In order to quantify the distance between vectors in $\mathbb{F}_{q^m}^n$, we endow $\mathbb{F}_{q^m}^n$ with a metric.

Below we will define different metrics that one can use. Each metric will induce a set of properties on codes and we will look at those details in later chapters.

Definition 2 (Hamming metric)**.** Let $n \in \mathbb{N}$. For $\mathbf{x} \in \mathbb{F}_{q^m}^n$, the *Hamming weight* of $\mathbf{x}$, denoted by $wt_H(\mathbf{x})$, is the number of its non-zero components. That is $wt_H(\mathbf{x}) = |\{i \in \{1, \ldots, n\} | x_i \neq 0\}|$.
For $\mathbf{x}, \mathbf{y} \in \mathbb{F}_{q^m}^n$, the *Hamming distance* of $\mathbf{x}$ and $\mathbf{y}$ is $d_H(\mathbf{x}, \mathbf{y}) = wt_H(\mathbf{x} - \mathbf{y})$. Note that we also have $d_H(\mathbf{x}, \mathbf{y}) = |\{i \in \{1, \ldots, n\} | \mathbf{x}_i \neq \mathbf{y}_i\}|$.

Example 1. Let $\mathbf{x} = (0, 1, 1, 0, 1)$, $\mathbf{y} = (1, 1, 0, 1, 1) \in \mathbb{F}_2^5$, then $d_H(\mathbf{x}, \mathbf{y}) = wt_H(\mathbf{x} - \mathbf{y}) = wt_H((1, 0, 1, 1, 0)) = 3$.

Definition 3 (Rank metric)**.** Fix a basis $\beta = \{\beta_1, \ldots, \beta_m\}$ of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Then for each element $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{F}_{q^m}^n$, we associate the matrix

$$\overline{\mathbf{x}} = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ x_{21} & x_{22} & \ldots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \ldots & x_{mn} \end{pmatrix} \in \mathcal{M}_{m,n}(\mathbb{F}_q),$$

where x_{ij} is the ith coordinate of x_j with respect to the basis β for $1 \leq i \leq m$, $1 \leq j \leq n$. The *rank weight* of $\mathbf{x}$ is $wt_R(\mathbf{x}) = Rank(\overline{\mathbf{x}})$. Note that the rank weight of $\mathbf{x}$ does not depend on the chosen basis.
For $\mathbf{x}, \mathbf{y} \in \mathbb{F}_{q^m}^n$, the *rank distance* of $\mathbf{x}$ and $\mathbf{y}$ is $d_R(x, y) = wt_R(x - y)$ for $x, y \in \mathbb{F}_{q^m}^n$.
A $[n, k]$ *rank linear code* over $\mathbb{F}_{q^m}$ is a $[n, k]$ linear code viewed as a rank metric space.

Example 2. For example, consider $q = 2, m = n = 6, k = 2$ and the finite field $\mathbb{F}_{2^m} = \mathbb{F}_{2^6} \equiv \mathbb{F}_2[x]/(x^6 + x^4 + x^3 + x + 1) \equiv \mathbb{F}_2[a]$ where a is a root of $x^6 + x^4 + x^3 + x + 1$.

Take the basis $\beta = \{1, a, a^2, a^3, a^4, a^5\}$ and consider the element

$$\mathbf{v} = \left(1,\, a^2 + 1,\, a^4 + 1,\, a^4 + a^3 + a,\, a^5 + a^4 + a^2 + a,\, a^5 + a^4\right).$$

The corresponding matrix is $\overline{\mathbf{x}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$ and $wt_R(\mathbf{x}) = rank(\overline{\mathbf{x}}) = 6$.

Definition 4 (Lee metric)**.** For the Lee metric-based codes, we require $m = 1$.
There are two equivalent definitions of the Lee weight of an element:

- Given an element a in $\mathbb{F}_p$ where the representation of a is chosen to be in $\{0, \ldots, q-1\}$, the Lee weight of a is $wt_L(a) := \min\{a, q-a\}$.
- Given an element a in $\mathbb{F}_p$ where the representation of a is chosen to be in $\{-\frac{q-1}{2}, \ldots, 0, \ldots, \frac{q-1}{2}\}$, the Lee weight of a is $|a|$.

The Lee Weight of a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$ is defined as $wt_L(\mathbf{x}) = \sum_{i=1}^n wt_L(x_i)$ and the the Lee distance of two vectors $\mathbf{x}$ and $\mathbf{y}$ is $d_L(\mathbf{x}, \mathbf{y}) = wt_L(\mathbf{x} - \mathbf{y})$.

Example 3. Let $p = 7$, then the Lee weight of $0, 1, 2, 3, 4, 5, 6$ will be respectively $0, 1, 2, 3, 3, 2, 1$. The Lee weight of the vector $(4, 5, 6, 3)$ is $wt_L((4, 5, 6, 3)) = 3 + 2 + 1 + 3 = 7$.

An $[n; k]$ Lee metric linear code $\mathcal{C}$ is a k-dimensional linear subspace of $\mathbb{F}_p^n$ endowed with the Lee metric. Note that for $q \in \{2, 3\}$, the Lee metric coincides with the Hamming metric. However, for other p, we can have a vector with a small Hamming weight and a large Lee weight or a vector with a large Hamming weight and a small Lee weight.

Definition 5 (Minimum Distance)**.** Let $\mathcal{C} \subseteq \mathbb{F}_{q^m}^n$ be a linear code endowed with a distance d and its associated weight function wt. The *minimum distance* of $\mathcal{C}$ is denoted by

$$d(\mathcal{C}) = \min\{d(\mathbf{x}, \mathbf{y}) | \mathbf{x}, \mathbf{y} \in \mathcal{C}, \mathbf{x} \neq \mathbf{y}\}$$

. Note that $\mathcal{C}$ has finitely many elements, so $d(\mathcal{C})$ is well-defined. Also, for any $\mathbf{x}, \mathbf{y} \in \mathcal{C}$, $\mathbf{x} - \mathbf{y} \in \mathcal{C}$, so $d(\mathcal{C})$ can be equivalently defined as $d(\mathcal{C}) = \min\{wt(\mathbf{v}) | \mathbf{v} \in \mathcal{C}, \mathbf{v} \neq \mathbf{0}\}$.

2.3 Hard problems in Coding Theory

Coding theory is the branch of mathematics that is focused on designing error-correcting codes that can be used efficiently to send reliable information across noisy channels. From the start of coding theory, some types of codes have been found to be easy to decode. Those will be the codes that we use in cryptography. However, for a general code, the decoding process is hard, so we transform our basis so that our public key resembles a random code. We present here a selected list of hard problems in coding theory and we will later give some cryptosystems based on their difficulties. We use the term *attack* to refer both to the act of solving an instance of a given hard problem and any particular algorithm designed for that purpose.

A linear code is a subspace of a finite-dimensional vector space over a finite field endowed with a distance function. This subspace is created using a "nice basis" or a "nice parity check matrix", a term that we will rigorously define for each cryptosystem that we will analyze, which is kept as the secret key. A "scrambled basis" or a "scrambled parity check matrix" is used as the public key and resembles a random matrix. Depending on the cryptosystem, the transformation from the "nice" to the "scrambled" may be kept secret [1] or published [13]. In a typical code-based encryption scheme, a ciphertext is an element of the code (a.k.a. a codeword) plus a random error vector of known weight (distance from the origin). The ciphertext can be easily computed by anyone using the scrambled basis. Suppose $\mathcal{C} \in \mathbb{F}_{q^m}^n$ is a $[n, k]$-linear code over $\mathbb{F}_{q^m}$ that is defined by a nice basis $\mathbf{G}$ and a scrambled basis $\mathbf{G}'$. For a message $\mathbf{x}$, to get the corresponding ciphertext, we *encode* $\mathbf{x}$. That is, we compute $\mathbf{y} = \mathbf{x}\mathbf{G}' + \mathbf{e}$ where $\mathbf{e}$ is randomly generated with given weight. The "nice basis" allows the extraction of the codeword, a process named *decoding* from the ciphertext very easily. A strong cryptosystem is designed so that using the public key and the ciphertext only, it is hard for an attacker to get the codeword. Such a cryptosystem relies on the difficulty of

1. Inverting the transformation, and hence, compute $\mathbf{G}$ or an equivalent matrix $\mathbf{G}_1$, from $\mathbf{G}'$. Here the term "equivalent" means that $\mathbf{G}_1$ may not be equal to $\mathbf{G}$ but it is also a nice basis of $\mathcal{C}$ that can be used interchangeably with $\mathbf{G}$.

2. Decoding $\mathbf{y}$ using $\mathbf{G}'$.

For a signature scheme, the nice basis is used to generate a signature for a message. The signature is a codeword with the following properties. It is easy to verify that the signature is a

codeword using the public key. The signature must satisfy a set of conditions, different for each cryptosystem, that is hard to achieve if an attacker starts with the public key. Such a cryptosystem relies on the difficulty of

1. Inverting the transformation.
2. Producing a codeword with the desired properties (usually a restriction on weight with other cryptosystem-specific characteristics).

2.3.1 The bounded decoding problem in Hamming metric

Problem 1 ((q, m, n, k, r)Bounded Decoding Problem in Hamming metric)**.**
Input: $G \in \mathcal{M}_{k,n}(\mathbb{F}_{q^m})$ of rank k, $\mathbf{y} \in \mathbb{F}_{q^m}^n$ and $r \in \mathbb{N}$ such that $wt_H(\mathbf{y} - \mathbf{x}G) \leq r$ for some $\mathbf{x} \in \mathbb{F}_{q^m}^k$.
Output: $\mathbf{x} \in \mathbb{F}_{q^m}^k$ and $\mathbf{e} \in \mathbb{F}_{q^m}^n$ such that $wt_H(\mathbf{e}) \leq r$ and $\mathbf{y} = \mathbf{x}G + \mathbf{e}$.

This problem is already extensively studied in the literature and is proven to be NP-complete. Up to now, the most efficient algorithms to solve a general instance of Problem 1 is based on a method called *Information Set Decoding*(ISD). Starting with an instance of Problem 1 which is promised to have a unique solution, the idea of ISD is to randomly select k positions of $\mathbf{y}$. Let us call the set of selected positions I. If the corresponding submatrix in $\mathbf{G}$ is full rank, then solve $\mathbf{y}_I = \mathbf{a}\mathbf{G}_I$. If the weight of $\mathbf{y} - \mathbf{a}\mathbf{G}$ is $\leq r$ then I correspond to the positions of $\mathbf{y}$ where there was no error then $\mathbf{a} = \mathbf{x}$. In case I do not give a solution that satisfies $\mathbf{y} - \mathbf{a}\mathbf{G} \leq r$ repeat the process until we get the desired result and $\mathbf{x}$ is found. Even with the help of quantum computers, the complexity of ISD is exponential, which works in favor of the cryptographic systems based on Problem 1.

For a family of code called Goppa code, an algorithm for fast decoding has been found. The Classic McEliece cryptosystem that we will study in Chapter 3 is taking advantage of that property.

2.3.2 The bounded Rank Decoding Problem

Problem 2 ((q, m, n, k, r)Bounded Rank Decoding Problem (RD))**.**
Input: $G \in \mathcal{M}_{k,n}(\mathbb{F}_{q^m})$ of rank k, $\mathbf{y} \in \mathbb{F}_{q^m}^n$ and $r \in \mathbb{N}$ such that $wt_R(\mathbf{y} - \mathbf{x}G) \leq r$ for some $\mathbf{x} \in \mathbb{F}_{q^m}^k$.
Output: $\mathbf{x} \in \mathbb{F}_{q^m}^k$ and $\mathbf{e} \in \mathbb{F}_{q^m}^n$ such that $wt_R(\mathbf{e}) \leq r$ and $\mathbf{y} = \mathbf{x}G + \mathbf{e}$.

This problem is not known to be hard, however, it can be reduced to Problem 1. There are two main categories of attacks on the RD problem: combinatorial attacks and algebraic attacks. In a typical combinatorial attack, the adversary makes an educated guess on the support of the error

e, that is which components of **e** will be non-zero. Then, they transform the RD problem into a system of linear equations on the components of **e** with respect to that guess. We show in Table 4.3.2 the complexity of the best-known combinatorial attacks.

Attack	Complexity
[14]	$\mathcal{O}\left(\min\left\{m^3t^3q^{(t-1)(k+1)}, (k+t)^3t^3q^{(t-1)(m-t)}\right\}\right)$
[15]	$\mathcal{O}\left((n-k)^3m^3q^{\min\left\{t\left\lceil\frac{mk}{n}\right\rceil,(t-1)\left\lceil\frac{m(k+1)}{n}\right\rceil\right\}}\right)$
[16]	$\mathcal{O}\left((n-k)^3m^3q^{t\left\lceil\frac{m(k+1)}{n}\right\rceil-m}\right)$

Table 2.1: Best known combinatorial attacks on the RD problem. [1]

For the algebraic attack, the RD problem is converted into a system of polynomials, and this system is solved using algebraic techniques. In the case where an educated guess is used to speed up the computation, the process is called a hybrid attack. In Table 4.3.2, we summarize the algebraic attacks that were considered in [1].

Attack	Condition and method used	Complexity
[15]	$\left[\frac{(t+1)(k+1)-(n+1)}{t}\right] \leqslant k$ Annulator polynomial	$\mathcal{O}\left(k^3t^3q^{t\left[\frac{(t+1)(k+1)-(n+1)}{t}\right]}\right)$
[17]	$m\binom{n-k-1}{t} \geqslant \binom{n}{t} - 1$	$\mathcal{O}\left(m\binom{n-p-k-1}{t}\binom{n-p}{t}^{\omega-1}\right)$, where $\omega = 2.81$ and $p = \min\left\{1 \leqslant i \leqslant n : m\binom{n-i-k-1}{t} \geqslant \binom{n-i}{t} - 1\right\}$
[18]	Maximal minors	$\mathcal{O}\left(\left(\frac{((m+n)t)^\top}{t!}\right)^\omega\right)$
[17]	$m\binom{n-k-1}{t} < \binom{n}{t} - 1$	$\mathcal{O}\left(q^{at}m\binom{n-k-1}{t}\binom{n-a}{t}^{\omega-1}\right)$ where $a = \min\left\{1 \leqslant i \leqslant n : m\binom{n-k-1}{t} \geqslant \binom{n-i}{t} - 1\right\}$
[18]	Maximal minors	$\mathcal{O}\left(\left(\frac{((m+n)t)^{t+1}}{(t+1)!}\right)^\omega\right)$

Table 2.2: Best known algebraic attacks on the RD problem. [1]

For example, take $\mathbf{G} = \begin{pmatrix} 1 & a+1 & a^2+1 & a^3+1 & a^4+1 & a^5+1 \\ 1 & a^2+1 & a^4+1 & a^4+a^3+a & a^5+a^4+a^2+a & a^5+a^4 \end{pmatrix} \in \mathcal{M}_{2,6}(\mathbf{F}_{2^6})$. We see here that $\mathbf{G}$ is full rank so its rows generate a $[6,2]$ rank linear code $\mathcal{C}$ over $\mathbb{F}_{2^6}$.

Encoding the vector $\mathbf{x} = (1+a, a^2+a)$ gives the element $\mathbf{xG} = (a^2+1, a^4+a^3+a+1, a^5+a^4+a, a^4+a^3+a^2, a^5+a^4+1, a^5+a^3+a^2+a) \in \mathcal{C}$. The vector $\mathbf{e} = (1, a+1, 0, 0, 0, 0)$, of rank weight 2 is not in $\mathcal{C}$. We can prove that by stacking $\mathbf{e}$ on top of $\mathbf{G}$ and taking the row echelon form.

$$\begin{pmatrix} \mathbf{e} \\ \mathbf{G} \end{pmatrix} \xrightarrow{REF} \begin{pmatrix} 1 & 0 & 0 & a^5+a^4+a^2+a & a^4+a^3+a^2 & a^5+a^4+a^3+a^2 \\ 0 & 1 & 0 & a^4+a & a^5+a^4+a^3+a^2+1 & a^4+a^2 \\ 0 & 0 & 1 & a^5+a^4+a^2 & a^2+1 & a^5+a^4+a^3+a^2 \end{pmatrix}.$$

Thus the vector $\mathbf{y} = \mathbf{xG}+\mathbf{e} = \left(a^2,\, a^4+a^3,\, a^5+a^4+a,\, a^4+a^3+a^2,\, a^5+a^4+1,\, a^5+a^3+a^2+a\right)$ is not in $\mathcal{C}$. We provide a code for solving this instance of RD problem using the annulator polynomial in Macaulay2 in Appendix **??** Code A.

The families Gabidulin Codes and Low-Rank Parity-Check Codes(LRPC) have fast decoding algorithms. Although many cryptosystems based on Gabidulin Codes have been completely broken, [1] proposed a cryptosystem based on Gabidulin Codes that stands against the attacks used on its peers. We will investigate this cryptosystem in Chapter 4.

2.3.3 Finding Codeword of Given Lee Weight

The problem of finding a codeword of Given Lee weight can be used to create signature schemes and is described as follows:

Problem 3 (Finding Codeword of Given Lee Weight)**.** Given $H \in \mathbb{F}_p^{(n-k)\times n}$ which is a parity check matrix for a $[n,k]$ linear code $\mathcal{C}$ in $\mathbb{F}_q$, and $w \in \mathbb{N}$, find a codeword $\mathbf{c} \in \mathbb{F}_p^n$ such that $\mathbf{cH}^\top = \mathbf{0}$ and $wt_L(\mathbf{c}) = w$.

The decision version of Problem 3 is proven to be NP-complete. All of the algorithms that are proposed to solve this problem belong to the family of ISD and therefore are of exponential complexity. The NIST candidate FuLeeca, which we will study in Chapter 5 marks the first cryptosystem based on the Lee metric.

Chapter 3

A cryptosystem based on Hamming linear codes: McEliece Cryptosystem,

3.1 Overview of the McEliece Cryptosystem

Code-based cryptography was born with the paper [11]. This was the same period where the RSA [5] and Diffie-Hellman [6] cryptosystems were introduced. RSA and Diffie-Hellman knew large practical use and the McEliece cryptosystem did not because of its large public key size. 45 years later, the practical use of RSA and Diffie-Hellman was challenged by the rise of quantum computers which broke their mathematical security foundation. All eyes are now on the original McEliece cryptosystem, which is believed to be quantum-resistant. Not only the original McEliece cryptosystem is one of the foundations of the NIST submission Classic McEliece but McEliece's idea of hiding the structure of an easy-to-decode linear code was borrowed from by many researchers to build many of our current code-based cryptosystems.

3.2 Background

3.2.1 History and development of the McEliece Cryptosystem

In [11], McEliece presents a secure cryptosystem with an extremely rapid data rate that is suitable for multi-user communication networks. He built his work on the existence of Goppa codes which will be explained in Section 3.2.2. It is the first cryptosystem that uses random error in the encryption process. This attractiveness of McEliece leads to two different directions in the cryptographic research.

The first direction is to use different types of linear codes that will lead to smaller public key sizes. A lot of McEliece variants that use Hamming weight but replace the Goppa codes were broken as the structure of their public key matrices could be exploited to solve for the private key. Therefore, the original version of McEliece is still considered the most secure version. Another way to use different types of codes is to consider different metrics. This leads to the study of rank-linear codes and Lee metric linear codes.

The second direction is the improvement of the cryptanalysis of McEliece. The most efficient algorithms to attack McEliece, called Information Set Decoding, are built from the work of [19]. Although those algorithms have been extensively studied, their complexity in time and in memory is exponential with respect to the number of errors in the codeword. The recent paper [20] proposes the fastest quantum ISD algorithm that still requires an exponential quantum memory. Therefore, even if the large-scale quantum computers are practical, the McEliece cryptosystem will still be secure. We will only need to update the parameters.

3.2.2 Theoretical foundation of the system: Goppa codes

Definition 6 (Goppa codes)**.** Let $g(z) = g_0 + g_1 z + \cdots + g_t z^t \in \mathbb{F}_{q^m}[z]$, and $L = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{F}_{q^m}$ such that $g(\alpha_i) \neq 0$, for all $\alpha_i \in L$. Then the *Goppa code* with parameters $g(z)$ and L is defined by

$$\Gamma(L, g(z)) = \left\{ \mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{F}_q^n : \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i} \equiv 0 \mod g(z) \right\}.$$

The Patterson Algorithm [21] we describe below decodes Goppa codes in polynomial time.

Algorithm 1: Patterson's Algorithm for Error Correction

Input: The received vector $\mathbf{y}$ and the Goppa code $\Gamma(L, g)$.
Output: The error vector $\mathbf{e}$.

1 Compute syndrome $S(\mathbf{y})$ an element of $\frac{\mathbb{F}_{q^m}[z]}{\langle g(z)\rangle}$
2 Compute $T(z) = S(\mathbf{y})^{-1} \bmod g(z)$
3 Compute $P(z) = \sqrt{T(z) + z} \bmod g(z)$
4 Compute $u(z)$ and $v(z)$ with $u(z) = v(z)S(\mathbf{y}) \bmod g(z)$
5 Compute the locater polynomial $\sigma(z) = u(z)^2 + zv(z)^2$
6 Find the roots of $\sigma(z)$
7 Find error positions, i.e., error vector $\mathbf{e}$

3.3 The scheme

The McEliece cryptosystem is a block-cipher, meaning the data to be encrypted needs to be divided into k-bit blocks. The key generation, encryption, and decryption processes for the McEliece cryptosystems are as follows.

Algorithm 2: The Key Generation Algorithm for McEliece

Input: Parameters integers m, k, n, t where $n = 2^m$ is the length of the code, $k \geq n - tm$ is the dimension of the code and t is the minimum distance of the code.
Output: $\{pk, sk\}$ a public key-secret key pair

1 Randomly select an irreducible polynomial of degree t over $\mathbb{F}(2^m)$
2 Produce a $k \times n$ generator matrix $\mathbf{G}$ for the Goppa code corresponding to t.
3 Randomly select a $k \times k$ invertible matrix $\mathbf{S}$
4 Randomly select a $n \times n$ permutation matrix $\mathbf{P}$
5 Computes $\mathbf{G}' = \mathbf{SGP}$
6 Return $sk = \{\mathbf{S}, \mathbf{G}, \mathbf{P}\}$ and $pk = \mathbf{G}'$

Algorithm 3: The Encryption Algorithm for McEliece

Input: A block of message $\mathbf{x} \in \mathbb{F}_2^k$, public key $\mathbf{G}'$
Output: $E_{\mathbf{G}'}(\mathbf{x})$ ciphertext of $\mathbf{x}$ encrypted using $\mathbf{G}'$

1 Randomly generate a vector $\mathbf{e}$ of length n and weight t.
2 Return $\mathbf{y} = \mathbf{xG}' + \mathbf{e}$

Algorithm 4: The Decryption Algorithm

Input: $\mathbf{y} = E_{\mathbf{G}'}(\mathbf{x})$ ciphertext of x encrypted using $\mathbf{G}'$, corresponding secret key $\mathbf{G}$
Output: The original message $\mathbf{x}$

1 Compute $\mathbf{y}' = \mathbf{yP}^{-1}$
2 Decode $\mathbf{y}'$ using Patterson's algorithm and let $\mathbf{x}'$ be the coefficient vector.
3 Return $\mathbf{x} = \mathbf{x}'\mathbf{S}^{-1}$

3.4 Strengths of the McEliece Cryptosystem

The first advantage of the McEliece Cryptosystem over the RSA algorithm was presented by McEliece himself in the original paper. The encryption and decryption of the McEliece cryptosystem are significantly faster than those of the RSA cryptosystem. Now that extensive cryptanalysis of McEliece has been done over decades, the McEliece cryptosystem is still believed to be quantum-resistant. Therefore, we still can expect to see the McEliece cryptosystem being in consideration long after the RSA and Diffie Hellman cryptosystems are retired. A lot of effort is also put into the improvement of the implementation of the McEliece scheme and it will only get better.

3.5 Weaknesses and Criticisms

Since its early days, the McEliece cryptosystem has been criticized for its large public key size. First, storing a whole matrix in itself requires a lot of memory. Also, for the scheme to be secure, we need to use codes of low rate [22]. That is the value of k needs to be significantly smaller than n.

The original version proposed by McEliece is not secure against the chosen ciphertext attack. That is, given a ciphertext $\mathbf{y}$, the adversary can request the decryption of any different ciphertext from a random oracle except the ciphertext of interest. Then, based on the plaintext he received, he can do some computation to retrieve the plaintext corresponding to the ciphertext of interest. Suppose $\mathbf{x}$ is encrypted as $\mathbf{y} = \mathbf{xG} + \mathbf{e}$. We can randomly choose one bit from the set of bits with entry 1 and one bit from the set of bits with entry 0. By inverting those bits, we obtain with probability $\mathbf{t(n-t)/n(n-1)}$ a new valid different ciphertext for $\mathbf{x}$ with exactly t errors. We can then send this new ciphertext to a random decryption oracle and get the message back! However, if the inverting does not give us a valid ciphertext, we will receive a decryption failure error.

3.6 McEliece today

The NIST candidate Classic McEliece is built upon on original McEliece cryptosystem and mitigates this last criticism of McEliece. That is if an adversary attempts a chosen ciphertext attack on Classic McEliece, instead of getting back a decryption failure error, Classic McEliece is designed to avoid leaking side channel errors and to return back a "fake plaintext". McEliece is used in the

iPhone and iPad app PQChat[1].

[1] https://post-quantum.com/messaging/index.html

Chapter 4

A Cryptosystem based on Rank linear codes

There have been many cryptosystems built on the rank metric, including Rollo[23] and RQC[24]. In this chapter, we will provide more detail on the scheme introduced in [1].

4.1 Background

4.1.1 Gabidulin codes

Definition 7 (Moore matrices)**.** For positive integers $k \leq n \leq m$, let us consider a vector $\mathbf{g} = (\mathbf{g}_1, \ldots, \mathbf{g}_n) \in \mathbb{F}_{q^m}^n$. The Moore matrix of k rows generated by $\mathbf{g}$ is the matrix $\mathbf{G} \in \mathcal{M}_{k,n}(\mathbb{F}_{q^m})$ defined by

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_{n-1} & \mathbf{g}_n \\ \mathbf{g}_1^q & \mathbf{g}_2^q & \cdots & \mathbf{g}_{n-1}^q & \mathbf{g}_n^q \\ \mathbf{g}_1^{q^2} & \mathbf{g}_2^{q^2} & \cdots & \mathbf{g}_{n-1}^{q^2} & \mathbf{g}_n^{q^2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{g}_1^{q^{k-1}} & \mathbf{g}_2^{q^{k-1}} & \cdots & \mathbf{g}_{n-1}^{q^{k-1}} & \mathbf{g}_n^{q^{k-1}} \end{pmatrix} = \left(\mathbf{g}_j^{q^{i-1}}\right)_{i \in [1..k], j \in [1..n]}.$$

Some properties of Moore matrices are described in the following proposition.

Proposition 1. 1. Let $A, B \in \mathcal{M}_{k,n}(\mathbb{F}_{q^m})$ be the two Moore matrices generated respectively

by $\mathbf{g}_1$ and $\mathbf{g}_2$. Then $A + B$ is the Moore matrix generated by $\mathbf{g}_1 + \mathbf{g}_2$. In fact, for each $i \in [1..k], j \in [1..n]$, $\left(\mathbf{g_1}_j + \mathbf{g_2}_j\right)^{q^{i-1}} = \left(\mathbf{g_1}_j\right)^{q^{i-1}} + \left(\mathbf{g_2}_j\right)^{q^{i-1}}$ as the intermediate terms in the binomial form will be multiple of q.

2. For a vector $\mathbf{g} \in \mathbb{F}_{q^m}^n$ and a matrix $\mathbf{Q} \in \mathcal{M}_{n,l}(\mathbb{F}_q)$, let $\mathbf{G} \in \mathcal{M}_{n,l}(\mathbb{F}_{q^m})$ be the Moore matrix of k rows generated by $\mathbf{g}$. Then $\mathbf{GQ}$ is the $k \times l$ Moore matrix generated by $\mathbf{gQ}$. This is because the Frobenius automorphism $u \mapsto u^q$ and its powers fix any element in $\mathbb{F}_q$.

3. For a vector $\mathbf{g} \in \mathbb{F}_{q^m}^n$ with $wt_R(\mathbf{g}) = n$, the Moore matrix $\mathbf{G}$ of k rows generated by $\mathbf{g}$ satisfies $rank(\mathbf{G}) = k$. In fact, if $wt_R(\mathbf{g}) = n$, then the set of the components of $\mathbf{g}$ is linearly independent over $\mathbf{q}$. We then obtain $\{\mathbf{g}_1, \ldots, \mathbf{g}_k\}$ is linearly independent. If $rank(\mathbf{G}) < k$ then there is any $\lambda \in \mathbb{F}_{q^m}^k$ such that $\lambda^\top \mathbf{G} = \mathbf{0}_{\mathbf{F}_{q^m}^n}$. In particular we should have t $\lambda^\top (\mathbf{g}_1, \ldots, \mathbf{g}_k)^\top = 0$ as $(\mathbf{g}_1, \ldots, \mathbf{g}_k)^\top$ is the first column of $\mathbf{G}$. Since that contradicts the fact that $\{\mathbf{g}_1, \ldots, \mathbf{g}_k\}$ is linearly independent, we conclude that $\mathbf{G}$ is full rank.

Definition 8 (Gabidulin codes)**.** For positive integers $k \leq n \leq m$ and $\mathbf{g} \in \mathbf{F}_{q^m}^n$ with $wt_R(\mathbf{g}) = n$, let $\mathbf{G}$ be the Moore matrix of k rows generated by $\mathbf{g}$. Then, the rowspace $\langle \mathbf{G} \rangle_{q^m}$ of $\mathbf{G}$ is called the $[n, k]$ *Gabidulin code* generated by $\mathbf{g}$.

Gabidulin codes were introduced in [25] where it was proved that a $[n, k]$ Gabidulin code has a minimum rank distance of $d = n - k + 1$. This makes them the first codes that can correct up to $\lfloor \frac{n-k}{2} \rfloor$ to be known such that the construction is possible for any $k \leq n \leq m$. Another reason that makes them valuable is that they have efficient decoding algorithms [25], [26], [27]. This allows us to use Gabidulin codes in a McEliece-like cryptosystem.

As we saw previously, a Gabidulin code can be determined using the generator $\mathbf{g}$ and the parameters q, m, n, k. That is, instead of storing or sending the matrix $\mathbf{G}$, we only need to store or send the vector $\mathbf{g}$. Another technique to cut down the memory used to store the code is by generating it using partial circulant matrices which will be described below.

4.1.2 Partial cyclic codes

Definition 9 (Partial circulant matrices)**.** Let $\mathbf{g} = \{\mathbf{g}_1, \ldots, \mathbf{g}_n\} \in \mathbf{F}_{q^n}^n$ and $k \leq n$ and integer. The $k \times n$ partial circulant matrix generated by $\mathbf{g}$ is obtained by cyclically right shifting its $i - 1$-th row for $2 \leq i \leq n$, that is

$$\mathrm{PC}_k(\mathbf{g}) = \begin{pmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_{n-1} & \mathbf{g}_n \\ \mathbf{g}_n & \mathbf{g}_1 & \cdots & \mathbf{g}_{n-2} & \mathbf{g}_{n-1} \\ \mathbf{g}_{n-1} & \mathbf{g}_n & \cdots & \mathbf{g}_{n-3} & \mathbf{g}_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{g}_{n-k+2} & \mathbf{g}_{n-k+3} & \cdots & \mathbf{g}_{n-k} & \mathbf{g}_{n-k+1} \end{pmatrix}.$$

A square partial circulant matrix is called *circulant matrix*. For a given field $\mathbb{F}$, which may be any of the fields we are working with, we will denote by $\mathrm{PC}_n(\mathbb{F})$ the set of all $n \times n$ circulant matrices over $\mathbb{F}$. The paper by Chalkley [28] gives a more in-depth study of partial circulant matrices. In particular, he proved that $\mathrm{PC}_n(\mathbb{F}_q)$ forms a commutative ring under usual matrix addition and multiplication. Guo and Fu [1] use invertible circulant matrices in $\mathcal{M}_{n,n}(\mathbb{F}_q)$ in the key generation. It is then important to give a sufficient and necessary condition for a circulant matrix to be invertible and the number of invertible circulant matrices over $\mathbb{F}_q$.

Proposition 2. [29] For a vector $\boldsymbol{m} = (m_0, \cdots, m_{n-1}) \in \mathbb{F}_q^n$, we define $\boldsymbol{m}(x) = \sum_{i=0}^{n-1} m_i x^i \in \mathbb{F}_q[x]$. A sufficient and necessary condition for $\mathrm{PC}_n(\boldsymbol{m})$ being invertible is $\gcd(\boldsymbol{m}(x), x^n - 1) = 1$.

Proposition 3. [30] For a polynomial $f(x) \in \mathbb{F}_q[x]$ of degree n, let $g_1(x), \cdots, g_r(x) \in \mathbb{F}_q[x]$ be r distinct irreducible factors of $f(x)$, i.e. $f(x) = \prod_{i=1}^{r} g_i(x)^{e_i}$ for some positive integers $e_1, \cdots, e_r$. Let $d_i = \deg(g_i(x))$ for $1 \leqslant i \leqslant r$, then

$$\Phi_q(f(x)) = q^n \prod_{i=1}^{r} \left(1 - \frac{1}{q^{d_i}}\right),$$

where $\Phi_q(f(x))$ denotes the number of polynomials relatively prime to $f(x)$ of degree less than n.

Remark 1. To generate the Gabidulin code in their cryptosystem, Guo and Fu [1] started with a normal element $g \in \mathbb{F}_{q^n}$ over $g \in \mathbb{F}_q$. The vector $\mathbf{g} = (g^{q^{n-1}}, g^{q^{n-2}}, \ldots, g) \in \mathbb{F}_{q^n}^n$ is then a $\mathbb{F}_q$ basis vector of $\mathbb{F}_{q^n}$. Note that the $n \times k$ partial circulant matrix generated by $\mathbf{g}$ is

$$\mathbf{G} = \begin{pmatrix} g^{q^{n-1}} & g^{q^{n-2}} & \cdots & g \\ g & g^{q^{n-1}} & \cdots & g^{q^1} \\ \vdots & \vdots & \ddots & \vdots \\ g^{q^{k-2}} & g^{q^{k-3}} & \cdots & g^{q^{k-1}} \end{pmatrix},$$

which coincides with the Moore matrix generated by $\mathbf{g}$. Therefore the $[n,k]$ linear code $\mathcal{G} = \langle \mathbf{G} \rangle_{q^n}$ is called *a partial cyclic Gabidulin code* generated by $\mathbf{g}$.

The following proposition gives the total number of normal elements.

Proposition 4. [30] Normal elements of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ are in one-to-one correspondence to circulant matrices in $\mathrm{GL}_n(\mathbb{F}_q)$. Therefore the total number of normal elements of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ can be evaluated as $\Phi_q(n) = \Phi_q(x^n - 1)$.

The cryptosystem [1] uses a semilinear transformation φ over $\mathbb{F}_{q^n}/\mathbb{F}_{q^m}$ as part of the secret key. Let us then define the semilinear transformations and give some properties which form the backbone of the cryptosystem.

4.2 Semilinear transformations

Definition 10. A *basis vector* of $\mathbb{F}_{q^m}$ is any vector $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_m) \in \mathbb{F}_{q^m}^m$ such that $\{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$ form a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$.

For example, let us consider $q = 2, m = 6$ and the finite field $\mathbb{F}_{2^m} = \mathbb{F}_{2^6} \equiv \mathbb{F}_2[x]/(x^6 + x^4 + x^3 + x + 1) \equiv \mathbb{F}_2[a]$ where a is a root of $x^6 + x^4 + x^3 + x + 1$. The vector $(1, x, x^2, x^3, x^4, x^5, x^6)$ is a basis vector and so is each of its image by permutation. We use the term *polynomial* (resp. *normal*) for an element $\alpha \in \mathbf{F}_q^m$ such that $(1, \alpha, \alpha^2, \ldots, \alpha^{m-1})$ (resp. $(\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}})$) is a basis vector of $\mathbb{F}_{q^m}$.

The term *semi-linear mapping* is an established term in the literature ([31], [32],...). It refers to certain types of mappings between codes and is related to projective geometry. However, we will keep the terminology used by Guo and Fu [1].

Definition 11 (linear automorphism of $\mathbb{F}_{q^n}/\mathbb{F}_{q^m}$)**.** In [1], we consider integers m, n, k such that $k \leq n$ and $n = ml$ for some integer l. Therefore, $\mathbb{F}_{q^n}$ can be seen as a vector space over $\mathbb{F}_{q^m}$. Let $\mathbf{a} = \{\mathbf{a}_1, \ldots, \mathbf{a}_l\}$ and $\mathbf{b} = \{\mathbf{b}_1, \ldots, \mathbf{b}_l\}$ be two basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_{q^m}$. In practice, $\mathbf{a}$ is fixed and $\mathbf{b}$ will be randomly generated. We define the corresponding automorphism of $\mathbf{F}_{q^n}$ by mapping an element $\mathbf{v} = \sum_{i=1}^{l} \lambda_i \mathbf{a}_i \in \mathbb{F}_{q^n}$ (with $\lambda_i \in \mathbb{F}_{q^m}$) to

$$\varphi(\mathbf{v}) = \sum_{i=1}^{l} \lambda_i \varphi(\mathbf{a}_i) = \sum_{i=1}^{l} \lambda_i \mathbf{b}_i.$$

Such map, by design, is $\mathbb{F}_{q^m}$ linear. Henceforth, we will denote $Aut_{\mathbb{F}_{q^n}/\mathbb{F}_{q^m}}$ the set of all $\mathbf{F}_{q^m}$ linear automorphism of $\mathbf{F}_{q^n}$. For any matrix $\mathbf{M} \in GL_n\mathbb{F}_{q^m}$, $\mathbf{Ma}$ gives us a unique basis $\mathbf{b}$ and all basis $\mathbf{b}$ can be obtained this way. Therefore we have the following:

Proposition 5. The total number of $\mathbf{F}_{q^m}$ linear automorphisms of $\mathbf{F}_{q^m}$ is

$$|Aut_{\mathbb{F}_{q^n}/\mathbb{F}_{q^m}}| = |GL_n\mathbb{F}_{q^m}| = \prod_{i=0}^{l-1}(q^{ml} - q^{mi}).$$

Definition 12 (semi-linear automorphism of $\mathbb{F}_{q^n}/\mathbb{F}_{q^m}$)**.** Given $\varphi \in Aut_{\mathbb{F}_{q^n}/\mathbb{F}_{q^m}}$, for $\mathbf{v} = \{v_1, \ldots, v_n\} \in \mathbb{F}_{q^n}^n$, let $\varphi(\mathbf{v}) = \{\varphi(v_1), \ldots, \varphi(v_n)\}$. In the same way, for a subset $\mathcal{V} \in \mathbb{F}_{q^n}^n$, let $\varphi(\mathcal{V}) = \{\varphi(\mathbf{v}) : \mathbf{v} \in \mathcal{V}\}$, for a matrix $M = (M_{ij}) \in \mathcal{M}_{k,n}((\mathbb{F}_{q^n})$, let $\varphi(M) = (\varphi(M_{ij}))$. In these settings, φ is called a *semilinear transformation* over $\mathbb{F}_{q^n}/\mathbb{F}_{q^m}$.

Definition 13 (Fully linear automorphism)**.** Let $\mathcal{C} \subseteq \mathbb{F}_{q^m}^m$ be an $[n, k]$ linear code over $\mathbb{F}_{q^n}$ and $\varphi \in Aut_{\mathbb{F}_{q^n}}/\mathbb{F}_{q^m}$. If $\varphi(\mathcal{C})$ is also a linear code over $\mathbb{F}_{q^n}$, we say that φ is linear over $\mathcal{C}$. If φ is linear over all linear codes over $\mathbb{F}_{q^n}$, we say that φ is *fully linear* over $\mathbb{F}_{q^m}$.

Guo and Fu [1] proved the following properties for semi-linear transformations.

Proposition 6 (Properties of semilinear transformations)**.** Let φ be a *semilinear transformation* over $\mathbb{F}_{q^n}/\mathbb{F}_{q^m}$.

1. For a vector $\mathbf{v} \in \mathbb{F}_{q^n}^n$, $wt_R(\varphi(\mathbf{v})) = wt_R(\varphi(v))$, whether we take the rank weight with respect to $\mathbb{F}_{q^m}$ or $\mathbb{F}_q$, (Proposition 6 and Remark 4 in [1])

2. The image $\varphi(\mathcal{C})$ of a code $\mathcal{C} \subseteq \mathbb{F}_{q^m}^m$ is always a $\mathbb{F}_{q^m}$ and $\mathbb{F}_q$ linear space.

3. Let $\mathbf{a} = \{a_1 \ldots., a_l\}$ be a basis vector of $\mathbb{F}_{q^n}$ over $\mathbb{F}_{q^m}$ and $\varphi \in Aut_{\mathbb{F}_{q^n}/\mathbb{F}_{q^m}}$. Let $A = [\varphi(a_1\mathbf{a})^\top, \ldots, \varphi(a_l\mathbf{a})^\top]$, then φ is fully linear if and only if $Rank(A) = 1$. (Theorem 8 in [1]). The nonlinearity of φ with extension degree l is defined to be $\mathrm{NL}_m(\varphi) = \frac{Rank(A)}{l}$.

4. The total number of fully linear transformations in $Aut_{\mathbb{F}_{q^n}/\mathbb{F}_{q^m}}$ is $l(q^n - 1)$. (Theorem 10 [1])

5. A semi-linear transformation $\varphi \in Aut_{\mathbb{F}_{q^n}/\mathbb{F}_{q^m}}$ defines a polynomial in $\mathbb{F}_{q^n}[X]$ with coefficients in $\mathbb{F}_{q^m}$. That is, we can always write $\varphi(x) = \sum_{i=0}^{l-1} a_i \left(x^{(q^m)}\right)^i$. Then $\mathrm{NL}_l(\varphi) = \frac{w}{l}$ where w is the number of non-zero coefficients in $(a_1, \ldots, a_l)$. (Proposition 7 [1])

Let us now discover the cryptosystem proposed by Guo and Fu [1].

4.3 The cryptosystem

4.3.1 Description of the proposal

For a given target security level, the authors propose the parameters q, m, n, k, l, λ_1 and λ_2 where $n = lm$. Let $\mathbf{g} = (g^{q^{n-1}}, g^{q^{n-2}}, \ldots, g)$ be a normal basis vector of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, and $G = PC_k(\mathbf{g})$. Note that $\mathbf{g}$ and G are publicly available.

Algorithm 5: The Key Generation Algorithm

Input: $(q, n, m, k, l, \lambda_1, \lambda_2)$
Output: The public key $(\mathbf{g}^\star, t)$. and the private key $(\mathbf{m_1}, \mathbf{m_2}, \varphi)$
1 For $i = 1, 2$, randomly choose the vector $\mathbf{m_i} \in \mathbb{F}_{q^n}^n$ such that $wt_R(\mathbf{m_i}) = \lambda_i$ and $M_i = PC_n(\mathbf{m_i})$ is invertible.
2 Randomly choose a semilinear transformation φ over $\mathbb{F}_{q^n}/\mathbb{F}_{q^m}$ that is not fully linear.
3 $\mathbf{g}^\star = \varphi(\mathbf{g}M_1^{-1})M_2^{-1}$
4 $t = \left\lfloor \frac{n-k}{2\lambda_1\lambda_2} \right\rfloor$.
5 Return the public key $(\mathbf{g}^\star, t)$. and the private key $(\mathbf{m_1}, \mathbf{m_2}, \varphi)$

Algorithm 6: The Encryption Algorithm

Input: A plaintext $\mathbf{x} \in \mathbb{F}_{q^m}^k$ and a public key $(\mathbf{g}^\star, t)$
Output: The ciphertext $\mathbf{y}$ of $\mathbf{x}$ encrypted using$(\mathbf{g}^\star, t)$
1 Randomly choose $\mathbf{e} \in \mathbb{F}_{q^n}^n$ with $wt_R(\mathbf{e}) = t$.
2 Return $\mathbf{y} = \mathbf{x}PC_k(\mathbf{g}^*) + \mathbf{e}$.

Algorithm 7: The Decryption Algorithm

Input: A ciphertext $\mathbf{y}$ encrypted using $(\mathbf{g}^\star, t)$ and the secret key $(\mathbf{m}_1, \mathbf{m}_2, \varphi)$
Output: The plaintext $\mathbf{x}$ corresponding to $\mathbf{y}$
1 Compute $\mathbf{y}M_2 = x\varphi(GM_1^{-1}) + \mathbf{e}M_2 = \varphi(\mathbf{x}GM_1^{-1}) + \mathbf{e}M_2$
2 Compute $\mathbf{y}' = \varphi^{-1}(\mathbf{y}M_2)M_1 = \mathbf{x}G + \varphi^{-1}(\mathbf{e}M_2)M_1$.
3 $\mathbf{e}' = \varphi^{-1}(\mathbf{e}M_2)M_1$ (Note that $wt_R(\mathbf{e}') \leq \left\lfloor \frac{n-k}{2} \right\rfloor$)
4 Apply the fast decoder of $\langle G \rangle_{q^n}$ to $\mathbf{y}'$ to reveals $\mathbf{e}'$
5 Recover $\mathbf{x}$ by solving $\mathbf{x}G = \mathbf{y}' - \mathbf{e}'$.
6 Return $\mathbf{y}$

4.3.2 Notes on the security

The authors made the following notes on the security of the schemes. All normal elements can be obtained by multiplying any normal element with an invertible circulant matrix. Therefore, they can work equivalently in this scheme and the security of the cryptosystem does not depend on

$\mathbf{g}$. Using a fully linear φ leads to a weak scheme which have already been broken. We also should avoid using $\mathbf{m}_1$ or $\mathbf{m}_2$ in $\mathbf{F}_{q^m}$ because doing so will reduce breaking the cryptosystem to only solving for φ. Given φ and $\mathbf{m}_1$, one can always solve for $\mathbf{m}_2$. So the security depends on keeping φ and $\mathbf{m}_1$ secret.

The cryptosystem [1] is secure against the Overbeck's attack [33], Coggia-Couvreur attack [34], and the Loidreau's attack [35]. Those attacks take advantage of the algebraic structure of the public key and thanks to the semi-linear transformation, their techniques do not work here.

One can do a brute force attack to recover a valid duple $(\overline{\varphi}, \overline{\mathbf{m}_1})$ but since there are $\mathcal{N}(\overline{\varphi}) \approx q^{(l-1)n}$ nonequivalent semi-linear transformations, the brute force is not computationally feasible. It is also possible to attack the cryptosystem using a generic RD attack which time complexity is given below.

In the appendix A, we propose an implementation of this scheme in Macaulay2.

Attack	Complexity
[14]	$\mathcal{O}\left(\min\left\{m^3t^3q^{(t-1)(k+1)}, (k+t)^3t^3q^{(t-1)(m-t)}\right\}\right)$
[15]	$\mathcal{O}\left((n-k)^3m^3q^{\min\left\{t\left\lceil\frac{mk}{n}\right\rceil, (t-1)\left\lceil\frac{m(k+1)}{n}\right\rceil\right\}}\right)$
[16]	$\mathcal{O}\left((n-k)^3m^3q^{t\left\lceil\frac{m(k+1)}{n}\right\rceil-m}\right)$

Table 4.1: Best known combinatorial attacks on the RD problem. [1]

Attack	Condition and method used	Complexity
[15]	$\left\lceil\frac{(t+1)(k+1)-(n+1)}{t}\right\rceil \leqslant k$ Annulator polynomial	$\mathcal{O}\left(k^3t^3q^{t\left\lceil\frac{(t+1)(k+1)-(n+1)}{t}\right\rceil}\right)$
[17]	$m\binom{n-k-1}{t} \geqslant \binom{n}{t} - 1$	$\mathcal{O}\left(m\binom{n-p-k-1}{t}\binom{n-p}{t}^{\omega-1}\right)$, where $\omega = 2.81$ and $p = \min\left\{1 \leqslant i \leqslant n : m\binom{n-i-k-1}{t} \geqslant \binom{n-i}{t} - 1\right\}$
[18]	Maximal minors	$\mathcal{O}\left(\left(\frac{((m+n)t)^\top}{t!}\right)^\omega\right)$
[17]	$m\binom{n-k-1}{t} < \binom{n}{t} - 1$	$\mathcal{O}\left(q^{at}m\binom{n-k-1}{t}\binom{n-a}{t}^{\omega-1}\right)$ where $a = \min\left\{1 \leqslant i \leqslant n : m\binom{n-k-1}{t} \geqslant \binom{n-i}{t} - 1\right\}$
[18]	Maximal minors	$\mathcal{O}\left(\left(\frac{((m+n)t)^{t+1}}{(t+1)!}\right)^\omega\right)$

Table 4.2: Best known algebraic attacks on the RD problem. [1]

Chapter 5

A cryptosystem based on the Lee metric code: FuLeeca (NIST candidate)

FuLeeca is a Lee-metric code-based signature scheme introduced in [13] and proposed in the Additional Signature Submission for NIST-PQC. It has been broken using LWE attacks but we will present a different style of attack. It proposes easier implementation and smaller public key and signature sizes. We present the structure of FuLeeca and the idea behind it. We will focus more on the key generation process and show that the private key can be obtained from the public key only using a mixed integer non-convex programming. This attack is based on the structure of the private key and therefore may not be applied directly to other future Lee-metric code-based schemes. We also note here that the key generation algorithm described in [13] slightly differs from the reference implementation that was submitted. We will cover both the algorithm described in the paper, which we will refer to as "textbook FuLeeca" and the reference implementation.

Since FuLeeca is the first digital signature on our survey, let us go through the high-level components of a digital signature scheme. First, we have two parties: *the signer*, who call Alice, and the *verifier* which can be anyone. We also need three algorithms: the key generation, the signature generation, and the signature verification. In the key generation, Alice randomly samples a pair $\{s_k, p_k\}$ of a secret key s_k and its associated public key p_k. The signer, Alice, then publishes the

public key in a public repository. Given a message $\mathbf{m}$, the signer uses the signature generation algorithm to compute a signature using $\mathbf{m}$ and s_k as inputs. The verifier has access to the p_k and uses the verification algorithm to check the validity of the signature using p_k and the constraints imposed by the scheme.

5.1 Background

We work with the finite field $\mathbb{F}_p$ where p is a prime and represent it using the symmetric notation, that is $\mathbb{F}_p = \left\{-\frac{p-1}{2}, \ldots, 0, \ldots, \frac{p-1}{2}\right\}$. The maximum Lee weight of an element in $\mathbf{F}_p$ is $M = \frac{p-1}{2}$ and $\mathrm{sgn}(x)$ denotes the signum of the element x in symmetric notation. That is $\mathrm{sgn}(x) = 0$ if $x = 0$, $\mathrm{sgn}(x) = 1$ if $x > 0$ and $\mathrm{sgn}(x) = -1$ if $x < 0$. For two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{F}_p^n$, we denote the number of sign matches as $\mathrm{mt}(\mathbf{x}, \mathbf{y}) := |\{i \in \{1, \ldots, n\}\, s.t.\, \mathrm{sgn}(x_i) = \mathrm{sgn}(y_i), x_i \neq 0, y_i \neq 0\}|$.

The following definitions are taken directly from [13].

Definition 14 (Logarithmic Matching Probability (LMP))**.** For a fixed $\mathbf{v} \in \mathbb{F}_p^n$ and $\mathbf{y} \xleftarrow{\$} \{\pm 1\}^n$, the probability of $\mathbf{y}$ to have $\mu := \mathrm{mt}(\mathbf{y}, \mathbf{v})$ sign matches with $\mathbf{v}$ is $B\left(\mu, \mathrm{wt}_H(\mathbf{v}), 1/2\right)$, where $B(k, n, q)$ is the binomial distribution defined as $B(k, n, q) = \begin{pmatrix} n \\ k \end{pmatrix} q^k (1-q)^{n-k}$.

We will use the notation $\mathrm{LMP}(\mathbf{v}, \mathbf{y}) = -\log_2\left(B\left(\mu, \mathrm{wt}_H(\mathbf{v}), 1/2\right)\right)$. This function can be efficiently approximated via additions and subtractions of precomputed values of $\log_2(x!)$, i.e. using a look-up table. As n tends to infinity we have, for any $x \in \mathbb{F}_p$, the probability that one entry of $\mathbf{x}$ is equal to x is given by

$$p_w(x) = \frac{1}{Z(\beta)} \exp\left(-\beta \mathrm{wt}_L(x)\right)$$

where Z denotes the normalization constant and β is the unique solution to $w = \sum_{i=0}^{p-1} \mathrm{wt}_L(i) p_w(x)$.

Definition 15 (Typical Lee Set)**.** For a fixed weight w, and a rounding function f. we define the typical Lee set as

$$T(p, n, w) = \left\{\mathbf{x} \in \mathbb{F}_p^n \mid \mathbf{x}_i = x \text{ for } f\left(p_w(x) n\right) \text{ coordinates } i \in \{1, \ldots, n\}\right\}$$

That is the set of vectors for which the element x occurs $f\left(p_w(x) n\right)$ times. In principle, f could be simply chosen as the rounding function. The authors gave a rounding function to use for

the reference implementation. Because of this rounding, the elements of $T(p, n, w)$ do in general not have Lee weight w.

5.2 Textbook FuLeeca

5.2.1 Description of the scheme

Let Hash be a cryptographically secure hash function (examples include SHA256) and CSPRNG a cryptographically secure pseudo-random number generator (we can also use SHA256 for this). The algorithms for key generation, signature generation, and signature verification are specified in the documentation [13] as follows.

Algorithm 8: Textbook FuLeeca Key-generation

Input: Prime p, code length n, security level λ, Lee weight w_{key}
Output: public key $\mathbf{T}$, private key $\mathbf{G}_{\text{sec}}$

1 Choose $\mathbf{a}, \mathbf{b} \xleftarrow{\$} T(p, n/2, w_{\text{key}})$.
2 Construct cyclic matrix $\mathbf{A} \in \mathbb{F}_p^{n/2 \times n/2}$ from all shifts of $\mathbf{a}$. $\mathbf{A}$ needs to be invertible. If this is not the case, resample $\mathbf{a}$ according to Line 1.
3 Construct cyclic matrix $\mathbf{B} \in \mathbb{F}_p^{n/2 \times n/2}$ from all shifts of $\mathbf{b}$.
4 Generate the secret key $\mathbf{G}_{\text{sec}} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \end{pmatrix} \in \mathbb{F}_p^{n/2 \times n}$.
5 Calculate the systematic form $\mathbf{G}_{\text{sys}} = \begin{pmatrix} \mathbf{I}_{n/2} \mathbf{T} \end{pmatrix}$ of $\mathbf{G}_{\text{sec}}$ with $\mathbf{T} = \mathbf{A}^{-1}\mathbf{B}$.
6 Return public key $\mathbf{T}$, private key $\mathbf{G}_{\text{sec}}$

We note that for a given security level, the multiset of the Typical Lee Set is fixed. For a signature scheme, Alice does not need to "decode" any ciphertext so we do not need a special type of code that has fast decoding algorithms. The linear code used in FuLeeca is called "quasi-cyclic code" which is generated by a "quasi-cyclic" matrix: a matrix obtained by adjoining two cyclic matrices. The use of quasi-cyclic code results in efficiency: instead of storing a whole matrix for the secret key, Alice only needs to store one row. For a given NIST security level, FuLeeca proposes a significantly smaller public key and signature key size compared to the Classic McEliece cryptosystem. We provide an implementation of a toy example for FuLeeca on this web-page.

The parameters proposed for FuLeeca in the NIST submission were selected so that the public code looks random and it is easy to find codewords that have enough sign matches with a given message digest. If Eve would like to impersonate Alice, there are two things that she could try. The first one is to recover the secret key directly from the public key. This attack is called a

Algorithm 9: FuLeeca Signature Generation

Input: Secret key $\mathbf{a}, \mathbf{b}$, message $\mathbf{m}$, threshold ε, signature weight w_{sig}, key weight w_{key}, scaling factor $s \in \mathbb{R}$, security level λ, number of concentrating iterations n_{con}.

Output: salt, signature $\mathbf{y}$

$\mathbf{G}_{sec} \leftarrow (\mathbf{A}, \mathbf{B})$, $\mathbf{G} = \begin{pmatrix} \mathbf{G}_{sec} \\ -\mathbf{G}_{sec} \end{pmatrix}$ with rows $\mathbf{g}'_i$

$\mathbf{m}' \leftarrow \mathrm{Hash}(\mathbf{m})$

repeat

  $\mathrm{salt} \xleftarrow{\$} \{0,1\}^{256}$ `// Simple signing starts`

  $\mathbf{c} = (c_1, \ldots, c_n) \leftarrow \mathrm{CSPRNG}(\mathbf{m}' \| \mathrm{salt})$

  $c_i \leftarrow (-1)^{c_i} \quad \forall i$

  $\boldsymbol{x} \leftarrow (0, \ldots, 0)$

  for $i \leftarrow 1$ *to* $n/2$ **do**

    $x_{mt} = \mathrm{mt}\left(\boldsymbol{g}_i, \boldsymbol{c}\right) - \frac{\mathrm{wt}_H(\boldsymbol{g}_i)}{2}$

    $\boldsymbol{x}_i = \lfloor x_{mt} \rfloor$ `// Simple signing ends`

  $\mathcal{A} \leftarrow \{1, \ldots, n\}$ `// Allowed row index set`

  $\nu \leftarrow x\boldsymbol{G}_{\mathrm{sec}}$ `// Concentrating starts`

  $\nu' \leftarrow (0, \ldots, 0), i' = 0$

  $\mathrm{lf} \leftarrow 1$

  for $j \leftarrow 1$ *to* n_{con} **do**

    for $i \in \{1, \ldots, n\}$ **do**

      $\nu'' \leftarrow \nu + \boldsymbol{g}'_i$

      if $|\mathrm{LMP}\left(\boldsymbol{\nu}'', \boldsymbol{c}\right) - (\lambda + 64 + \varepsilon)| \leq |\mathrm{LMP}\left(\boldsymbol{\nu}', \boldsymbol{c}\right) - (\lambda + 64 + \varepsilon)|$ **then**

        if $i \in \mathcal{A} \| lf = 0$ **then**

          $\boldsymbol{\nu}' \leftarrow \boldsymbol{\nu}'', i' \leftarrow i$

    $w' \leftarrow \mathrm{wt}_L\left(\boldsymbol{\nu}'\right)$

    if $w' > w_{sig} - w_{key}$ **then**

      $lf \leftarrow 0$

    if $w' \leq w_{sig}$ **then**

      $\nu \leftarrow \nu'$

    if $i' \leq \frac{n}{2}$ **then**

      $\mathcal{A} \leftarrow \mathcal{A} \backslash \{i' + n/2\}$

    else

      $\mathcal{A} \leftarrow \mathcal{A} \backslash \{i' - n/2\}$

  if $\mathrm{wt}_L(\boldsymbol{\nu}) \leq w_{sig}$ &&$\mathrm{wt}_L(\boldsymbol{\nu}) > w_{sig} - 2w_{key}$ && $\mathrm{LMP}(\boldsymbol{\nu}, \boldsymbol{c}) \geq \lambda + 64$ **then**

    $[\boldsymbol{y}, \boldsymbol{yT}] \leftarrow \boldsymbol{\nu}$ return salt, $\mathrm{ENCODE}(\boldsymbol{y})$

  else

    go to Line 3 `// Concentrating ends`

until *A signature is returned*

"key recovery attack" and is the type of attack we propose. The other option is to forge a signature directly from the public key. For both options, Eve will need to find a codeword with a given Lee weight, which is solving Problem 3. The key recovery attack requires us to find two permutations of

Algorithm 10: FuLeeca Signature Verification

Input: signature (salt, $\mathbf{y}'$) message $\mathbf{m}$, public key $\mathbf{T}$, Lee weight w_{sig}
Output: Accept or Reject

$\mathbf{y} \leftarrow \text{DECODE}(\mathbf{y}')$
$\mathbf{m}' \leftarrow \text{Hash}(\mathbf{m})$
$\mathbf{c} = (c_1, \ldots, c_n) \leftarrow \text{CSPRNG}(\mathbf{m}' \| \text{ salt })$
$c_i \leftarrow (-1)^{c_i} \quad \forall i$
$\mathbf{v} = [\mathbf{yyT}]$
if $\text{wt}_L(\mathbf{v}) \leq w_{\text{sig}}$ *and* $\text{LMP}(\mathbf{v}, \mathbf{c}) \geq \lambda + 64$. **then**
  Accept
else
  Reject.

the Typical Lee Set, which is already fixed. However, for a signature forgery, we do not have prior knowledge about the form of the codeword and we have the additional constraint of having a given number of sign matches. Since the known attacks for solving Problem 3 are all based on ISD, our proposed attack based on Mixed Integer Non-Convex Programming provides a new perspective.

5.2.2 Our Attack

Let λ be a security level, $\mathbf{m}$ be the multiset of the typical Lee Set corresponding to λ viewed as a row vector, and $\mathbf{T}$ a given FuLeeca public key for that level. Suppose we have $\{\mathbf{a}, \mathbf{b}\}$ a valid secret key corresponding to $\mathbf{T}$. Note that $\{\mathbf{a}, \mathbf{b}\}$ is not unique because we can do a right shift on both $\mathbf{a}$ and $\mathbf{b}$ to get another valid and equivalent secret key. Since $\mathbf{a}$ and $\mathbf{b}$ are both elements of the Typical Lee Set, they are permutation images of $\mathbf{m}$ and can be written respectively as $\mathbf{m}P^a$ and $\mathbf{m}P^b$. Here, P^a and P^b are both permutation matrices so their entries are taken from $\{0, 1\}$ and only one element for each row and each column is 1. The vectors $\mathbf{a}$ and $\mathbf{b}$ are related to the fact that $\mathbf{b} = \mathbf{aT}$ in $\mathbb{F}_p$.

The key recovery of FuLeeca can be modeled as follows.

$$
\begin{aligned}
\min_{\mathbf{a},\mathbf{b}} \quad & 0 \\
\text{s.t.} \quad & \mathbf{aT} = \mathbf{b} \mod p, \\
& \mathbf{a} = \mathbf{mP_a}, \\
& \mathbf{b} = \mathbf{mP_b}, \\
& \sum_{j=1}^{n/2} \mathbf{P}^a_{ij} = 1 \, \forall i \in [1..n/2], \\
& \sum_{j=1}^{n/2} \mathbf{P}^b_{ij} = 1 \, \forall i \in [1..n/2], \\
& \sum_{i=1}^{n/2} \mathbf{P}^a_{ij} = 1 \, \forall j \in [1..n/2], \\
& \sum_{i=1}^{n/2} \mathbf{P}^b_{ij} = 1 \, \forall j \in [1..n/2], \\
& 0 \leq \mathbf{a}_i, \mathbf{b}_i \leq p-1 \, \forall i \in [1..n/2], \\
& \mathbf{a}_i, \mathbf{b}_i \in \mathbb{Z} \, \forall i \in [1..n/2], \\
& \mathbf{P}^a_{ij}, \mathbf{P}^b_{ij} \in \{0,1\} \forall i,j \in [1..n/2]
\end{aligned}
\tag{5.1}
$$

This modeling is mathematically correct. However, it is not in a format that can be solved by a computer. We then reformulate this basic modeling as follows.

1. Let $\mathbf{e}$ be an entry of one of the permutation matrices. Instead of imposing $\mathbf{e}$ to be a binary variable, we impose that $\mathbf{e}$ is a continuous variable between 0 and 1 such that $\mathbf{e}^2 - \mathbf{e} = 0$.

2. For the modulo equation, we use the transformation proposed by [36].

The transformation is described as follows:

Since $\mathbf{aT} = \mathbf{b} \mod p$, we have $0 \leq \mathbf{a}, \mathbf{b} \leq p-1$ and $\mathbf{aT} + \mathbf{yK} = \mathbf{b}$ for some vector $\mathbf{y} \in \mathbb{Z}^{n/2}$ where $\mathbf{K}$ is the diagonal matrix $p\mathbf{I}_{n/2}$. Let $\mathbf{P} = \begin{pmatrix} \mathbf{T} & \mathbf{K} \\ \mathbf{I}_{n/2} & \mathbf{0}_{n/2} \end{pmatrix}$. Assuming that $\mathbf{aT} = \mathbf{b} \mod p$ is consistent, the matrix $\mathbf{P}$ is an integer non-singular matrix. Therefore, $\mathbf{P}$ has a Hermite Normal Form $\mathbf{N}$. Let $\mathbf{Q}$ be the unimodular matrix such that $\mathbf{PQ} = \mathbf{N}$. Write $\mathbf{N} = \begin{pmatrix} \mathbf{R_1} & \mathbf{0} \\ \mathbf{R_2} & \mathbf{H} \end{pmatrix}$

and $\mathbf{Q} = \begin{pmatrix} \mathbf{Q_1} & \mathbf{Q_2} \\ \mathbf{Q_3} & \mathbf{Q_4} \end{pmatrix}$. Substituting $\begin{pmatrix} \mathbf{a} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{Q_1} & \mathbf{Q_2} \\ \mathbf{Q_3} & \mathbf{Q_4} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{z} \end{pmatrix}$ in the original constraints yields

$$\begin{cases} \mathbf{R_1 u} = \mathbf{b}, \\ 0 \leq \mathbf{R}_2\mathbf{u} + \mathbf{Hz} \leq (p-1)\mathbf{J_{n/2}} \quad \text{where } \mathbf{J_{n/2}} \text{ is the column vectors with all entry 1.} \\ \mathbf{u}, \mathbf{z} \in \mathbb{Z} \end{cases}$$

By transforming the key recovery attack on FuLeeca into a mixed integer problem, we can take advantage of commercial solvers and other improvements in the field of mixed integer programming to finish our attack.

We made our own implementation of the Notebook FuLeeca with the proposed attack in a Python library. All of the codes we explored can be accessed in this GitHub repository. Running the attack on the level 1 security level for NIST leads us to an overflow error when computing the Hermite Normal Form. This can be remediated by setting the type of variables to hold a larger range of values. Since the typical Lee set for a fixed value of p, n and w_{key} is fixed, we believe that the complexity of the computation should be similar to solving the same problem using an arbitrary multiset with low-weight elements. We illustrate our attack in Appendix B using a toy example with $p = 5$ and $n = 22$.

5.3 Reference Implementation of FuLeeca

5.3.1 Description of the scheme

For the implementation of FuLeeca, we can see the following difference from the textbook FuLeeca.

Typical Lee Set In the reference implementation, the typical Lee set refers to a list of Lee weights. That is to generate $\mathbf{a}$ and $\mathbf{b}$, we take the Typical Lee Set $\mathbf{S}$, perform a Fisher-Yates shuffle, and then randomly assign a sign to each of the entries of $\mathbf{a}$ and $\mathbf{b}$. The choice of Fisher-Yates shuffle is to prevent a side-channel timing attack during the key generation process and flipping the sign will make sure that achieving an invertible matrix $\mathbf{A}$ is always possible.

Data Types of the secret key In the reference implementation, $\mathbf{a}$ and $\mathbf{b}$ are viewed both as arrays and as polynomials in $\mathbb{F}_p$. That is $\mathbf{a} = \sum_{i=1}^{n/2} a_i x^{i-1}$ and $\mathbf{b} = \sum_{i=1}^{n/2} b_i x^{i-1}$. Instead of doing

matrix computation, polynomial operations were used. For example, the reference implementation uses the polynomial/array $\mathbf{u}$ such that $\mathbf{ua} \equiv 1 \mod [p, x^{n/2-1}]$ instead of $\mathbf{A}^{-1}$.

The presentation of the Public Key The public key of the reference implementation is obtained by multiplying $\mathbf{u}$, described above with $\mathbf{b}$. This results in a polynomial that is stored in an array instead of a matrix. However, the matrix $\mathbf{T}$ can be recovered by reconstructing the cyclic matrix.

5.3.2 Updated attack

In order to mitigate the difference between Textbook FuLeeca and the reference implementation, we need to make a few changes to the previous model.

Change in the definition of Typical Lee Set Now $\mathbf{a}$ and $\mathbf{b}$ are obtained by getting a permutation of $\mathbf{m}$ followed by a random sign flip. In order to model $\mathbf{a}$ and $\mathbf{b}$, we do the following $\mathbf{a} = \mathbf{a}_p - \mathbf{a}_m$ where $\mathbf{a}_p, \mathbf{a}_m \geq 0$, $\mathbf{a}_p = \mathbf{mP_{ap}}, \mathbf{a}_m = \mathbf{mP_{am}}$, where $\mathbf{P_{ap}}$ and $\mathbf{P_{am}}$ are matrices with entries in $\{0, 1\}$ that sum up into a permutation matrix.

Change in the representation of the public key We only need to recover the public key matrix from the public key array.

Our Python module attackfuleeca.py that can be viewed in the GitHub repository implements these changes and Appendix C illustrate its use using a toy example for $p = 5$ and $n = 11$.

5.4 Why are we interested in implementing a MIP attack?

Quantum computers improve the complexity of solving MIP [37], [38],[39]. Solving a general MIP is NP-hard. Ongoing research on using quantum computers to solve optimization problems, Commercial optimization solvers are getting better and better. Therefore, if a cryptosystem can be attacked using MIP techniques, all of those improvements should be taken into account.

5.5 Future work on our attack

For the textbook FuLeeca and the reference implementation, we give a proof of concept that a new attack can be used. Our future work in this direction includes getting a closer look at the

overflow error and mitigating it. This would allow us to perform the attack on instances in the NIST security levels. Once we are able to use the attack on level 1, we will generate 100 instances and attack them using our method. We will also explore the different ways to solve an MIP to see what works best with our modeling. Due to its sparsity, our model may be prone to numerical instability. Therefore, we will also put an effort into strengthening the model. The attack we proposed works because of the specific structure of the FuLeeca secret key. A good question to ask is then, can this attack be modified to be applied to other cryptosystems or solve other problems? We also plan on comparing the performance of our attack to the performance of the other attacks, namely the ISD attack and the LWE attack.

Chapter 6

Conclusions and Discussion

To conclude, the complexity of solving hard problems in coding theory gives advantages to code-based cryptosystems in the NIST PQC competition. To build a strong code-based cryptosystem, one must consider all the different attacks that can be applied to their scheme. For the McEliece cryptosystem, the best attacks are based on ISD and are all of exponential complexity. Code-based cryptography comes in different flavors and one needs to examine the rise and fall of each previously proposed scheme. Other advancements in other areas of maths should also be taken into account as they may lead to other styles of attacks. One of the areas of maths that is actually on the rise is machine learning. We speculate that it can also be used to attack code-based cryptosystems.

Appendices

Appendix A Our implementation of the attack based on Annulator polynomial

```
-- Macaulay2 code for solving a toy instance of the RD problem using the annulator polynomial
restart
K= ZZ/2
-- Defining Fq
R = K[ p0,p1, x1,x2,a, MonomialOrder=>Lex] -- The variables used
J = ideal(a^6 + a^4 + a^3 + a + 1)
S = R/J -- Fq^m
-- G1 and G2 are the rows of G
G1 = vector{1,a +1 ,a^2 +1, a^3  +1,a^4  +1, a^5 +1}
G2 = vector{1, a^2+1, a^4 +1, a^4+a^3 +a, a^5 +a^4+a^2+a, a^5 +a^4}
-- Computing y = xG +e
y = (1+a)*G1+ (a^2+a)*G2 +vector{1,a+1, 0,0,0,0}
-- This is the ciphertext we send
-- Now suppose we got only G1,G2 and y but not x and e
-- We can just copy and paste the value we received
y = vector{ a^3, a^4+a^3, a^5+a^4+a,a^4+a^3+a^2, a^5+a^4+1, a^5+a^3+a^2+a}
-- Specifying the value of z in the polynomial
z = y - x1*G1-x2*G2
-- Write the polynomials in the setting
L = for i from 0 to 5 list ( z_(i)^4 + p1*z_(i)^2 +p0*z_(i))
monomials L_(0)
I = ideal(L)
v = gens gb I -- Solving the system
```

Appendix B Our implementation of the Textbook FuLeeca with HNF and MIP attack

hnftextbookfuleecainuse

November 7, 2023

Textbook FuLeeca with HNF in use

The python module generate a key pair and write then in textfiles. Then it loads the public key, format it so that it can be fed into the Pyomo model. Finally, use the modules to attack the intsance and we can see that we get a valid secret key.

[1]:
```
#!pip install pyomo
#!pip install hsnf==0.3.13
```

[1]:
```
import TextbookFuLeecawithtNHF
from datetime import datetime
import pandas as pd
```

```
Solver you already have
['base', 'gurobi', 'highs']
```

[2]:
```
attack = TextbookFuLeecawithtNHF.attack(option = "quadratic")
```

```
WARNING: Implicitly replacing the Component attribute Ca (type=<class
'pyomo.core.base.constraint.IndexedConstraint'>) on block unknown with a new
Component (type=<class 'pyomo.core.base.constraint.IndexedConstraint'>). This
is usually indicative of a modelling error. To avoid this warning, use
block.del_component() and block.add_component().
WARNING: Implicitly replacing the Component attribute Cb (type=<class
'pyomo.core.base.constraint.IndexedConstraint'>) on block unknown with a new
Component (type=<class 'pyomo.core.base.constraint.IndexedConstraint'>). This
is usually indicative of a modelling error. To avoid this warning, use
block.del_component() and block.add_component().
```

[3]:
```
print(attack)
```

```
Hello, I am the python class used to attack FuLeeca,
 I can be used to generate a key pair for FuLeeca,
 I also can load a public key from a file,
 I format the public key so that it can be used in Pyomo,
, I use Pyomo to attack the FuLeeca cryptosystem and you got to choose which
solver to use!
```

0.1 Trying the attack on the smallest toy example

[4]:
```
attack.generate_key(level =0,verbose =True)
```

```
p is5
halfn is 11
Alice's a value: [1, 2, 0, 0, 2, 1, 1, 2, 0, 1, 1]
Alice's b value: [1 1 2 1 2 1 0 0 0 2 1]
Alice's T value:
[[2 3 0 4 3 2 1 3 1 4 3]
 [3 2 3 0 4 3 2 1 3 1 4]
 [4 3 2 3 0 4 3 2 1 3 1]
 [1 4 3 2 3 0 4 3 2 1 3]
 [3 1 4 3 2 3 0 4 3 2 1]
 [1 3 1 4 3 2 3 0 4 3 2]
 [2 1 3 1 4 3 2 3 0 4 3]
 [3 2 1 3 1 4 3 2 3 0 4]
 [4 3 2 1 3 1 4 3 2 3 0]
 [0 4 3 2 1 3 1 4 3 2 3]
 [3 0 4 3 2 1 3 1 4 3 2]]
This should be 1 if the sekret keys are valid
1
```

[5]:
```
T = attack.get_T('T.csv')
```

[6]:
```
print(attack.T_to_dat(T, level = 0))
```

```
Task done, see your file at toyexample.dat
param level:=0;
param p:= 5;
param uba:= 2;
param halfn:= 11;
param: i:
       Mset:=
      1 0
      2 0
      3 0
      4 1
      5 1
      6 2
      7 2
      8 2
      9 1
      10 1
      11 1;
param T:  1 2 3 4 5 6 7 8 9 10 11:=
        1 2 3 0 4 3 2 1 3 1 4 3
        2 3 2 3 0 4 3 2 1 3 1 4
```

```
        3 4 3 2 3 0 4 3 2 1 3 1
        4 1 4 3 2 3 0 4 3 2 1 3
        5 3 1 4 3 2 3 0 4 3 2 1
        6 1 3 1 4 3 2 3 0 4 3 2
        7 2 1 3 1 4 3 2 3 0 4 3
        8 3 2 1 3 1 4 3 2 3 0 4
        9 4 3 2 1 3 1 4 3 2 3 0
        10 0 4 3 2 1 3 1 4 3 2 3
        11 3 0 4 3 2 1 3 1 4 3 2;
param Q1:  1 2 3 4 5 6 7 8 9 10 11:=
        1 1 4 2 3 4 4 2 4 3 3 3
        2 3 3 0 2 3 4 4 2 4 3 3
        3 3 3 3 0 2 3 4 4 2 4 3
        4 1 2 3 3 0 2 3 4 4 2 4
        5 2 2 3 3 3 0 2 3 4 4 2
        6 4 0 3 3 3 3 0 2 3 4 4
        7 4 2 4 3 3 3 3 0 2 3 4
        8 4 4 2 4 3 3 3 3 0 2 3
        9 3 4 4 2 4 3 3 3 3 0 2
        10 2 3 4 4 2 4 3 3 3 3 0
        11 0 2 3 4 4 2 4 3 3 3 3;
param R1:  1 2 3 4 5 6 7 8 9 10 11:=
        1 1 0 0 0 0 0 0 0 0 0 0
        2 0 1 0 0 0 0 0 0 0 0 0
        3 0 0 1 0 0 0 0 0 0 0 0
        4 0 0 0 1 0 0 0 0 0 0 0
        5 0 0 0 0 1 0 0 0 0 0 0
        6 0 0 0 0 0 1 0 0 0 0 0
        7 0 0 0 0 0 0 1 0 0 0 0
        8 0 0 0 0 0 0 0 1 0 0 0
        9 0 0 0 0 0 0 0 0 1 0 0
        10 0 0 0 0 0 0 0 0 0 1 0
        11 0 0 0 0 0 0 0 0 0 0 1;
param Q2:  1 2 3 4 5 6 7 8 9 10 11:=
        1 5 0 0 0 0 0 0 0 0 0 0
        2 0 5 0 0 0 0 0 0 0 0 0
        3 0 0 5 0 0 0 0 0 0 0 0
        4 0 0 0 5 0 0 0 0 0 0 0
        5 0 0 0 0 5 0 0 0 0 0 0
        6 0 0 0 0 0 5 0 0 0 0 0
        7 0 0 0 0 0 0 5 0 0 0 0
        8 0 0 0 0 0 0 0 5 0 0 0
        9 0 0 0 0 0 0 0 0 5 0 0
        10 0 0 0 0 0 0 0 0 0 5 0
        11 0 0 0 0 0 0 0 0 0 0 5;
param J:  1 2 3 4 5 6 7 8 9 10 11:=
        1 -1 -4 -2 -3 -4 -4 -2 -4 -3 -3 -3
        2 -3 -3 0 -2 -3 -4 -4 -2 -4 -3 -3
```

```
        3 -3 -3 -3 0 -2 -3 -4 -4 -2 -4 -3
        4 -1 -2 -3 -3 0 -2 -3 -4 -4 -2 -4
        5 -2 -2 -3 -3 -3 0 -2 -3 -4 -4 -2
        6 -4 0 -3 -3 -3 -3 0 -2 -3 -4 -4
        7 -4 -2 -4 -3 -3 -3 -3 0 -2 -3 -4
        8 -4 -4 -2 -4 -3 -3 -3 -3 0 -2 -3
        9 -3 -4 -4 -2 -4 -3 -3 -3 -3 0 -2
        10 -2 -3 -4 -4 -2 -4 -3 -3 -3 -3 0
        11 0 -2 -3 -4 -4 -2 -4 -3 -3 -3 -3;
param H:  1 2 3 4 5 6 7 8 9 10 11:=
        1 5 0 0 0 0 0 0 0 0 0 0
        2 0 5 0 0 0 0 0 0 0 0 0
        3 0 0 5 0 0 0 0 0 0 0 0
        4 0 0 0 5 0 0 0 0 0 0 0
        5 0 0 0 0 5 0 0 0 0 0 0
        6 0 0 0 0 0 5 0 0 0 0 0
        7 0 0 0 0 0 0 5 0 0 0 0
        8 0 0 0 0 0 0 0 5 0 0 0
        9 0 0 0 0 0 0 0 0 5 0 0
        10 0 0 0 0 0 0 0 0 0 5 0
        11 0 0 0 0 0 0 0 0 0 0 5;
```

[7]:
```
attack.forge_lin_sk(solvername = 'gurobi', ampl = False,verbose = True)
```

```
[    0.00] starting timer
We just have built the instance, start solving stay tuned!
Solver script file: '/local_scratch/pbs.1520213.pbs02/tmpkd_sn23w.gurobi.script'
Solver log file: 'my.log'
Solver solution file: '/local_scratch/pbs.1520213.pbs02/tmpb7q0a_oy.gurobi.txt'
Solver problem files: ('/local_scratch/pbs.1520213.pbs02/tmptt67k6gv.pyomo.lp',)
feasible
0
Verifying that indeed aT = b mod p
 This should be 1 if the sekret keys are valid:

1
[+   3.05] task 1
elapsed time: 3.0 s
Value of a[0, 1, 2, 2, 0, 1, 2, 1, 1, 1, 0]
Value of b[0, 0, 1, 2, 1, 2, 1, 1, 1, 0, 2]
Value of aT[0 0 1 2 1 2 1 1 1 0 2]
```

[7]:
```
[1, 3.047435585409403]
```

Appendix C Our implementation of the Reference FuLeeca with HNF and MIP attack

newfuleecainuse

November 7, 2023

Reference implementation in use

The python module generate a key pair and write then in textfiles. Then it loads the public key, format it so that it can be fed into the Pyomo model. Finally, use the modules to attack the intsance and we can see that we get a valid secret key.

[1]:
```
#!pip install pyomo
#!pip install hsnf==0.3.13
```

[1]:
```
import attackfuleeca
from datetime import datetime
import pandas as pd
```

```
Solver you already have
['base', 'gurobi', 'highs']
```

[2]:
```
attack = attackfuleeca.attack(option = "quadratic")
```

[3]:
```
print(attack)
```

```
Hello, I am the python class used to attack FuLeeca,
 I can be used to generate a key pair for FuLeeca,
 I also can load a public key from a file,
 I format the public key so that it can be used in Pyomo,
, I use Pyomo to attack the FuLeeca cryptosystem and you got to choose which
solver to use!
```

[4]:
```
attack.generate_key(level =0,verbose =True)
```

```
p is5
halfn is 11
Alice's a value: [ 0  1 -1 -1 -2  1 -2  0  1  0  2]
Alice's b value: [ 0  2  0  1 -2 -1 -2 -1  0 -1  1]
Alice's T value:
[[1 1 1 1 0 2 0 1 1 3 2]
 [2 1 1 1 1 0 2 0 1 1 3]
 [3 2 1 1 1 1 0 2 0 1 1]
 [1 3 2 1 1 1 1 0 2 0 1]
 [1 1 3 2 1 1 1 1 0 2 0]
```

```
 [0 1 1 3 2 1 1 1 1 0 2]
 [2 0 1 1 3 2 1 1 1 1 0]
 [0 2 0 1 1 3 2 1 1 1 1]
 [1 0 2 0 1 1 3 2 1 1 1]
 [1 1 0 2 0 1 1 3 2 1 1]
 [1 1 1 0 2 0 1 1 3 2 1]]
This should be 1 if the sekret keys are valid
1
```

[6]:
```
T = attack.get_T('T.csv')
```

[7]:
```
print(attack.T_to_dat(T, level = 0))
```

```
Task done, see your file at toyexample.dat
param level:=0;
param p:= 5;
param uba:= 2;
param halfn:= 11;
param: i:
       Mset:=
      1 0
      2 0
      3 0
      4 1
      5 1
      6 2
      7 2
      8 2
      9 1
      10 1
      11 1;
param T:  1 2 3 4 5 6 7 8 9 10 11:=
        1 3 0 0 2 0 0 0 3 4 3 0
        2 0 3 0 0 2 0 0 0 3 4 3
        3 3 0 3 0 0 2 0 0 0 3 4
        4 4 3 0 3 0 0 2 0 0 0 3
        5 3 4 3 0 3 0 0 2 0 0 0
        6 0 3 4 3 0 3 0 0 2 0 0
        7 0 0 3 4 3 0 3 0 0 2 0
        8 0 0 0 3 4 3 0 3 0 0 2
        9 2 0 0 0 3 4 3 0 3 0 0
        10 0 2 0 0 0 3 4 3 0 3 0
        11 0 0 2 0 0 0 3 4 3 0 3;
param Q1:  1 2 3 4 5 6 7 8 9 10 11:=
        1 0 0 0 0 0 0 0 0 0 0 0
        2 4 2 4 4 1 3 3 0 1 0 0
        3 4 1 1 3 0 4 1 3 1 1 0
        4 3 0 4 4 3 2 1 0 3 0 0
```

```
        5 4 0 4 3 0 1 0 1 1 3 0
        6 1 3 1 0 1 0 1 2 4 3 0
        7 1 0 4 2 3 1 0 3 0 1 0
        8 3 2 3 2 2 0 3 4 3 4 0
        9 0 1 2 3 4 1 4 4 1 4 0
        10 0 3 1 2 0 3 0 0 1 2 0
        11 2 0 0 3 1 1 4 3 4 4 0;
param R1:  1 2 3 4 5 6 7 8 9 10 11:=
        1 1 0 0 0 0 0 0 0 0 0 0
        2 0 1 0 0 0 0 0 0 0 0 0
        3 0 0 1 0 0 0 0 0 0 0 0
        4 0 0 0 1 0 0 0 0 0 0 0
        5 0 0 0 0 1 0 0 0 0 0 0
        6 0 0 0 0 0 1 0 0 0 0 0
        7 0 0 0 0 0 0 1 0 0 0 0
        8 0 0 0 0 0 0 0 1 0 0 0
        9 0 0 0 0 0 0 0 0 1 0 0
        10 0 0 0 0 0 0 0 0 0 1 0
        11 4 4 4 4 4 4 4 4 4 4 5;
param Q2:  1 2 3 4 5 6 7 8 9 10 11:=
        1 1 0 0 0 0 0 0 0 0 0 0
        2 1 5 0 0 0 0 0 0 0 0 0
        3 1 0 5 0 0 0 0 0 0 0 0
        4 1 0 0 5 0 0 0 0 0 0 0
        5 1 0 0 0 5 0 0 0 0 0 0
        6 1 0 0 0 0 5 0 0 0 0 0
        7 1 0 0 0 0 0 5 0 0 0 0
        8 1 0 0 0 0 0 0 5 0 0 0
        9 1 0 0 0 0 0 0 0 5 0 0
        10 1 0 0 0 0 0 0 0 0 5 0
        11 1 0 0 0 0 0 0 0 0 0 5;
param J:  1 2 3 4 5 6 7 8 9 10 11:=
        1 0 0 0 0 0 0 0 0 0 0 0
        2 -4 -2 -4 -4 -1 -3 -3 0 -1 0 0
        3 -4 -1 -1 -3 0 -4 -1 -3 -1 -1 0
        4 -3 0 -4 -4 -3 -2 -1 0 -3 0 0
        5 -4 0 -4 -3 0 -1 0 -1 -1 -3 0
        6 -1 -3 -1 0 -1 0 -1 -2 -4 -3 0
        7 -1 0 -4 -2 -3 -1 0 -3 0 -1 0
        8 -3 -2 -3 -2 -2 0 -3 -4 -3 -4 0
        9 0 -1 -2 -3 -4 -1 -4 -4 -1 -4 0
        10 0 -3 -1 -2 0 -3 0 0 -1 -2 0
        11 -2 0 0 -3 -1 -1 -4 -3 -4 -4 0;
param H:  1 2 3 4 5 6 7 8 9 10 11:=
        1 1 0 0 0 0 0 0 0 0 0 0
        2 1 5 0 0 0 0 0 0 0 0 0
        3 1 0 5 0 0 0 0 0 0 0 0
        4 1 0 0 5 0 0 0 0 0 0 0
```

```
5 1 0 0 0 5 0 0 0 0 0 0
6 1 0 0 0 0 5 0 0 0 0 0
7 1 0 0 0 0 0 5 0 0 0 0
8 1 0 0 0 0 0 0 5 0 0 0
9 1 0 0 0 0 0 0 0 5 0 0
10 1 0 0 0 0 0 0 0 0 5 0
11 1 0 0 0 0 0 0 0 0 0 5;
```

[8]:
```
attack.forge_lin_sk(solvername = 'gurobi', ampl = False,verbose = True)
```

```
[    0.00] starting timer
We just have built the instance, start solving stay tuned!
Solver script file: '/local_scratch/pbs.1517163.pbs02/tmphqop_l4n.gurobi.script'
Solver log file: 'my.log'
Solver solution file: '/local_scratch/pbs.1517163.pbs02/tmpppyreho0.gurobi.txt'
Solver problem files: ('/local_scratch/pbs.1517163.pbs02/tmp06prv5m0.pyomo.lp',)
feasible
0
Verifying that indeed aT = b mod p
 This should be 1 if the sekret keys are valid:

1
[+   7.60] task 1
elapsed time: 7.6 s
Value of a[1, 0, 1, 0, 0, 1, 2, 1, 2, 2, 1]
Value of b[0, 2, 0, 1, 1, 2, 3, 1, 0, 1, 4]
Value of aT[0 2 0 1 1 2 3 1 0 1 4]
```

[8]:
```
[1, 7.602939046919346]
```

Bibliography

[1] Wenshuo Guo and Fangwei Fu. Semilinear transformations in coding theory and their application to cryptography. *CoRR*, abs/2107.03157, 2021.

[2] Joel Greenberg. *The Enigma machine*. Oxford University Press, January 2017.

[3] NIST. Advanced encryption standard (AES). Technical report, 2001.

[4] W. Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, 1988.

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

[6] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[7] Peter W. Shor. Polynomial time algorithms for discrete logarithms and factoring on a quantum computer. In *Lecture Notes in Computer Science*, pages 289–289. Springer Berlin Heidelberg, 1994.

[8] Ibm unveils 433-qubit osprey chip. https://spectrum.ieee.org/ibm-quantum-computer-osprey, 2023. Accessed: 2023/10/02.

[9] Dustin Moody. Round 2 of the nist pqc "competition". https://csrc.nist.gov/CSRC/media/Presentations/Round-2-of-the-NIST-PQC-Competition-What-was-NIST/images-media/pqcrypto-may2019-moody.pdf, 2019. Accessed: 2023/5/16.

[10] Classic mceliece nist submissions. `https://classic.mceliece.org/nist.html`. Accessed: 2023/2/5.

[11] Robert J. McEliece. A public key cryptosystem based on algebraic coding theory. 1978.

[12] Round 1 additional signatures. `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`, 2023. Accessed: 08/18/2023.

[13] Stefan Ritterhoff, Georg Maringer, Sebastian Bitzer, Violetta Weger, Patrick Karl, Thomas Schamberger, Jonas Schupp, and Antonia Wachter-Zeh. Fuleeca: A lee-based signature scheme. Cryptology ePrint Archive, Paper 2023/377, 2023. `https://eprint.iacr.org/2023/377`.

[14] A Ourivski and Thomas Johansson. New techniques for decoding codes in rank metric and its cryptographic applications. *Problems of Information Transmission*, 38(3):237–246, 2002.

[15] Philippe Gaborit, Olivier Ruatta, and Julien Schrek. On the complexity of the rank syndrome decoding problem. *IEEE Transactions on Information Theory*, 62(2):1006–1019, 2016.

[16] Nicolas Aragon, Philippe Gaborit, Adrien Hauteville, and Jean-Pierre Tillich. A new algorithm for solving the rank syndrome decoding problem. *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2421–2425, 2018.

[17] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier Verbel. Improvements of algebraic attacks for solving the rank decoding and minrank problems. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 507–536, Cham, 2020. Springer International Publishing.

[18] Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich. An algebraic attack on rank metric code-based cryptosystems. *CoRR*, abs/1910.00810, 2019.

[19] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[20] Naoto Kimura, Atsushi Takayasu, and Tsuyoshi Takagi. Memory-efficient quantum information set decoding algorithm. In Leonie Simpson and Mir Ali Rezazadeh Baee, editors, *Information Security and Privacy*, pages 452–468, Cham, 2023. Springer Nature Switzerland.

[21] N. Patterson. The algebraic decoding of goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[22] Jean-Charles Faugère, Valérie Gauthier-Umanã, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high rate mceliece cryptosystems. In *2011 IEEE Information Theory Workshop*, pages 282–286, 2011.

[23] Rollo nist submission. `https://pqc-rollo.org/`. Accessed: 2023/5/16.

[24] Rqc nist submission. `http://pqc-rqc.org/`. Accessed: 2023/5/16.

[25] Ernst Gabidulin. Theory of codes with maximum rank distance (translation). *Problems of Information Transmission*, 21:1–12, 01 1985.

[26] Pierre Loidreau. A welch–berlekamp like algorithm for decoding gabidulin codes. volume 3969, pages 36–45, 03 2005.

[27] G. Richter and Simon Plass. Error and erasure decoding of rank-codes with a modified berlekamp-massey algorithm. 01 2004.

[28] Roger Chalkley. Circulant matrices and algebraic equations. *Mathematics Magazine*, 48(2):73–80, 1975.

[29] Ayoub Otmani, Jean-Pierre Tillich, and Léonard Dallot. Cryptanalysis of two mceliece cryptosystems based on quasi-cyclic codes. *CoRR*, abs/0804.0409, 2008.

[30] Gary Mullen and Daniel Panario. Handbook of finite fields. *Handbook of Finite Fields*, 06 2013.

[31] Zihui Liu and Xiangyong Zeng. Further results on the semilinear equivalence of linear codes. *Information Sciences*, 221:571–578, 2013.

[32] Zihui Liu and Jie Wang. The relative generalized hamming weight and the semilinear equivalence of codes. *SCIENCE CHINA Information Sciences*, 54:787–794, 04 2011.

[33] Raphael Overbeck. A new structural attack for gpt and variants. In Ed Dawson and Serge Vaudenay, editors, *Progress in Cryptology – Mycrypt 2005*, pages 50–63, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[34] Daniel Coggia and Alain Couvreur. On the security of a loidreau's rank metric code based encryption scheme. *CoRR*, abs/1903.02933, 2019.

[35] Analysing the key recovery complexity for a rank-metric code-based cryptosystem. `https://drive.google.com/file/d/1FuMgqm0NfGMJOxaZyrIrI1OWn0UICwPo/view`.

[36] GORDON H. BRADLEY. Modulo optimization problems and integer linear programming. In S.K. Zaremba, editor, *Applications of Number Theory to Numerical Analysis*, pages 433–451. Academic Press, 1972.

[37] Shouvanik Chakrabarti, Pierre Minssen, Romina Yalovetzky, and Marco Pistoia. Universal quantum speedup for branch-and-bound, branch-and-cut, and tree-search algorithms, 2022.

[38] Abolfazl Jalilvand and Sohrab Khanmohammadi. A new method for constructing the search tree in branch and bound algorithm. In *2005 Pakistan Section Multitopic Conference*. IEEE, dec 2005.

[39] Rhonda Au-Yeung, Nicholas Chancellor, and Pascal Halffmann. NP-hard but no longer hard to solve? using quantum computing to tackle optimization problems. *Frontiers in Quantum Science and Technology*, 2, feb 2023.