Post Quantum Cryptography with random split of St-Gen codes

Samy Saad Samy Shehata

Master in Security and Mobile Computing
Submission date: June 2017
Supervisor: Danilo Gligoroski, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

To my late father, my mother and my sister. If ever I had a reason to succeed, it is you.

Preface

This thesis titled ”Post Quantum Cryptography with Random Split of ST-Gen Codes”, is submitted in discussion of cryptographic systems in general, and one in particular, under a post quantum threat. It was written in fulfillment of the graduation requirements for the Security and Mobile Computing – NordSecMob – master’s program at the Norwegian University of Science and Technology. The writing of this thesis was concluded on the 6^{th} of June, 2017.

This thesis was carried out under the supervision of Professor Danilo Gligoroski of NTNU, and is based on his previous work in the field of post quantum cryptography. It was also remotely co-supervised by Professor Antti Yl-Jski from the University of Aalto.

First I would like to thank the Erasmus Mundus Programme, and all those involved with it, for facilitating this great opportunity. I give my thanks to Professor Danilo, without whose patient explanations, I would have been utterly lost amidst cryptographic hieroglyphics. I would also be remiss not to thank Mona Nordone, the NordSecMob program coordinator at NTNU, who I have often turned to for help, only to find her generous with advice and assistance. Finally, I would like to thank my friends and family. You are a constant reminder of the value of undertaking such endeavours.

Thank you each, thank you all.

Table of Contents

List of Tables

List of Figures

Abbreviations

ISD	=	Information Set Decoding
St-Gen	=	Staircase Generator
RST-Gen	=	Randomly split staircase generator

Chapter 1

Introduction

Cryptography has been historically defined as "the art of writing or solving codes". It remains one of the oldest sciences in existence today. Based on the history outlined in *The Codebreakers* by David Kahn, we can see early limited uses of cryptography by the ancient Egyptians around 4000 years ago. Fast forwarding to the 20^{th} century, we can see cryptography play a huge role in the determining the outcome of both world wars.

Throughout history, cryptography was predominantly the industry of militant and political parties, meant to be used for protecting state secrets and war strategies. As computers and communication systems began to be commercialized, demand arose for security measures that can protect digital information. Efforts to meet the demand started with the work of Feistel and IBM in the 1970s, which lead to the introduction of Data Encryption Standard (DES) in 1977. This was the birth moment of symmetric cryptography and wide spread use of encryption in the private sector. (Menezes et al., 1996)

In 1976, Deffie and Hellman revolutionized cryptography once again, when they published their paper, *New Directions in Cryptography*, where they propose the idea of public key cryptography and introduce a new method for key exchange. It was not until 1978 that Rivest, Shamir and Adleman introduced RSA as the first practical implementation of public key cryptography. To this day, there are no known practical attacks that can render RSA insecure. Another achievement that can be attributed to public key cryptography is digital signatures, which widen the influence of cryptography from just the domain of confidentiality, to include both authentication and integrity.

The widespread use of public key cryptography necessitated new efforts to find efficient ways for solving the factorization problem and the discrete logarithm problem, two hard mathematical problems that provide the corner stone for RSA security. While there have been no successful attempt to solve those problems using classic computers, those efforts finally bore fruit in 1994, when Peter Shor demonstrated an algorithm for efficient factorization using quantum computers. (Buchmann and Ding, 2008)

Thus, the advent of quantum computers threatens to make the current cryptographic and security infrastructure, relied on by millions of internet users, both obsolete and irrelevant. This thesis is a small part of a wide spread, organized and consistent effort to keep the current cryptographic infrastructure one step ahead of malicious attacks.

1.1 Background

In asymmetric cryptography, also known as public key cryptography, each party must have two separate keys. If Alice and Bob would like to communicate securely, using public key cryptography, each of them would generate a private key only known to its owner and a public key that is known to each other (and possibly publicly available). For Alice to send a message to Bob, she would use Bob's public key to encrypt the message, at which point only Bob can decrypt that message since he is the only one with the private key.

For authentication, before encrypting the message, Alice can use her own private key to add a digital signature to the message. After decrypting the message, Bob can use Alice's public key to verify that the message, or rather the signature, really came from her. Figures **1.1** and **1.2** summarizes the above process. RSA is one of the most popular public key cryptography systems today.

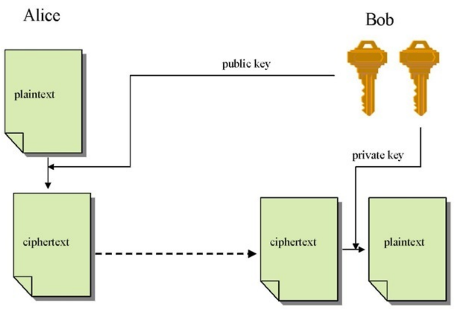

Figure 1.1: Public key encryption.

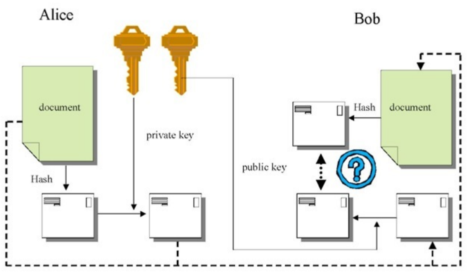

Figure 1.2: Public key signature.

RSA security is built around the factorization problem and the discrete logarithm problem. Two hard mathematical problems with no efficient algorithms to solve using classical computers. The two problems are defined as follows:

The Factorization Problem The factorization problem is defined as follows: Let n be a composite natural number such that $n \in N$ and $n = pq$ where p and q are two unique prime numbers. By finding p and q in polynomial time, RSA can be broken. The factoring problem is considered a hard problem.

The Discrete Logarithm Problem The discrete logarithm problem is defined as follows: Given α, β such that $\beta = \alpha^a$, find a in polynomial time. This is considered a hard problem if the given numbers are large enough.

Since its conception, RSA was considered unbreakable for lack of efficient solutions for those problems. This is no longer the case, since the introduction of Shor's algorithm, capable of efficiently factorizing a number using a quantum computer. In classical computers, information is encoded in bits as the most basic unit of data. The state of a single bit is always defined as either 0 or 1 exclusively, *i.e* an n bit word can be represented by a string of $0s$ and $1s$. In quantum computers, the basic unit of information is the qubit. It can be represented by an atom, also in one of two different states, detonated as 0 or 1.

In contrast with classical computers, however, qubits can exist in a superposition of the two states, as in one qubit can have both states 0 and 1 at a single point in time, each with a given probability. This means that a quantum computer with just two qubits can have four states at a single point in time and act on all of them simultaneously. This has been shown to offer great speedups in many areas of computations such as solving hard mathematical problems.

Here we give a brief overview of Shor's algorithm taken from Buchmann and Ding (2008). For a detailed discussion of the correctness of Shor's algorithm, the reader is refered to Buchmann and Ding (2008). For an input of a composite number $n \in N$, the algorithm does the following steps:

1. Pick $x \in 2, \ldots, n-1$ uniformly at random.
2. If $gcd(x, n) \neq 1$, return $gcd(x, n)$.
3. Find period r of $f(a) = x^a mod\, n$.
   (This part requires quantum computation.)
4. if r is odd or $x^{r/2} \equiv -1 mod\, n$, Goto 1.
5. return $gcd(x^{r/2} \pm 1, n)$

Post-quantum cryptography is a field of study that aims to update the current cryptographic primitives such that it remains secure with the existence of quantum computers. Research efforts in this field revolve around studying cryptosystems that make use of different problems, other than the two already mentioned, that remain hard to solve even by an attacker armed with quantum computing. Several classes of cryptographic algorithms in existence today are already believed to resist quantum attacks (Chen et al., 2016). Those classes are:

- **Code based cryptosystems** are encryption algorithms that use the same principles of retrieving the original bits of information transmitted over a noisy channel. This is done by encoding the signal with a specific scheme that can be recovered up to

a certain number of errors in transmission. This can be translated to the problem of encryption by adding some artificial noise to the secret message that can then be decrypted if specific information about the coding scheme is known.
Example: McEliece hidden-Goppa-code public key encryption. (1978)

- **Lattice-based cryptosystems** rely on the hardness of solving certain problems in multidimensional lattices.
  Example: Hoffstein-Pipher-Silverman ”NTRU” public-key-encryption system. (1998)
- **Multivariate public key cryptosystems** make use of random sets of quadratic equations where the encryption/decryption procedure uses the evaluation of these equations at certain points. These algorithms security rely on the hardness of solving such equations simultaneously.
  Example: Patarin's “$HFEv$-” public key signature system. (1996)
- **Hash-based cryptography** uses hash functions to ensure the integrity of messages.
  Example: Merkle's hash-tree public-key signature system. (1979)

Of the existing quantum resistant cryptographic paradigms, this thesis will mainly deal with code based cryptography, specifically the McEliece cryptosystem. Coding based cryptography is built on coding theory introduced by Claude Shannon in 1948. Coding theory was originally need to be able to retrieve an original message that has been modified due to transmission errors after being sent through a noisy channel.

A simple way to achieve this would be to duplicate the message multiple times and use a majority vote method at the destination to determine the original message. This, however, is not efficient. While redundancy is needed to be able to recover from errors, this redundancy can have a better structure that allows correcting errors with less extra information sent.

This concept is easily adaptable to cryptography. In cryptography, the plain text can be thought of as the original message. The encryption process would include encoding the message with a certain structure before adding some artificial noise giving the cipher text. Only the intended receiver, by knowing additional information about the encoding scheme can retrieve the original message.

Code based cryptography suffers from certain disadvantages that prevent it form competing with RSA on a level playing field. The McEliece cryptosystem for example has much larger memory requirements and large key sizes compared to RSA. However, it is expected to see a gain in popularity as we move into a post quantum cryptographic era.

The recent work done by Gligoroski et al. (2014), introduced a new family of linear codes known as staircase-generated codes. These are (n, k) binary codes that support a fast list decoding algorithm. The efficiency of the decoding algorithm is a result of the random step wise structure of the generator matrix. ST-Gen codes decoding algorithm can correct up to $\frac{n}{2}$ errors in a bounded channel with error density ρ.

In Moody and Perlner (2016), it was demonstrated that both schemes are susceptible to key recovery attacks using Information Set decoding and a distinguishing strategy similar to the one used in Sendrier and Tillich (2014). Information Set Decoding is a technique to find the error vector used to encrypt a message m. It works by selecting a sub matrix of the generator matrix G, called G_I. Then determining error pattern e_I that corresponds to the error vector with respect to I. Finally, m can be calculated as $m = (c_I + e_I)G_I^{-1}$. c_I is a part of the cipher text that corresponds to I.

To remedy this flaw, a new technique is developed that splits the generator matrix exposed in the public key randomly into several matrices. These requires the addition of a new security parameter s, specifying the number of splits. A new encryption and a digital signature schemes were developed in Gligoroski and Samardjiska (2016a), Gligoroski and Samardjiska (2016b) and Samardjiska and Gligoroski (2016) using this new technique. This thesis will provide a practical C code implementation of this modification and show case the algorithms new performance.

1.2 Terminology

Block Codes In coding theory, a block code is any error correcting codes that operates on data in blocks. It is considered one of the most important and commonly used family of error correcting codes.

Codeword The result of applying an error correcting code to a plain text message.

Error-correcting-code An error correcting code is the name given to any encoding scheme used to attach extra information to a message. This redundant information can be used to recover the message in the case of transmission errors.

Generator Matrix A generator matrix of a linear code C is any matrix G such that $\forall s, w = sG$ where w is a codeword in C.

Hamming Distance The hamming distance is defined as the number of positions in which two strings of equal length vary in symbols. For example, the two strings 1100 and 1010 have a hamming distance of 2.

Hamming Weight The hamming weight of a string is defined as the number of positions that are not equal to the null-symbol of the alphabet used. For example, using the binary alphabet $\{0, 1\}$, the string 1001 and all its permutations would have a hamming weight of 2.

Linear Binary Codes A linear code is any error correcting code where the linear combination of any codewords is also a codeword. If the linear code is defined over a binary alphabet, it is called a linear binary code.

1.3 Objectives

As the current cryptographic primitives are shown to be vulnerable to quantum attacks, new primitives will need to be developed to maintain our current level of security. Two such algorithms are introduced in Gligoroski et al. (2014). The first one is an encryption scheme and the second one is a digital signature scheme, both are based on the McEliece cryptosystem.

These schemes make use of a specific sub family of error correcting codes known as stair-case generator codes which is a subgroup of binary Goppa codes. These codes allow decoding of a given error set with overwhelming probability. The name comes from the distinct random stepwise block structure of the generator matrix. This gives way to a very efficient list decoding algorithm.

These schemes, however, were proven to be non secure due to vulnerability of their public keys to distinguishing strategies as was shown by Sendrier and Tillich (2014). In Moody and Perlner (2016), a practical ISD attack is demonstrated for full key recovery. This is inherent to the fact that a public key scheme based on staircase generator codes must, in some form, expose the staircase generator matrix as part of the public key, making it susceptible to structural attacks.

In order to address this vulnerability a modification to the above schemes is proposed in Samardjiska and Gligoroski (2016) and Gligoroski and Samardjiska (2016a). The proposed solution is a random split of the generator matrix exposed in the public key. This introduces a new parameter s which determines the number of splits. By applying this modification it can be shown that the probability of an attacker mounting a successful ISD attack as described by Moody and Perlner (2016) and Sendrier and Tillich (2014) is negligible.

The purpose of this thesis is provide practical C code implementations of the two schemes using concrete parameter sets given in Samardjiska and Gligoroski (2016) and Gligoroski and Samardjiska (2016a). This is followed by an analysis of the resulting performance and a discussion of both their advantages and disadvantages compared to other existing solutions.

1.4 Methodology

The aim of this study is to provide a practical implementation of two cryptographic primitives, an encryption scheme and a digital signature scheme. The starting point of this study will be to review similar and alternative solutions that have been developed to address the research problem in recent literature. This will be used to develop some simple metrics to measure the performance of the final implementation as well as a framework for comparison with said solutions.

A key point of the literature review is investigating ISD attacks and understanding how the proposed modifications would allow the implemented algorithms to resist such attacks. Hopefully the literature review will also provide a clear understanding of the advantages and disadvantages of other existing solutions, thus bringing the areas in need of improvement into clear focus.

Following the literature review, the study will move into implementation phase. The first clear milestone of a successful implementation is to achieve successful encryption and decryption of a single message in the case of the encryption scheme. For the digital signature scheme, obtaining a signature and successfully verifying its authenticity will be the same. The implementation will use the concrete parameter sets given in Samardjiska and Gligoroski (2016) and Gligoroski and Samardjiska (2016a).

The chosen programming language of implementation will be the C programming language. The choice of programming language is motivated by the need for an efficient implementations. While programming purely in C can significantly increase the complexity of the implementation due to lack of standard libraries and other programming idioms present in more high level languages, C offers a programming level that is lower than most other programming languages. This provides a more refined control over running complexity and allows for a wider range of optimizations. Hence, the C programming language is often used in software where efficiency is of critical importance.

Once working implementations are obtained and verified, the implementation process will continue onto optimization. Different optimizations will be added to the code wherever possible in order to bring down running time and memory requirements of the algorithms.

After the implementations are considered sufficiently optimized, we will start analyzing its performance. The implementations will be tested on a personal laptop with the following specifications:

- Intel(R) Core(TM) i7-6600U CPU @ 2.6Ghz
- 16 GB of RAM
- Operating System is Arch Linux version: 4.10.13-1-ARCH

After monitoring run times and memory consumption of several runs, we will draw several comparisons with other cryptosystems. This will be followed by final analysis of the complete solution and a brief discussion of its perceived advantages and disadvantages.

1.5 Limitations

Due to the limited capabilities of the test machine, running parameters are kept within practical limitations. Also due to time constraints, these implementations are considered far from optimal. Several known optimizations were not implemented but will be discussed at the end of this thesis. We encourage future work to expand on these optimizations and further reduce both space complexity and running time.

1.6 Outline

This thesis is split into six chapters. The first chapter provides some background information to establish the need for post quantum cryptographic primitives. The chapters will go as follows:

Chapter 2 will cover the literature review, describing the state of current popular cryptographic algorithm with the existence of quantum computers. It will also describe other attempts and alternative approaches to building post quantum cryptographic algorithms. It will also provide an in depth analysis of the proposed attacks against the previous versions of staircase generator codes cryptographic primitives.

Chapter 3 will explain the theoretical basis upon which these implementations are built. We will go through some additional background information on staircase generated codes, ISD attacks, the McEliece system and the notion behind the idea of randomly splitting the public key. Some previously mentioned concepts will also be expanded upon for further reference.

Chapter 4 will discuss the final implementations, showcasing the performance metrics and explaining the setup for the test runs. It will also provide a brief summary of optimizations both implemented and proposed for future work.

Chapter 5 will draw some comparisons of the observed performance with that of other existing solutions. It will also proved a brief discussion of the advantages and disadvantages of the final solution.

Chapter 6 is a summary of the study proceedings as well as the most prominent findings. It will also include some recommendations for future work.

Chapter 2

Literature Review

This chapter will attempt to outline the research effort that has already gone into solving the problem of maintaining cryptographic primitives in the quantum era as found in recent literature. It will describe the effects quantum computation is expected to have on classical cryptography, give an overview of different solutions that have been proposed as quantum resistant cryptographic primitives as well as lay the ground work for this thesis research questions.

2.1 Classical cryptography under quantum computation

Cryptography, specifically public key cryptography has become indispensable in the past three decades. Just as research has gone into optimizing the current cryptographic primitives and securing their implementation, it has also went into the search for a viable method of compromising their security. This has included intensive search for efficient algorithms to solve the factorization problem and the discrete logarithm problem.

These efforts have not been successful up until the point where Peter Shor published his factorization algorithm in 1994, demonstrating the possibility for factorizing a complex natural number N in polynomial time. Since Shor's algorithm conception, research into quantum computing theory has intensified greatly. While the question of when large scale quantum computers will be practically available to an attacker remains unanswered, experts have estimated the deadline to be within the next twenty years.

The looming threat of rendering our current cryptographic infrastructure obsolete has sparked a dedicated community of international researchers that endeavour to find viable solution to the problem. In Chen et al. (2016), researches from the National institute of Standards and Technology (NIST) give a report of the current status of classical cryptographic primitives (both symmetric and asymmetric) with regards to the impact of large scale quantum computation. **Table 2.1** summarizes their findings.

It can be seen from their report that main contenders in the field of public key cryptography, Elliptic curve and RSA, are both no longer secure under quantum computation as they both rely on hard mathematical problems where quantum computers have been shown

Cryptosystem	Current status
AES	Large key sizes needed
SHA-2	Larger output needed
SHA-3	Larger output needed
RSA public key encryption	Broken
Deffie-Hellman key-exchange	Broken
Elliptic curve cryptography	Broken
Buchmann-Williams key-exchange	Broken
Algebraically Homomorphic	Broken
McEliece public key encryption	Not broken yet
NTRU public key encryption	Not broken yet
Lattice-based public key encryption	Not broken yet

Table 2.1: State of classic cryptographic systems under the quantum computing.

to give exponential speed ups. Key exchange protocols such as Deffie-Hellman will also be affected and symmetric encryption systems like AES will be forced to use much larger key sizes. Even with the use of larger key sizes for symmetric encryption and larger output for hash algorithms, it is unlikely that such temporary solutions will continue to keep up with improvements in quantum computations in any practical sense.

For further motivation of this thesis, we address the concern that research into quantum resistant cryptography has started prematurely while practical large scale quantum computations remains far away. Bernstein et al. (2009) outlines several reasons why this research needs to be addressed today, rather than 15 years from now. Below is quick overview of the reasons given.

Efficiency While slow cryptographic operations might be viable in some applications, one of the main use cases of cryptography is in transferring secure digital information across the internet. This means web servers must be able to respond to thousands of clients requests each second using encrypted and signed data packets.

Elliptic curve signature algorithm has a space complexity of $O(b)$-bits for b-bits of security which is considered barely able to match the efficiency demands. If quantum resistant cryptography algorithms intend to be practical replacements to such systems, they must be able to boost similar or better efficiency standards. While research has already produced some impressive improvements in space and time constraints of quantum resistant algorithms, it remains a slow and challenging process.

Confidence McEliece hidden Goppa codes public key encryption system remains secure after thirty years of continued research into ways of compromising its security. Other quantum resistant algorithms such as lattice based cryptography and multivariate-quadratic cryptography, however, are not as old and are yet to withstand the same level of testing.

The community's choice of cryptographic system is greatly influenced by the amount of confidence garnered by that system that can only be gained over years of testing and improvements. Thus, such younger systems will remain a second preference to older more practically tested systems. The sooner new quantum resistant cryptographic standards are established, the sooner cryptoanalysts can start their search for viable attacks that may compromise the security of such systems.

Usability Taking RSA as an example, there is a clear gap between its original conception as trapdoor one way function and the complete encryption and signature system that is used today. RSA implementation does not simply calculate the cube of a message modulo n. Instead several layers of randomization, padding and techniques to handle long messages were added to the system before it could reach its current polished state.

This infrastructure was developed gradually over many years, and many initial proposals turned out to be a disaster (PKCS#1 v1$\dot{5}$ padding scheme). Furthermore once correctness of the infrastructure is established, many software and hardware implementations are needed, each customized to a different type of applications. Many of such implementations expose their own vulnerabilities and new vectors of attacks.

In conclusion, the process of moving the current secure applications from using classical cryptography to post quantum cryptography is a time consuming process that can not be delayed until the point when it is actually needed. Time must be allowed for many revisions, mistakes and general improvement.

2.2 Post quantum cryptographic paradigms

Fortunately quantum resistant cryptographic systems do not need to be built from scratch. Many existing cryptographic systems are already believed to resist quantum attacks, as they rely on a variety of hard problems for their security other than mathematical problems relied on by RSA and Elliptic Curve Cryptography. Some of this systems have existed for over thirty years (the McEliece cryptosystem was proposed in 1978) while others are relatively new.

2.2.1 Hash-based Cryptography

Hash functions can be used for digital signature schemes. The only requirement is a standard cryptographic hash function that produces $2b$ bits of information. SHA-256 have been used for cases of $b = 128$, although many concerns have been raised in recent years over its security. While all known classic attacks remain extremely expensive, SHA-256 is unlikely to remain secure in the quantum era.

An example of a quantum-resistant hash based digital signature scheme is the Lamport-Diffie One-Time Signature, published in 1979. As the name suggests the scheme is designed for cases where only one signature is needed, however, it can be adapted for use with multiple messages using a technique called "chaining".

In this scheme, the public key and private key each have $8b^2$ bits: i.e 16 kilobytes for $b = 128$. The size of a signature produced by this scheme is $2b(2b + 1)$ bits. To guarantee the security of the algorithm the signer must use the key pair for signing a message only once. For signing multiple messages, the signer can include a new public key in the contents of the first message, after which the corresponding new private key can be used to sign a second message.

That means the signature of the n^{th} message will contain all $n-1$ previously signed messages, causing the size of the signature to grow linearly with the size of the chain. A more advanced system such as Merkel's hash tree signature system offers logarithmic growth rate relative to the number of messages signed. For detailed information on the working of both systems, the reader is referred to Buchmann and Ding (2008).

2.2.2 Lattice-based Cryptography

Lattice based cryptographic systems rely on lattice related hard problems. In cryptography, a lattice is defined as an abelian subgroup of R^n where n is greater than 0. There are different lattice based computational problems that can be used for cryptographic systems such as:

- shortest vector problem.
- closest vector problem.
- shortest independent vector problem.

The shortest vector problem, being the most commonly used problem, is computationally hard as it tries to approximate the minimum euclidean distance of a non-zero lattice vector. (Buchmann and Ding, 2008)

2.2.3 Multivariate Cryptography

In multivariate cryptography, public keys are formed of a sequence of $2b$ polynomials using $4b$ variables that have binary coefficients $\in F_2 = \{0, 1\}$. There can be no squared terms, i.e each polynomial can at most be of the 2^{nd} degree. The public key can be represented as a sequence of bits $1 + 4b + 4b(4b - 1)$. Calculating the final size of the public key, it comes out to $16b^3 + 4b^2 + 2b$. For $b = 128$, the public key is 16 megabytes.

The signature of a message m has a size of $6b$ bits, consisting of $4b$ binary values $\in F_2$ and a $2b$-bits string r. For a multivariate digital signature system S with polynomials $P_1, P_2, ...P_{2b}$ and polynomial variables $w_1, w_2, ..., w_{4b}$, r would satisfy the formula:

$$H(r, m) = (P_1(w_1, w_2, \ldots, w_{4b}), \ldots, P_{2b}(w_1, w_2, \ldots, w_{4b})) \tag{2.1}$$

where H is a standard hash function.

Verifying the signature would involve calculating the values of the polynomials using the given values for the variables as well as calculating the hash function. That gives a total of $3b + n$ operations where n is the cost of evaluating the hash function. Multivariate cryptographic systems hold the advantage of giving very small signature sizes, with more modern systems reducing the size even further. Some systems even give the option of shorter public keys.

The security of multivariate cryptosystems depends the hardness of finding a sequence of bits $w_1, w_2, \ldots, w_{4b}$ that can satisfy the above equation i.e produce the desired $2b$ output $H(r, m)$. On average, a brute force attack would need 2^{2b} operations. Advanced equation solving attacks can reduce the number of operations, but no known attack can practically be expected to succeed in 2^b operations or less.

For a legitimate user to verify a signature, the signer generates a public key containing the polynomials $P_1, P_2, \dots, P_{2b}$ with a secret structure known as Hidden Field Equations $"HFE^{v-}"$. It is theoretically possible for an attacker to find the polynomials hidden in the public key using several legitimate signatures but no such attacks have been found. For futher discussion of multivariate cryptosystems key generation and signature procedures, the reader is refered to Bernstein et al. (2009)

2.2.4 Code-based Cryptography

Most interesting to the subject of this thesis, is code based cryptography. Code based cryptosystems use coding theory and error correcting codes for encryption and digital signature schemes. A classic code based cryptosystem is defined by parameters: n, d and t. Assuming b is a power of 2, we define those parameters as $n = 4b\ logb$, $d = \lceil logn \rceil$ and $t = \lfloor 0.5n/d \rfloor$.

The public key is a generator matrix G with dimensions $dt \times n$ with binary coefficients $\in F_2$. The input message is a bit string with length n and hamming weight t. The encryption procedure of a code based cryptosystem is simple multiplication of the input vector and the generator matrix. The cipher text is a bit vector of size dt.

The process of reversing the multiplication is known as syndrome decoding, which is the problem presented to an attacker of the system. While it is relatively easy to find an n bit vector v that satisfies the equation:

$$Gv = Gm \tag{2.2}$$

the number of possibilities for v is too large. The known solutions to such problem have exponential performance at best.

In order for the receiver to decrypt the message, there needs to be a matrix with secret structure known as hidden Goppa codes that allows the use of decoding algorithms such as unique decoding and list decoding.

As the basis for our new cryptosystem, we will cover in detail the workings of the McEliece system, which is considered the first code based cryptosystem proposed, as well as the Niederriter variant, as the more commonly used version of it.

It is worth noting that the cryptographic paradigms described so far can offer several advantages over the commonly used RSA cryptosystem, aside from them being quantum resistant. Code based cryptosystems have extremely efficient encryption, decryption and key generation procedures and multivariate cryptosystem produce signatures of very small space complexity.

However, these systems have historically been outdone by the RSA cryptosystems and similar classical cryptographic systems mainly because of their much larger key sizes requirements. Now that recommendations for RSA key size continue to grow consistently in order to maintain the same level of security with faster machines, and as we approach the quantum era, these considerations are likely to soon become inconsequential.

2.3 The McEliece Cryptosystem

The McEliece cryptosystem was proposed in 1978 as the first code based cryptographic system. During the key generation procedure a generator matrix G is chosen from the key space of Gobba codes, a random invertible binary matrix S dimensions $k \times k$ and permutation matrix P with dimensions $n \times n$ [1].

The three matrices are used as the private key and kept secret. The public key is a matrix G' = SGP. The sender takes an k bit vector as the message m and calculates the cipher text C as $C = mG' + e$ where e is a random n bit with hamming weight t, known as the error vector.

For decryption, the receiver calculates P^{-1} and S^{-1}. In order to retrieve mS, the receiver first calculates CP^{-1} giving $mSG + eP^{-1}$ and then applies one of several syndrome decoding algorithms that calculates $x = mS$. Finally, to get the original message m, x is simply multiplied by S^{-1}.

The original McEliece system and its subsequent variations exposed two types of attack vectors:

Structural Attack Structural attacks aim at reconstructing the decoder matrix G given the knowledge of the cipher text C and the encrypting public key matrix G'. If the attack is successful, the private key G is revealed and the attacker becomes able to decrypt the message.

Decoding Attack A decoding attack aims at decrypting a single cipher text to retrieve the original plain text. While this attack would compromise a single message, the secrets of the cryptosystem remains intact. (Au et al., 2003)

2.4 St-Gen Code Based Cryptography

In Gligoroski et al. (2014), the original version of the cryptosystem presented in thesis, is proposed. The proposal starts with introducing a novelty family of codes that can decode a given error set with overwhelming probability. The codes are named staircase generator codes in reference to the special structure of the generator matrix, which can be described as a step wise random block structure. Like the McEliece system, this cryptosystem relies on the hardness of the decoding problem and the hardness of recovering the original structure of the code for its security.

Another novelty approach proposed in the paper, is imposing a specific structure on the error vectors. In the McEliece system, and other code based cryptosystems, the error vectors have traditionally been completely random vectors with the only restriction of having a hamming weight of t, where t is lower bound of the error correcting capability of the generator matrix G. This goes back to coding theory where error vectors are used to model transmission errors occurring due to a noisy channel.

[1] In some literature the notation of k and n are interchanged, making n refer to the size of the input vector and k refer to the size of the output cipher text. This is inconsequential as the only restriction on k and n is that the size of the cipher text is larger than the input vector i.e in our case $k > n$. This is the notation that will be used throughout this thesis.

While a random error vector is a proper model in such context, it does not offer any advantages in the context of code based cryptosystems. Thus, the paper proposes replacing the noisy channel model with artificial noise where the sender has full control over the level and structure of the noise. This translates into a set of accepted error vectors known as error sets, all sharing the same two characteristics: granulation and density (instead of hamming weight as in the case of the classic McEliece system).

Finally the proposal outlines an encryption and a signature scheme that follow the same basic structure of the McEliece system. It should be noted the the specific structure of the generator matrix make it more susceptible to structural attacks as pointed out by the proposal. However, a clear advantage of this cryptosystem is the natural progression from an encryption scheme to a digital signature scheme. While other code based cryptosystems required specific tweaking to the decoding algorithm in order to maximize the probability of finding a decodable syndrome, this cryptosystem makes use of the decoding algorithm directly with high probability of success.

2.5 Information Set Decoding attack

Information Set Decoding was proposed by Prange in Prange (1962), followed by later optimizations. It is essentially a set of techniques used to compromise the security of code based cryptosystems. It can be used to recover error vectors or find the hidden structure of the generator matrix. It relies in both cases on finding low weight code words.

In Moody and Perlner (2016), two valid attacks are demonstrated, one for the recovery of the error vector and the other for the private key. As these attacks are the main motivation for the modifications done in this thesis, we give a detailed overview of the error vector attack below:

Let C be a cipher text, where $C = mG' + e$. An ISD attack can be used to recover both e and m given C. The basic approach of the attack is to guess k bits of the error vector and calculating the rest of vector using linear algebra. As the probability of guessing k bits of the error vector correctly increase, the number of iterations needed for the attack decreases. Algorithm 1 gives a simple outline for an ISD attack, taken from Moody and Perlner (2016).

Moody and Perlner (2016) provides a valid approach to maximizing the probability of guessing k bits. Let e be the error vector form of consecutive 2 bit blocks drawn randomly from the set $E = 00, 01, 10$. It can be seen, that within each block, a single bit has a probability $\frac{2}{3}$ of being 0. This means if the attack chooses to guess a single bit within each 2 bit block, the probability of success is $(\frac{2}{3})^k$. This greatly undermines the 80-bit security expected from the original version of the algorithm.

For a detailed discussion of this attack as well as the private key ISD attack, the reader is referred to Moody and Perlner (2016).

2.6 Random split of St-Gen Codes

In order to address the security vulnerabilities outlined by Moody and Perlner (2016), a modification to the staircase generator codes cryptosystem is proposed. A key point of

Algorithm 1: ISD attack for the Error Vector

```
Input  : cipher text $c$, and a parameter $k$
Output: message $m$, error $e$
  1. Choose random permutation matrix $P'$, and use it to permute cipher text $C$.
```

$$\begin{aligned} c' &= (mG' + e)P' \\ &= mG'P' + eP' \\ &= m(A|B) + (e'_1|e'_2) \\ &= (mA + e'_1)|(mB + e'_2) \end{aligned} \tag{2.3}$$

```
     where $A$ and $e'_1$ are the first $k$ columns of the matrix $G'P'$ and the error vector $eP'$
     respectively.
  2. If $A$ is not invertible go to step 1.
  3. Guess $e'_1$. If correct the message can be reconstructed as:
```

$$m = ((mA + e'_1) - e'_1)A^{-1} \tag{2.4}$$

```
     the error vector can also be recovered as $e = c - mG'$.
  4. If the error vector belongs to the error set ( which is publicly known ), return $m$ and
     $e$. Otherwise go back to step1.
```

the ISD attacks is the exposure of the very specific structure of the generator matrix in the public key. To thwart such attack, the new proposal suggests splitting the public key into several random matrices. This introduces a new parameter s to the cryptosystem, where s is the number of splits.

This proposal is then translated into two new schemes for encryption and digital signatures. The purpose of this thesis is to implement these two schemes in C and provide an analysis of their performance. The next chapter will provide a detailed, ground up explanation of the theoretical background behind this modification. (Gligoroski and Samardjiska, 2016a; Samardjiska and Gligoroski, 2016)

Chapter 3

Basic Theory

This chapter will give a ground up construction of the background theory and workings of the cryptosystem implemented during this thesis. We will start with an explanation of code based cryptography and the way it draws on the practicalities of coding theory. Following that, will give an overview of the workings of the McEliece system as the basis of the current system. Finally, this chapter will conclude with the detailed workings of the staircase generator codes cryptosystem with random splitting of the public key.

3.1 Coding Theory

Coding theory goes back to the principles introduced by claude shannon in 1948. The original purpose of coding theory was not cryptography, but integrity. When a message is sent through a communication channel, transmission errors may occur due to noise that cause the message to be distorted. A binary message for example can have some of its bits flipped. Coding theory is a collection of techniques and methods of attaching extra information to the message that allows the receiver to recover the original message free of errors. (Buchmann and Ding, 2008)

A simple example of using coding theory for error recovery is the triple redundancy method, where three copies of the message are sent instead of one. At the message destination, the receiver used a majority vote method to determine the contents of the original message. This method may be effective but it is very inefficient. Coding theory revolves around finding optimized methods that can allow the receiver to recover from as many transmission errors as possible while keeping message redundancy and overhead to a minimum.

3.1.1 Linear Binary Code

Let V be the set of all n length vectors of size k defined over the elements of $\mathbb{F}_2$. We can define a linear binary code of dimension n as a subspace of $V \cong \mathbb{F}_2^n$. This binary linear code is defined by its parameters n and k, written as (n, k), and called the length

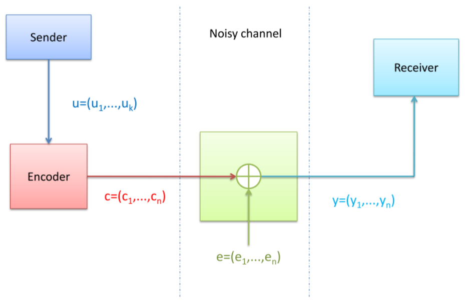

Figure 3.1: Encoding and decoding a message.

and dimension of the code. A codeword is any vector that happens to occur in the code. The code is considered linear since any linear combination of two codewords will itself be a codeword.

Studying some interesting properties of codewords, we start with the weight of a codeword. The weight of a codeword in linear binary code is the number of non zero elements in the codeword i.e the weight of the codeword 0111 is 3. This gives rise to a property known as the minimum weight d of the code, which is the smallest weight of any non zero codeword i.e excluding the all zero codeword.

The minimum weight of a code is not always trivial to find, however, when known it is used as a third parameter of the code, written as (n, k, d). This gives a direct formula to the number of errors the code can correct. An (n, k, d) can correct up to t errors, where $t = \lfloor \frac{d-1}{2} \rfloor$. This result is apparent from figure 3.2.

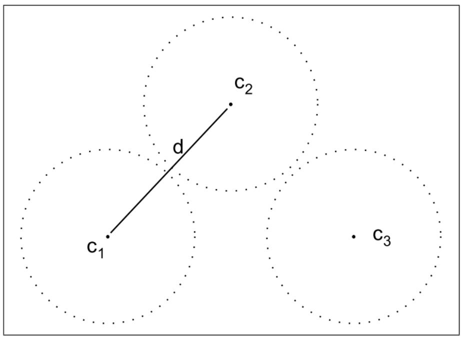

Figure 3.2: The distance between two codewords define the correcting capability of the code.

While d defines the effectiveness of a code, in how many errors it can correct, the ration between n and k define the efficiency of the code. If n is large compared to k, then the code sends too many extra bits with each message, and so not very efficient. In our previous code example of triple duplication would have $n = 3k$, which gives a ratio of 3, illustrating the fact that for each bit of the original message we send 3 bits through the communication channel. A clearly very inefficient transmission rate.

In order to generate the codewords of a specific code, we use a generator matrix. For a (n, k) code, the generator matrix is formed of k linearly independent vectors with entries in BF. By taking different linear combinations of the vectors in the generator matrix, we can generate all codewords in the code.

3.1.2 The Hamming Code

The Hamming code is an example of a linear binary code used to describe general encoding and decoding procedures. The Hamming code is defined as a $(7, 4, 3)$ code. The generator matrix of the Hamming code is:

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

Using the parameters of the code, we can see that an input vector to this code would be 4 bits and the output vector will have length of 7 bits. The encoding process uses the equation:

$$c = vG \tag{3.1}$$

where c is the codeword, v is input vector and G is the above generator matrix. An example input would be the vector 1100 which would give the codeword 0011001.(Au et al., 2003)

3.1.3 The Decoding Problem

Once a message is encoded using the above mentioned technique and sent, it must be decoded on the receiver end to recover the original message. Continuing the example of the hamming code, we know that it can only recover messages with only 1 transmission error, as given by the equation

$$d = \frac{3-1}{2} = 1 \tag{3.2}$$

This means that any message being decoded must be of the form: $c = m+e$, where e is the error vector of hamming weight 1. A simple decoding method would be to construct a look up table that matches all possible error vectors with all possible input vectors giving all possible decodable messages. For the hamming Code the table would contain 128 codewords, which is already quite large. For a proper code with larger parameters, this method would be completely impractical.

Coding theory offers a more practical method of decoding, known as syndrome decoding. First, we define a parity check matrix H for an (n, k) code C, which is a $(n-k) \times n$ matrix such that the dot product of any row of the matrix with any codeword in C is 0.

The parity check matrix can be used to calculate the syndrome of a vector c, where c is the received message in need of decoding, as given by the formula:

$$Syndrome(c) = H.c^t \tag{3.3}$$

It can be shown that for encoded message c and transmission error e, the syndrome of the message is the same as the syndrom of the error vector e, as given by the equation:

$$H.\bar{c}^t = H.(c+e)^t = H.c^t + H.e^t = 0 + H.e^t = H.e^t \tag{3.4}$$

Hence, by using the party check matrix we can find the syndrome of the received message which lets us identify the error vector of the message. Other decoding algorithms exist in coding theory, all based on the idea of syndrome decoding. (Au et al., 2003)

3.1.4 Goppa Codes

The McEiece cryptosystems, as well as the system implemented rely on a family of codes known as Goppa codes. These are codes of fixed length and dimension that exhibit interesting properties. While we will not go into the details of Goppa codes, we will argue the case of why they are used. A Gobba code is generated though an irreducible polynomial of degree t, where t defines the lower bound of the error correcting capability of the code.

Goppa codes give the advantage of a fast polynomial time decoding algorithm. Generator matrices of Goppa codes are nearly random, making them very hard to find. There are also many Goppa codes, for any fixed length n and their number increases exponentially with the length of the code and with the degree of the polynomial. (Au et al., 2003)

3.1.5 Hard Problems

Binary Syndrome Decoding problem Given an $r \times n$ parity check matrix H over F_2, a target binary vector s and an integer $t \geq 0$, find a binary x where $s = Hx^T$.

This problem has been shown to be NP-complete. There exists a q-ary version of this problem defined over F_q, that is also NP-complete, though it is of little importance to code based cryptosystems.

Goppa Code Distinguishing problem Given a random (n, k) Gobba-code C, find an $(n-k) \times n$ binary matrix H, where H is the parity check matrix of C. This problem has also been proven to be NB-complete.

This two hard problems form the basis of security for the McEliece system and other code based cryptosystems.

3.2 Code Based Cryptosystem

Though not its original purpose, the techniques of coding theory and error correcting codes lend themselves easily to cryptography. Taking the plain text to be the original message, the encryption process would first be encoding the message with a generator matrix. A random error vector would then be added to the resulting vector as artificial noise. This effectively hides the original contents of the message giving the cipher text.

For the decryption process, the receiver would have the generator matrix as part of the private key. The receiver would use a decoding algorithm to get the original message. Only by having the private key, the generator matrix, can the receiver have enough knowledge of the code that allows decoding the message. This concept is the basis on which the McEliece cryptosystem and other code based cryptosystems, including the ones discussed in following sections, are built.

3.3 The McEliece cryptosystem

The McEliece system is considered one of the first code based cryptosystems introduced. It was proposed by McEliece in 1978, followed by several modifications and optimizations. The most commonly known version of the McEliece system today is the Niederreiter variant in 1986. The McEliece system is one of the major candidates for post quantum cryptography research as it remains unbroken after nearly thirty years of cryptoanalysis

3.3.1 Key Generation Procedure

Algorithm 2 outlines the key generation procedure of the McEliece system. Initial parameters n, k and t are chosen according to the security parameter required. (Au et al., 2003)

Algorithm 2: McEliece Key Generation Procedure

```
1. Select parameters $n, k$ and $t$ to form the length, dimension and error correcting
   capability of the Goppa code to be used as the key.
2. Generate a random generator matrix $G$ of dimension $k \times n$, capable of generating
   an $(n, k, 2n + t)$ linear binary code.
3. Select a random $n \times n$ permutation matrix $P$ over $\mathbb{F}_2$.
4. Select a random invertible $k \times k$ matrix $S$ over $\mathbb{F}_2$.
5. Compute $G_{pub} = SGP$.
```

The public key of the algorithm is formed of (G_{pub}, t) and the private key of the algorithm is given by (S, G, P).

3.3.2 The Encryption Procedure

Algorithm 3 shows the procedure of encrypting message $m \in \mathbb{F}_2^k$ using the public key.

3.3.3 The Decryption Procedure

Algorithm 4 shows decryption procedure of the McEliece cryptosystem, using the private key.

```
Algorithm 3: McEliece Encryption Procedure
  Input   : message $m$, public key $(G_{pub}, t)$
  Output: cipher text $c$
    1. Randomly generate an error vector $e$ of length $k$ and weight $t$.
    2. Calculate $c = mG_{pub} + e$.
```

```
Algorithm 4: McEliece Decryption Procedure
  Input   : cipher text $c$, private key $(S, G, P)$
  Output: original plain text $m$
    1. Compute $cP^{-1}$ according to the following equation.
           $cP^{-1} = (mG' + e)P^{-1} = mSG + eP^{-1}$      (3.5)
       Since $P$ is a permutation matrix, It is clear that the hamming weight of $eP^{-1}$ is
       equal to the hamming weight of $e$.
    2. Use the decoding alogrithm to decode $cP^{-1}$, getting $y = mS$.
    3. compute $m = yS^{-1}$
```

3.3.4 Analysis of the McEliece Cryptosystem

The first clear advantage of the McEliece cryptosystem is the efficiency of the encryption and decryption procedures. Both procedures rely solely on matrix multiplication which is much faster than exponentiation required by the RSA system.

However, the McEliece cryptosystem, as well as all code based cryptosystems, suffers from the disadvantage of large key sizes due to the requirement of storing the generator matrix, which can get very large as the security parameter increases.

Table 3.1 relates the sizes of the public keys to generator matrix sizes and security parameters. For comparison, the current recommendation for the RSA key size is 2048-bits. It should also be noted that the McEliece cryptosystem does not offer a digital signature scheme. (Au et al., 2003)

Table 3.1: Public key sizes for the McEliece Cryptosystem

Code	**security**	**size of public key in bits**
(1024, 524, 2 50+1)	2^{64}	535,576
(2048, 1025, 2 93+1)	2^{106}	2,099,200
(4096, 2056, 2 170+1)	2^{136}	8,421,376

3.4 The Niederreiter Cryptosystem

A variation of the McEliece cryptosystem that was introduced in 1986. The key difference between the Niederreiter scheme and the original McEliece proposal is that the Niederrreiter scheme uses the party check matrix while the McEliece scheme uses the generator matrix. (Buchmann and Ding, 2008)

3.4.1 Key Generation Procedure

Algorithm 5: Niederreiter Key Generation Procedure

```
1. Select parameters $n, k$ and $t$ to form the length, dimension and error correcting
   capability of the Goppa code to be used as the key.
2. Generate a random parity check matrix $H$ of dimension $n - k \times n$, capable of
   generating an $(n, k, 2n + t)$ linear binary code.
3. Select a random $n \times n$ permutation matrix $P$ over $\mathbb{F}_2$.
4. Select a random invertible $n - k \times n - k$ matrix $S$ over $\mathbb{F}_2$.
5. Compute $H_{pub} = SHP$.
```

The public key of the algorithm is formed of (H_{pub}, t) and the private key of the algorithm is given by (S, H, P).

3.4.2 The Encryption Procedure

Algorithm 6 shows the procedure of encrypting message $m \in \mathbb{F}_2^k$ using the public key.

Algorithm 6: Niederreiter Encryption Procedure

```
Input  : message $m$, public key $(H_{pub}, t)$
Output: cipher text $c$
    1. Calculate $c = Hm^T$.
```

3.4.3 The Decryption Procedure

Algorithm 7 shows decryption procedure of the Niederreiter cryptosystem, using the private key.

```
Algorithm 7: Niederreiter Decryption Procedure
  Input  : cipher text $c$, private key $(S, G, P)$
  Output: original plain text $m$
    1. Compute $S^{-1}c$ according to the following equation.
         $S^{-1}c = S^{-1}SHPm^T = HPm^T$    (3.6)
       Since $P$ is a permutation matrix, It is clear that the hamming weight of $Pm^T$ is
       equal to the hamming weight of $m$.
    2. Use the decoding algorithm to decode $HPm^t$, getting $y = Pm^T$.
    3. compute $m^T = P^{-1}y$
```

3.4.4 Analysis of the Niederreiter Cryptosystem

The Niederreiter scheme offers several improvements over the McEliece scheme. Mainly the smaller matrix dimensions of H compared to G, which allows for a near 10 times faster encryption procedure. It also reduces the sizes of the public key and the private key, which is key goal for any code based cryptosystem as it remains one of the major drawbacks. However, the main advantage of the Niederrieter cryptosystem is it can be translated into a digital signature scheme as shown by Courtois, Finiasz and Sendrier. (Buchmann and Ding, 2008)

3.4.5 The CFS Digital Signature Scheme

The CFS signature scheme is built on the same basics of the Niederreiter encryption scheme using the party check matrix. Given a message m, a hash function h with outputs of length $n - k$ and a syndrom decoding algorithm, we do the following.

1. Set initial value of i to 0.
2. Find $s \in \mathbb{F}_2^n$ of given weight t such that $h(h(m)||i) = Hs^T$ where H is the parity check matrix. This can be done by applying the decoding algorithm to $h(h(m)||i)$
3. If $h(h(m)||i)$ is not decodable, increase i and try again.

The signature of the message is given by $(s||i)$. Figure 3.3 shows the procedure of incrementing a counter until a decodable signature is found.

Goppa codes are very useful for this kind of scheme as they are a dense family codes, increasing the probability of finding a decodable signature. The CFS scheme gives signatures of length 131 bits with signature cost of 12×10^{10} operations. The verification cost of the scheme is 1296 operations and the size of the public key 9 Mbits for parameters that offer 80-bits of security against ISD attacks.

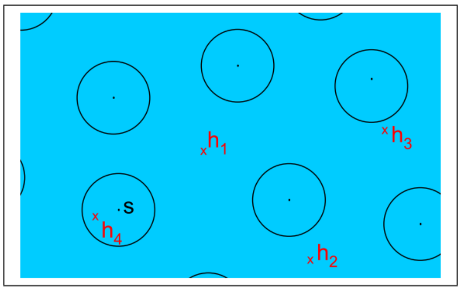

Figure 3.3: CFS signature procedure. (Buchmann and Ding, 2008)

It can be seen from these parameters that the CFS is a very expensive scheme. Many codewords need to be decoded before a valid signature can be found, as well as, the usual concern over the size of the public key. (Courtois et al., 2001)

3.5 The Staircase Generator Codes Cryptosystem

Based on the McEliece system, Gligoroski et al. (2014) introduces a code based cryptosystem called staircase generator codes cryptosystem. This system includes both an encryption and a digital signature scheme. It imposes new restriction on the code generator matrix and the error vectors. It also replaces the classically used unique syndrome decoding algorithm with a list decoding algorithm.

3.5.1 Error Sets

In traditional coding theory, and in previous code based cryptosystems, the error vector is a random vector of hamming weight less than t. The restriction of the hamming weight guarantees the ability of the generator matrix to recover the original message after adding the error vector to it.

This classic approach has been chosen with the intention of modeling transmission errors that occur when the message passes through a noisy channel. This, however, offers no real advantage when dealing with cryptographic systems. Error vectors added to plain text messages in cryptographic systems simulate artificial noise, with the purpose of hiding the original contents of the message.

Using the hamming weight of the error vector as the only restriction, we get an error set that partially covers a hamming sphere surrounding each codeword. In this case, the codeword c can be recovered from the hidden cipher text $\bar{c} = c + e$, where e is the error vector, as long as the hamming distance between c and $\bar{c}$ do not exceed the minimum hamming distance d of the code. If d is exceeded, then $\bar{c}$ lies in the area where its no longer possible to uniquely decide on the original codeword. Figures 3.4 and 3.5 visualize this idea.

In contrast, to define the error set for this cryptosystem first we define a positive integer l known as the granulation of the error set. Then we define S as a set of multivariate

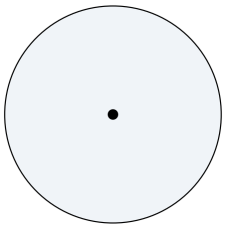

Figure 3.4: Classic error sets represented by a hamming sphere around a codeword.

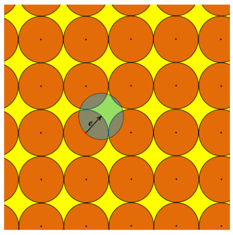

Figure 3.5: Limitations of unique decoding for classical error sets.

polynomials defined over $\mathbb{F}_2$. E_l is an error set if it is the kernel of S as given in the following definition:

$$E_l = Ker(S) = e \in \mathbb{F}_2^l | f(e) = 0, \forall f \in S \tag{3.7}$$

Another property of the error set is its density ρ, where $\rho = |E_l|^{1/2}$. Having defined an error set E_l, we can find codes that can correct error vectors drawn from $E_n = {}^{m}_{l} = E_l \times E_l \times \cdots \times E_l$. This allows for larger error sets unrestricted by the hamming metric. Figures 3.7 and 3.6 highlight the difference between this approach to error vectors and the classic approach.

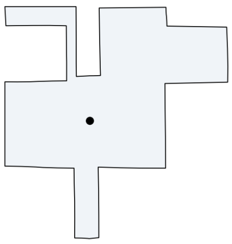

Figure 3.6: Arbitary error set surrounding a codeword.

As a concrete example, we can take the set $E_2^2 = 00, 01, 10$ to be an error set, where error vectors would be constructed form elements randomly chosen from E. That is, the error vector e of length n would be equal to $e_1 || e_2 || \dots || e_m$, where $e_i \in E$, $m = \frac{n}{l}$ and $l = 2$

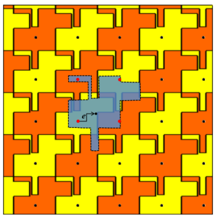

Figure 3.7: Arbitary error sets does not allow unqiue decoding. However, list decoding succeeds with overwhelming probability.

3.5.2 Staircase Generator Codes

Having defined the notion of an error set, proposes a new family of codes that can correct such errors with overwhelming probability. These codes are called staircase generator codes, due to the specific random stepwise block structure of the generator matrix.

Let C be a binary code with length n, dimension k and generator matrix G. Figure 3.8 show an example of the matrix G. The matrix G is formed of several random binary matrices, each matrix B_i is of the dimension $\sum_{j=1}^{i} K_j \times n_i$, where $k = k_1 + k_2 + \cdots + k_w$ and $n = k + n_1 + n_2 + \cdots + n_w$.

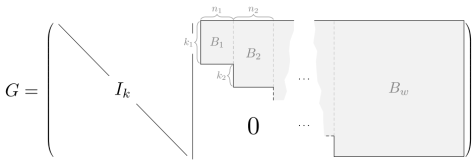

Figure 3.8: Generator matrix for (n, k) binary code with stepwise block structure.

A big advantage of using staircase generator codes is that it allows using an efficient list decoding algorithm. Unlike the classic syndrome decoding algorithms that output only one unique decoding of the codeword, list decoding can output a list of possible decoding. This means it can correct a larger number of errors. A general list decoding algorithm is given by algorithm 8, for (n, k) code C with error set E_l, where l divides n and $m = \frac{n}{l}$

3.5.3 Key Generation Procedure

The staircase generator codes cryptosystem follow a key generation procedure very similar to that of the McEliece cryptosystem. Algorithm 9 outlines this procedure.

The public key of the algorithm is formed of $G_{pub} = SGP$ and the private key of the algorithm is given by (S, G, P).

3.5.4 The Encryption Scheme

Algorithm 10 and 11 shows the procedure of encrypting and decrypting a message $m \in \mathbb{F}_2^k$ using the generated keys. Both procedures are identical to the McEliece procedures, with

```
Algorithm 8: General List Decoding
  Input        : vector $y \in \mathbb{F}_2^n$, generator matrix $G$
  Output       : List $L_w \subset \mathbb{F}_2^k$ of valid decodings of $y$
  Preliminaries: Let $K_i = K_1 + \dots + K_i$. Let decoding $x \in \mathbb{F}_2^k = x_1||x_2||\dots||x_w$
                 where the length of $x_i$ is $k_i$. The same is done for $y \in \mathbb{F}_2^n$, hence
                 $y = y_0||y_1||y_2||\dots||y_w$, such that the length of $y_i$ is $n_i$ and
                 $|y_0| = k$. Finally, $y_0$ is defined as $y_0 = y_0[1]||y_0[2]||\dots||y_0[w]$
                 where the length of $y_0[i]$ is $k_i$.
    1. Start with list $T_0 = L_0$ with all possible decodings of $y_0[1]$.
           $T_0 \leftarrow \{x' = y_0[1] + e | e \in E^{k_1/l}\}$    (3.8)
    2. While $i \leq w$, For each $x' \in T_{i-1}$, add to $L_i$ all new candidates where
       $x'B_i + y_i \in E^{n_i/l}$.
           $L_i \leftarrow \{x' \in T_{i-1} | x'B_i + y_i \in E^{n_i/l}\}$    (3.9)
    3. If $i < w$ build $T_i$ from $L_i$ using the formula
           $T_i \leftarrow \{x'||(y_0[i+1] + e) | x' \in L_i, e \in E^{k_{i+1}/l}\}$    (3.10)
    4. Return $L_w$.
```

the difference being in the construction of the keys and the error vectors. The error vector e is randomly from the error set $E_l^m = E_l \times E_l \times \dots \times E_l$

3.5.5 The Digital Signatures Scheme

The staircase generator codes encryption scheme holds the advantage of being directly translatable to a digital signature scheme, unlike the McEliece encryption scheme. It also does not require the counter and repeated decoding modifications to the Niederreiter, as required by the CFS digital.

The St-Gen codes signature scheme introduces a new decoding algorithm. While algorithm 8 is valid for signing messages, it generates all decodings of a syndrome, while a signature scheme only needs to generate one.

The reader is referred to Gligoroski et al. (2014) for a formal proof that algorithm 12 will find a signature with probability $> 1/2$. Algorithms 13 and 14 show the full signature and verification procedure.

3.5.6 Analysis Of St-Gen Codes Cryptosystem

The St-Gen codes cryptosystem offers new encryption and digital signature schemes. The encryption scheme is directly translatable to a digital signature scheme without the need

Algorithm 9: St-Gen Key Generation Procedure

1. Generate a random generator matrix G of dimension $k \times n$ with the form given in figure 3.8, capable of generating an (n, k) linear binary code.
2. Select a random $n \times n$ permutation matrix P over $\mathbb{F}_2$. P should only permute the m substrings of input vector y and is selected as follows:
   Select permutation π on $\{1, 2, \ldots, m\}$ and let P be any permutation matrix induced by π such that:
   $$\begin{aligned} y &= y_1||y_2||\cdots||y_m \\ yP &= y_{\pi(1)}||y_{\pi(2)}||\cdots||y_{\pi(m)} \end{aligned} \tag{3.11}$$
3. Select a random invertible $k \times k$ matrix S over $\mathbb{F}_2$.
4. Compute $G_{pub} = SGP$.

Algorithm 10: St-Gen Encryption Procedure

Input : message m, public key G_{pub}, error vector $e \in E_l^m$
Output: cipher text c

1. Calculate $c = mG_{pub} + e$.

for added counters, unlike the CFS scheme, or any similar modifications. The step wise block structure of the generator matrix allows defining a very efficient list decoding algorithm that can correct the errors with overwhelming probability. It follows the principles of code based cryptosystems which are one of the main candidates for post quantum cryptosystems.

The cryptosystem relies for its security on the same principles relied on by the McEliece and the Niederreiter cryptosystems. However the specific structure of the generator matrix of St-Gen codes makes it more vulnerable to ISD attacks and distinguisher attacks.

In Moody and Perlner (2016), a practical ISD attack is outlined that allows for full recovery of the private key. The attack shows that using the suggested parameters for 80 bits of security, the system can be broken on average in less than two hours. An ISD attack essentially translates to the following:

Find an information set $\mathbb{I}$ that is an invertible sub-matrix of the generator matrix G. This gives an error vector $e_{\mathbb{I}}$ with a specific pattern with respect to $\mathbb{I}$. This allows partial decryption of cipher text using the formula:

$$m = (c_{\mathbb{I}} + e_{\mathbb{I}})G_{\mathbb{I}}^{-1} \tag{3.13}$$

A practical example of an ISD attack against this cryptosystem would go as follows. Let $x \in \mathbb{F}_2^k$, $y' = xSG$ where G is the generator matrix of an St-Gen code. This means both y' and $y'P$ are codewords generated by G, and since P is a permutation matrix, both y'

Algorithm 11: St-Gen Decryption Procedure

```
Input  : cipher text $c$, private key $(S, G, P)$
Output: original plain text $m$
  1. Compute $cP^{-1}$ according to the following equation.
```

$$\begin{aligned} cP^{-1} &= (mG' + e)P^{-1} \\ &= mSG + eP^{-1} \end{aligned} \tag{3.12}$$

```
  2. Use the decoding algorithm 8 to decode $cP^{-1}$, getting $y = mS$.
  3. Compute $m = yS^{-1}$
```

Algorithm 12: Signature List Decoding

```
Input          : vector $y \in \mathbb{F}_2^n$, generator matrix $G$
Output         : valid decoding $s \in \mathbb{F}_2^k$
Preliminaries: Using same notation from algorithm 8 and adding variables
                 $ExpLimit_i \leq r^{n_i}$.
  1. While $1 \geq i < w$, find $x' \in \mathbb{F}_2^{k_i}$ such that $x'B_i + y_i \in E^{n_i/l}$ trying at most
     $ExpLimit_i$.
  2. Expand x' into at most $ExpLimit_{i+1}$ by appending random errors from $E^{n_{i+1}/l}$
     with the sum of $y_0[i + 1]$.
  3. Increment $i$ and go back to step 1.
  4. If $i = w - 1$ switch to the list decoding alogrithm 8.
  5. Return $s \leftarrow L_w$.
```

and $y'P$ have the same hamming weight.

Due to the structure of the generator matrix, there exists a supspace of dimension k_w with support of size $(k_w + n_w)$. Finding enough codewords with hamming weight $(k_w + n_w)/2$ allows the attacker to find this support, partially revealing the permutation matrix P, which in turn reveals part of the structure of G. This attack can be done repeatedly until the entirety of generator matrix G has been found.(Samardjiska and Gligoroski, 2016)

Moody and Pelner's analysis in Moody and Perlner (2016) shows that this cryptosystem is vulnerable to an efficient and practical attack that can be used to recover the private keys. In particular it points out that an attacker can opt to use the parity check matrix H instead of the generator matrix G which makes the attack even more efficient. The following section outlines an improvement to this cryptosystem to secure it against such attacks.

Algorithm 13: St-Gen Signature Procedure

```
Input  : message $m \in \mathbb{F}_2^n$, private key $S, G$ and $P$
Output: A valid signature $\sigma \in \mathbb{F}_2^k$
    1. Compute $y = zP^{-1}$.
    2. Decode $y$ using algorithm 12 to get s.
    3. Compute $\sigma = sS^{-1}$.
    4. Return $\sigma$.
```

Algorithm 14: St-Gen Verification Procedure

```
Input  : pair $(m, \sigma) \in \mathbb{F}_2^n \times \mathbb{F}_2^k$, public key $G_{pub}$
Output: True if signature is valid, false otherwise
    1. Calculate $e = \sigma G_{pub} + m$.
    2. Return true if $e \in E_l^m$, false otherwise.
```

3.6 Random Split of Staircase Generator Codes Cryptosystem

Random split of St-Gen codes is a variation of the cryptosystem given in the previous section. It aims at hiding the structure of the generator matrix exposed in the public key, thus securing it against ISD attacks and distinguisher attacks. The proposed modification is random splitting of the public key matrix. By introducing a new parameter s to the system, which defines the number of splits, we guarantee that the probability of mounting a successful ISD attack against the system becomes negligible.(Samardjiska and Gligoroski, 2016; Gligoroski and Samardjiska, 2016a)

3.6.1 Valid Error Splits

Taking the same notion of an error set introduced in section 3.5.1, the proposal introduces the concept of a valid error split. Let E_l be an error set of granulation l, density ρ and s being the number of summands in which the generator matrix will be split. An error split is defined as any $s-$element set $\{e_1, \dots, e_s\}$, where $e_i \in \mathbb{F}_2^l$, for $1 \leq i \geq s$, if and only if the sum of all its elements permuted by any permutation $\sigma_i \in \mathbb{S}_l$ is an element in E_l according to the following equation:

$$e = \sum_{i=1}^{s} \sigma_i(e_i) \in E_l \tag{3.14}$$

As a concrete example of valid error splits, we take $l = 4$, $s = 4$and E_l to be the error set {{0,0,0,0}, {0,0,1,1}, {0,1,0,1}, {0,1,1,0}, {1,0,1,1}, {1,1,0,0}, {1,1,1,0}, {1,1,1,1}}.

A vald error split for E_l would be the set 1,0,0,0, 1,1,1,1, 1,1,1,1, 1,1,0,1 as the sum of its all elements permuted by all 24 possible permutations would result into an element in E_l. It is important to note that the elements in a valid error split do not have to be themselves in E_L.

Algorithm 15 outlines the procedure of defining a set of all valid error splits. While this algorithm is quite computationally expensive, it should be noted that it is only run once during an initialization phase of the system. Moreover this set can be precomputed and publicly available in practice.

Algorithm 15: St-Gen Valid Error Splits Procedure

```
Input  : granulation $l$, error set $E_l$, number of splits $s$
Output: a set of all valid error splits ValidErrorSplits
for $\forall(e_1, \ldots, e_s) \in (\mathbb{F}_2^l)^s$. do
    if $\sum_{i=1}^{s} \sigma_i(e_i) \in E_l \forall(\sigma_1, \ldots, \sigma_s) \in (\mathbb{S}_l)^s$ then
        Add $(e_1, \ldots, e_s)$ to ValidErrorSet.
    end
end
```

3.6.2 Key Generation Procedure

Algorithm 16 provides a variation of algorithm 9 with splitting of the public key, given granulation l, (n, k) linear binary code C and $m = n/l$.

The public key is given by $G_{pub}^1, G_{pub}^2, \ldots, G_{pub}^s$ and the private key is given by $S, G, P_1, P_2, \ldots, P_s$

3.6.3 The Encryption Scheme

Given the keys generated in algorithm 16, algorithm 17 outlines the procedure used to encrypt a message m of length n. Algorithm 18 gives a similar description for the decryption process. It should be noted that these algorithms are slower than their McEliece counterparts by a linear factor s for encryption, and a small constant overhead for decryption.

3.6.4 The Digital Signatures Scheme

Using the same keys generated using 16, we can outline two procedures for digitally signing a message and verifying the signature. Once again the use of staircase generator codes allows for a digital signature scheme that is directly translated from the encryption scheme without the need for any specially introduced mechanics.

Algorithm 12 is used again to only produce a single decoding needed as a signature, in contrast to algorithm 8 that returns the list of all possible decodings needed for decryption. Algorithm 19 outlines a procedure to find a signature using a private key and algorithm 20 outlines another procedure to verify a given signature is valid using the corresponding public key.

Algorithm 16: Random Split St-Gen Key Generation with Procedure

1. Generate a random generator matrix G of dimension $k \times n$ with the form given in figure 3.8, capable of generating an (n, k) linear binary code.
2. Generate an array of permutation matrices $P_1, P_2, \ldots, P_s$, selected as follows: Select permutation π on $\{1, 2, \ldots, m\}$ and let P be any permutation matrix induced by π such that:
$$\begin{aligned} y &= y_1||y_2||\cdots||y_m \\ yP &= y_{\pi(1)}||y_{\pi(2)}||\cdots||y_{\pi(m)} \end{aligned} \tag{3.15}$$
For $1 \leq i \leq s$, select m random permutations $\sigma_j^i \in \mathbb{S}_l$, for $1 \leq j \leq m$. Define each P_i according to the following equation:
$$yP = \sigma_1^i(y_{\pi(1)})||\sigma_2^i(y_{\pi(1)})||\cdots||\sigma_m^i(y_{\pi(1)}) \tag{3.16}$$
where $\sigma_j^i = \sigma_j^i(x_1, x_2, \ldots, x_l)$.
3. Select a random invertible $k \times k$ matrix S over $\mathbb{F}_2$.
4. Generate uniformly at random matrices $G_1, G_2, \ldots, G_{s-1}$ of size $k \times n$ over $\mathbb{F}_2$.
5. Compute $G_s = G + G_1 + G_2 + \cdots + G_{s-1}$.
6. Compute $G_{pub}^i = SG_iP_i$, for $1 \leq i \leq s$.

3.6.5 Analysis of St-Gen codes cryptosystem with random split

Here will give a theoretical discussion of the effectiveness of the split technique as a measure against the ISD attacks. Chapter 5 will provide analysis into the practical performance of the cryptosystem based on the implementation carried out for this thesis.

A simple way of defending against ISD attack in the St-Gen codes cryptosystem is to increase the parameters of the cryptosystem (mainly n and k). Unfortunately that would require parameters that are no longer practical. Instead Gligoroski and Samardjiska (2016a) and Samardjiska and Gligoroski (2016) propose the technique of splitting as a defense against ISD attacks. The idea is to make the conditions required for an attacker to mount a successful attack have negligible probability of occurrence.

The following shows how the splitting technique achieves the above goal. We refer the reader to Samardjiska and Gligoroski (2016) for a more detailed proof. First we define every permutation matrix $P_i = PP_i^*$ where P is a permutation matrix that permutes whole blocks of length l and P_i^* is a permuation matrix which only permutes within a single block. We then define $G_{p,j}^i$ to be a submatrix of SG_iP.

This would mean that applying a permutation $\sigma_j^i(G_{p,j}^i)$ corresponding to a permutation matrix P_i^* gives the submatrix $G_{pub,j}^i$. It is easy to see that in order to mount an ISD attack,

```
Algorithm 17: Random Split of St-Gen Encryption Procedure
  Input  : message $m$, public key $G^1_{pub}, G^2_{pub}, \ldots, G^s_{pub}$, set of valid error splits
           ValidErrorSplit
  Output: cipher text $c_i$, for $1 \leq i \leq s$
    1. Generate error vectors $e_i$, with length $n$ for $1 \leq i \leq s$, where $e^j_i$ is taken from
       $Split_j = (e^j_1, e^j_2, \ldots, e^j_s) \in ValidErrorSplits$.
    2. Compute $c_i = mG^i_{pub} + e_i$.
```

```
Algorithm 18: Random Split of St-Gen Decryption Procedure
  Input  : cipher text $c_i$, for $1 \leq i \leq s$, private key $S, G, P_1, P_2, \ldots, P_s$
  Output: decrypted message $m$
    1. Compute $c'_i = c_i P_i^{-1}$.
    2. Compute $c' = \sum_{i=1}^{s} c'_i$.
    3. Compute $m'$ as output of decoding algorithm 8.
    4. Compute $m = m'S{-}1$
    5. Return $m$.
```

the attacker must find permutations μ^i_j such that the equation:

$$\sum_i \mu^i_j(G^i_{pub,j}) = \sum_i \mu^i_j \sigma^i_j(G^i_{p,j}) \tag{3.17}$$

holds for $1 \leq i \leq s$ and $1 \leq j \leq \frac{n}{l}$. In fact, the case in which this happens the key structure degrades into the case of the cryptosystem without splitting. In Samardjiska and Gligoroski (2016), it is shown that the probability of $\mu^i_j \sigma^i_j$ agreeing on t coordinates is:

$$\left(\frac{\binom{n}{k}\lfloor\frac{(l-t)!}{e}+\frac{1}{2}\rfloor}{l!}\cdot\left(\frac{(l-t)!}{l!}\right)^{s-2}\right)^{\frac{k}{t}} \tag{3.18}$$

where l is the granulation. This is a negligible probability of being able to reveal the structure of a submatrix of G, making it even harder to reveal the structure of the entire generator matrix.

Algorithm 19: Random split of St-Gen codes Signature Procedure

```
Input  : message $z_1, z_2, \dots, z_s \in \mathbb{F}_2^n$, private key $S, G$ and $P_1, P_2, \dots, P_s$
Output: A valid signature $\sigma \in \mathbb{F}_2^k$
    1. Compute $y_i = z_i P_i^{-1}$.
    2. Compute $y = \sum_{i=1}^{s} y_i$.
    3. Decode $y$ using algorithm 12 to get s.
    4. Compute $\sigma = sS^{-1}$.
    5. Return $\sigma$.
```

Algorithm 20: Random split of St-Gen codes Verification Procedure

```
Input  : pair $(z, \sigma) \in \mathbb{F}_2^n \times \mathbb{F}_2^k$, public key $G_{pub}^1, G_{pub}^2, \dots, G_{pub}^s$
Output: True if signature is valid, false otherwise
    1. Calculate $e_i = \sigma G_{pub}^i + z_i$.
    2. Return true if $e \in E_l^m$, false otherwise.
    3. If $(e_{j,1}, e_{j,2}, \dots, e_{j,s}) \in ValidErrorSplits$, where $1 \leq j \leq s$, return true.
    4. Otherwise return false.
```

Chapter 4

Experiment

This chapter will highlight some aspects of the St-Gen cryptosystem implementation that was done for this thesis. It will also cover the setup for the experiments carried out to establish the performance of the system.

4.1 Design Choices

4.1.1 Programming Language

The implementation for the cryptosystem was done using the C programming language. This decision was based on prioritizing efficiency over implementation elegance. The C programming language is a very low level language which allows execution speed to come very close to the maximum capabilities of the hardware.

The C programming language also offers other advantages such as full control over memory management and access to low level instructions that can be used for highly optimized code snippets. Finally it offers maximum portability for different platforms. For these reasons most commonly used cryptographic libraries have implementation in C or would be implemented in C before mass adoption.

4.1.2 Data Structures and Interface

Most of the data bulk that go into the cryptosystem take the form of matrices. A decision was made not to use any specialized C libraries for handling matrices or matrix operations. This reduces that chances for any hidden operations that may be affecting the running time of different procedures. This also allows clearly identifying optimization vectors that are implemented or need to be implemented.

For the above reasons all matrices are represented by simple multidimensional arrays. The implementations also defines basic matrix operations such as matrix addition, multiplication, the transpose operation and the inverse matrix calculation procedure.

The cryptosystem provides a simple interface with a common key generation procedure, an encryption and decryption procedure for the encryption scheme and a signature and verification procedures for the digital signature scheme. Both schemes share the use of many underlying helper functions such decoding functions.

4.2 Setup

Once the correctness of the cryptosystem is established by carrying out successful encryption and decryption for the encryption scheme and a signing followed by verification for the signature scheme, two experiments were setup to compare the performance of this cryptosystem to commonly used cryptosystems.

The experiments were run on a personal laptop running Arch Linux system. The processor specification as returned by the **lscpu** command line tool is an Intel Core I7-6600U model with two cores and two threads per core. The base clocking speed is 2.6 GHz and it has 4 MB SmartCache.

Taking advantage of cache hits in this case is considered a clear vector of optimization for improving matrix multiplication efficiency. The system also has 16 GB of memory which is was considered noteworthy due to the heavy memory requirements mandated by the size of the keys and the list decoding algorithm.

The standard implementation of the clock function in the $time\_t.h$ C header file was used to time the different procedures of the implementation, mainly the encryption, decryption, singing, verification and key generation procedures. Two experiments were run, one for each scheme of the cryptosystem, each five times. The times reported by each experiment were recorded and the average over the five runs was calculated to avoid any sharp variations due to execution anomalies.

For both experiments we used the concrete set of parameters given in Samardjiska and Gligoroski (2016) for the encryption scheme and Gligoroski and Samardjiska (2016a) for the digital signature scheme. These parameters are reported to offer 128bits of the security. For a discussion of how these parameters are established the reader is referred to the above literature.

To showcase the efficiency of the implementation we compare the results against the RSA cryptosystem running time. To acquire benchmarks for RSA, we use the **OpenSSL** command line tool speed command with cipher RSA4096 with -elapsed option to measure the total time consumed by the encryption/verification and decryption/sign procedures. Chapter 5 reports the recorded results and discusses their implications.

Chapter 5

Analysis

This chapter will outline the results recorded from the experiments described in the previous chapter. It will also provide a brief discussion on the implications of these results.

5.0.1 Correctness

The correctness of the implementation of the two schemes was verified. For the encryption scheme, correctness is established if a binary vector can be encrypted and decrypted with the decryption outcome being identical to the original. The correctness of the signature scheme is based on the signing of a binary vector followed by successful verification of that signature.

It should be noted that for the signature scheme verification can occasionally fail. This is attributed to the probablistic nature of the algorithm. Since other probabilistic digital signature schemes exhibit the same behavior, this was not considered incorrect.

5.0.2 Efficiency

Table 5.1 lists the running times recorded for our implementation of the cryptosystem. It can be seen that the times for the signing procedure exhibit some sharp variations and is considerably higher than the running time of the decryption procedure.

Despite both procedures using the same steps, the main difference lies in the decoding algorithm. The signing procedure uses algorithm 12 while the decryption procedure uses 8. Algorithm 8 does list decoding until the correct decoding is found. With the assumption that the size of the list is reduced with each iteration, the complexity of the algorithm is dominated by the size of the initial list.

In algorithm 12, however, the algorithm has to start over if the signature found is not decodable. This can mean many extra iterations while looking for a signature due to the probabilistic nature of the algorithm. This would also explain the wide variations in the recorded running times. We hypothesis that this running time can be improved by increasing the threshold used to limit the sizes of the lists in the list decoding algorithm for signatures.

Table 5.2 gives the same recordings of running times for RSA, as retrieved from the **OpenSSL** benchmarking tool. It should be noted that the tool only reports times for the sign and verify procedures so the decryption and encryption procedures are assumed to be the same. The averages of both systems are compared in table 5.3.

It is assumed that the running procedures of RST-GEN cryptosystem would be lower than the RSA system. This is based on the fact that the RST-GEN cryptosystem procedures mostly rely on matrix multiplication while RSA system procedures rely on exponentiation and modulus division, both of which are very expensive operations. It can be seen that table 5.3 does not support this assumption.

We hypothesis that this is a flaw in the implementation and not in the design of the cryptosystem. Our implementation has minimal optimization while being compared against the **OpenSSL** highly optimized mass adopted implementation. This largely due to prioritizing a proof of concept rather than efficiency during implementation.

However, we do outline several clear optimization that were not implemented due time constraints. The current implementation of matrix multiplication is quite inefficient due to the use of column major order in matrix representation, which leads to cache thrashing. Also the permutations matrices are treated as normal matrices and are applied through matrix multiplication. It would be much more efficient to store the permutations as a sequence of indices and apply them as such.

Table 5.1: Running times for RST-GEN

Sign	**Verify**	**Encryption**	**Decryption**
3.347577	0.002889	0.000721	2.166893
15.62105	0.003694	0.000681	1.801135
15.268985	0.002673	0.000719	1.692233
104.251773	0.003141	0.000713	1.494761
24.042725	0.003259	0.000697	1.503881
Average			
32.506422	0.0031312	0.0007062	1.7317806

Table 5.2: Running times for RSA

Sign	**Verify**	**Encryption**	**Decryption**
0.00565	0.000084	0.000084	0.00565
0.005727	0.000085	0.000085	0.005727
0.004941	0.000072	0.000072	0.004941
0.005308	0.000084	0.000084	0.005308
0.005297	0.000082	0.000082	0.005297
Average			
0.0053846	0.0000814	0.0000814	0.0053846

Table 5.3: Comparing average running times for RSA and RST-GEN

Alogrithm	**Sign**	**Verify**	**Encryption**	**Decryption**
RSA4096	0.0053846	0.0000814	0.0000814	0.0053846
RST-GEN	32.506422	0.0031312	0.0007062	1.7317806

Chapter 6

Conclusion

Cryptography provides the corner stone of securing digital information. It guarantees confidentiality, integrity and authenticity of data stored and passed around on the internet. Cryptosystems such as RSA and Deffie-Hellman key exchange are the most commonly used systems for that purpose. These systems have made use of the same hard mathematical problems that have been believed to be beyond the capability of classical computers to guarantee their security.

However, quantum computers stand to threaten this established infrastructure by promising to allow attackers to solve those problems in polynomial times. Cryptography research efforts must provide new families of cryptosystems that rely on different paradigms of problems for their security, to be able to remain secure in the post quantum era.

Many existing cryptography paradigms are believed to be resistant to quantum attacks. Among these are hash based cryptography, multivariate cryptography, lattice based cryptography and code based cryptography. While these paradigms have historically lacked popularity, they offer cryptosystems that stand as candidates for mass adoption as post quantum cryptographic systems.

This thesis is mainly concerned with one such system that comes out of code based cryptography. This system follows the same structure of the McEliece system, one of the earliest examples of code based cryptographic systems. The McEliece system has stood the test of cryptoanalysts for over thirty years, and offers efficient encryption and decryption algorithms that rival the currently used RSA.

Based on its scheme, a new system was developed and named staircase generator codes cryptosystem. This system makes use of a newly proposed subfamily of Goppa codes named staircase generator codes which imposes new restrictions on the generator matrix structure. It also innovates a new approach to error vectors and a new list decoding algorithm. The new system holds the advantage, over the McEliece system, by providing a digital signature scheme directly adaptable from the encryption scheme using the decryption procedure.

Having been shown to be susceptible to structure attacks, due to the specific structure of the generator matrix, the new system is modified with a splitting technique for the public key. This modified system, referred to in this thesis as randomly split ST-Gen codes cryptosystem, is inoculated against structural attacks such as ISD.

In this thesis, we provide and discuss a C implementation of the RST-Gen code cryptosystem. The implementation showcases the correctness of the system and its underlying concepts. The efficiency of the implementation is benchmarked and compared against benchmarks from the RSA cryptosystem. For future work, we recommend addressing the missing optimizations mentioned in chapter 5 as well as those that have been overlooked. We also postulate the following research question. How, if possible, can the size of the keys of code based cryptographic systems be reduced.

Bibliography

Au, S., Eubanks-Turner, C., Everson, J., 2003. The mceliece cryptosystem. Unpublished manuscript 5.

Bernstein, D. J., Buchmann, J., Dahmen, E., 2009. Post-quantum cryptography. Springer Science & Business Media.

Buchmann, J., Ding, J., 2008. Post-quantum cryptography. In: second international workshop, PQCrypto. pp. 17–19.

Cayrel, P.-L., Meziani, M., 2010. Post-quantum cryptography: Code-based signatures. In: Advances in Computer Science and Information Technology. Springer, pp. 82–99.

Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D., 2016. Report on post-quantum cryptography. National Institute of Standards and Technology Internal Report 8105.

Courtois, N. T., Finiasz, M., Sendrier, N., 2001. How to achieve a mceliece-based digital signature scheme. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer, pp. 157–174.

Ding, J., Yang, B., 2009. Post-quantum cryptography.

Gligoroski, D., Samardjiska, S., 2016a. A digital signature scheme based on random split of st-gen codes. IACR Cryptology ePrint Archive 2016, 391.

Gligoroski, D., Samardjiska, S., 2016b. Semantic security and key-privacy with random split of st-gen codes. In: Conference on Computability in Europe. Springer, pp. 105–114.

Gligoroski, D., Samardjiska, S., Jacobsen, H., Bezzateev, S., 2014. Mceliece in the world of escher. IACR Cryptology ePrint Archive 2014, 360.

Kabatianskii, G., Krouk, E., Smeets, B., 1997. A digital signature scheme based on random error-correcting codes. Crytography and Coding, 161–167.

Menezes, A. J., Van Oorschot, P. C., Vanstone, S. A., 1996. Handbook of applied cryptography. CRC press.

Moody, D., Perlner, R., 2016. Vulnerabilities of mceliece in the world of escher. In: International Workshop on Post-Quantum Cryptography. Springer, pp. 104–117.

Mosca, M., 2013. Post-quantum cryptography. Lecture Notes in Computer Science 8772.

Mosca, M., 2014. Post-Quantum Cryptography: 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings. Vol. 8772. Springer.

Perlner, R. A., Cooper, D. A., 2009. Quantum resistant public key cryptography: a survey. In: Proceedings of the 8th Symposium on Identity and Trust on the Internet. ACM, pp. 85–93.

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

Samardjiska, S., Gligoroski, D., 2016. An encryption scheme based on random split of st-gen codes. In: Information Theory (ISIT), 2016 IEEE International Symposium on. IEEE, pp. 800–804.

Sendrier, N., 2010. Post-quantum cryptography. In: third international workshop, PQCrypto, Darmstadt, Germany. Springer.

Sendrier, N., Tillich, J.-P., 2014. Private communication, october 2014. Cited on, 2.

Takagi, T., 2016. Post-quantum cryptography. Lecture Notes in Computer Science 9606.

Umana, V. G., 2011. Post-quantum cryptography. Post-Quantum Cryptography.

Yang, B.-Y., 2008. Post-quantum cryptography.

Yang, B.-Y., 2011. Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29-December 2, 2011, Proceedings. Vol. 7071. Springer.

Appendix

Listing 6.1: RST-Gen Codes cryptosystem Interface

```
#include "reference_s.h"
#include "params.h"
#include "util.h"
#include "print_utils.h"

#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>


/* ———————————— Key generation functions ————————————
    */

static void create_S(size_t k, uint8_t S[k][BYTES(k)]);
static void create_G(size_t k, size_t n, uint8_t G[n][BYTES
   (k)]);
static void create_P(size_t n, uint8_t P[n][BYTES(n)]);
static void create_Pi_from_P(size_t n, uint8_t P[n][BYTES(n
   )], uint8_t Pi[n][BYTES(n)]);

void key_gen(size_t k,
             size_t n,
             size_t s,
             uint8_t S_inv[k][BYTES(k)],
             uint8_t G[n][BYTES(k)],
             uint8_t P_inv[s][n][BYTES(n)],
             uint8_t G_pub[s][n][BYTES(k)])
{

    uint8_t S[k][BYTES(k)];
    uint8_t p[s][n][BYTES(n)]; printf("here...\n");
    uint8_t P[n][BYTES(n)];
    uint8_t SG[s][n][BYTES(k)];
    uint8_t g[s][n][BYTES(k)];
    create_S(k, S);
    invert(k, S, S_inv);

    create_P(n, P);
    create_G(k, n, G);

    // Compute G_1 ... G_s-1
    memset(g, 0, s * n * BYTES(k));
    for (size_t i = 0; i < s-1; i++)
      random_matrix(k, n, g[i]);

    // Compute G_s
    matrix_add(k, n, G, g[s-1], g[s-1]);
    for(size_t i = 0; i < s - 1; i++)
      matrix_add(k, n, g[i], g[s-1], g[s-1]);

    // Compute public key Gi
    memset(G_pub, 0, s* n * BYTES(k));
    memset(SG, 0, s * n * BYTES(k));
    for (size_t i = 0; i < s; i++) {
      matrix_multiply(k, k, n, S, g[i], SG[i]);
      create_Pi_from_P(n, P, p[i]);
      matrix_multiply(k, n, n, SG[i], p[i], G_pub[i]);
    }

    // Compute the inverse permutation
    for (size_t i = 0; i < s; i++)
      invert_permutation_matrix(n, p[i], P_inv[i]);
}



static void create_S(size_t k, uint8_t S[k][BYTES(k)])
{
    memset(S, 0, k * BYTES(k));
    do {
        random_matrix(k, k, S);
    } while (rank(k, k, S) != k);
}

static void create_G(size_t k, size_t n, uint8_t G[n][BYTES
    (k)])
{
        memset(G, 0, n * BYTES(k));
        identity_matrix(k, n, G);

    size_t Ki = 0;
    size_t ni = k;
        for (size_t i = 0; i < NUMBER_OF_BLOCKS; i++) {
        Ki += B_k[i];
                for (size_t column = ni; column < ni + B_n[
                    i]; column++) {
                        uint8_t random_column[BYTES(k)];
                        for (size_t j = 0; j <= Ki / 8; j
                            ++) {
                                random_column[j] = randr
                                    (0,255);
                        }
                        BytewiseOperation(xor, k, 0, Ki, G[
                            column], random_column, G[column
                            ]);
                }
        ni += B_n[i];
        }
}

static void create_P(size_t n, uint8_t P[n][BYTES(n)])
{
  unsigned int perm[n/L];
  memset(P, 0, n * BYTES(n));
  random_permutation(n / L, perm);

  for (size_t i = 0; i < n / L; i++) {
    for (size_t j = 0; j < L; j++)
      P[L * i + j][(L * perm[i] + j) / 8] = 1 << (7 - (L *
          perm[i] + j) % 8);
  }

}

static void create_Pi_from_P(size_t n, uint8_t P[n][BYTES(n
    )], uint8_t Pi[n][BYTES(n)])
{
  unsigned int perm[L];
  memcpy(Pi, P, n*BYTES(n));
  for (size_t i = 0; i < n / L; i++) {
    memset(perm, 0, L);
    random_permutation(L, perm);
    permute_columns(n, n, Pi, L, perm, i * L);
  }
}


/* —————————————— Encryption/decryption functions
    ————————— */

//TODO: Fix error vector input
void encrypt(size_t k, size_t n, size_t s, uint8_t G_pub[s
    ][n][BYTES(k)], uint8_t m[BYTES(k)], uint8_t e[s][BYTES(
    n)], uint8_t c[s][BYTES(n)])
{
    memset(c, 0, s*BYTES(n));
    for (size_t i = 0; i < s; i++) {
      vector_matrix_mult(k, n, m, G_pub[i], c[i]);
      BytewiseOperation(xor, n, 0, n, c[i], e[i], c[i]);
            // TODO: fix error vector application
    }
}

void decrypt(size_t k, size_t n, size_t s, uint8_t S_inv[k
    ][BYTES(k)], uint8_t G[n][BYTES(k)], uint8_t P_inv[s][n
    ][BYTES(n)], uint8_t z[s][BYTES(n)], List *decrypted)
{
  uint8_t m[BYTES(k)];
  uint8_t y[BYTES(n)];
  uint8_t c_prime[s][BYTES(n)];
  List L;
  list_init(&L, k);

  memset(c_prime, 0, s*BYTES(n));
  memset(y, 0, BYTES(n));

  for (size_t i = 0; i < s; i++) {
      vector_matrix_mult(n, n, z[i], P_inv[i], c_prime[i]);
      BytewiseOperation(xor, n, 0, n, c_prime[i], y, y);
  }

  decode(k, n, G, y, &L);

  for (size_t i = 0; i < L.size; i++) {
    vector_matrix_mult(k, k, list_get(&L, i), S_inv, m);
    list_append(decrypted, k, m);
  }
  list_free(&L);
}



/* ------------ Decoding functions ------------ */

static void create_first_block_candidates(List *T0, size_t
    k, size_t n,
     uint8_t G[n][BYTES(k)], uint8_t y[BYTES(n)], uint8_t y0
        [BYTES(k)]);

static bool valid_candidate(size_t k, size_t n, uint8_t G[n
    ][BYTES(k)], uint8_t y[BYTES(n)], uint8_t cand[BYTES(k)
    ], size_t block);

static void extend(List *next, size_t k, uint8_t y0[BYTES(k
    )], uint8_t cand[BYTES(k)], size_t block, size_t step);

static unsigned int list_decode(size_t k, size_t n, uint8_t
     G[n][BYTES(k)], uint8_t y[BYTES(n)],uint8_t y0[BYTES(k)
    ], List *Lp, size_t start_round);


void decode(size_t k, size_t n, uint8_t G[n][BYTES(k)],
    uint8_t y[BYTES(n)], List *Lp)
{
  uint8_t y0[BYTES(k)];
  memcpy(y0, y, sizeof(y0));
  create_first_block_candidates(Lp, k, n, G, y, y0);
  list_decode(k, n, G, y, y0, Lp, 1);
}

static unsigned int list_decode(size_t k, size_t n,
     uint8_t G[n][BYTES(k)],
     uint8_t y[BYTES(n)],
     uint8_t y0[BYTES(k)],
     List *Lp,
     size_t start_block)
{
  List T;
  list_init(&T, k);

  printf("L%zd = %zd\n", start_block, Lp->size);
  for (size_t block = start_block; block < NUMBER_OF_BLOCKS
      ; block++) {
    T.size = 0;
    for (size_t i = 0; i < Lp->size; i++) {
      extend(&T, k, y0, list_get(Lp, i), block, 0);
    }

    Lp->size = 0;
    for (size_t i = 0; i < T.size; i++) {
      if ( valid_candidate(k, n, G, y, list_get(&T, i),
          block) ) {
        list_append(Lp, k, list_get(&T, i));

        if (SIGNATURE) {
          list_free(&T);
          return 1;
        }
      }
    }

    printf("L%zd␣=␣%zd\n", block + 1, Lp->size);
  }
  list_free(&T);

  return Lp->size;
}

static void create_first_block_candidates(List *T0, size_t
    k, size_t n,
     uint8_t G[n][BYTES(k)], uint8_t y[BYTES(n)], uint8_t y0
         [BYTES(k)])
{
        List E_K0;
        list_init(&E_K0, k);
    uint8_t seed_error[BYTES(k)];
    memset(seed_error, 0, sizeof(seed_error));
        all_errors(&E_K0, B_k[0], k, seed_error);

    for (size_t i = 0; i < E_K0.size; i++) {
        uint8_t (*e)[BYTES(k)] = list_get(&E_K0, i);
                BytewiseOperation(xor, k, 0, B_k[0], *e, y0
                    , *e);
        if ( valid_candidate(k, n, G, y, *e, 0) ) {
            list_append(T0, k, *e);
        }
    }

        list_free(&E_K0);
}

static bool valid_candidate(size_t k, size_t n, uint8_t G[n
    ][BYTES(k)], uint8_t y[BYTES(n)], uint8_t cand[BYTES(k)
    ], size_t block)
{
        uint8_t cand_encoding[BYTES(n)];
    memset(cand_encoding, 0, sizeof(cand_encoding));
        size_t block_start_column = k + sum(block, B_n);
    uint8_t mask = 0xff >> (8-L);

        for (size_t i = block_start_column; i <
            block_start_column + B_n[block]; i += L) {

        for (size_t j = 0; j < L; j++)
            cand_encoding[(i+j) / 8] ^= (scalar_prod(k, n,
                i+j, cand, G) << (7 - (i+j) % 8));

                uint8_t slice = ((cand_encoding[i / 8] ^ y[
                    i / 8]) >> (6 - i % 8)) & mask;

                if (slice == mask) {
                        return false;
                }
        }

        return true;
}

static void extend(List *next, size_t k, uint8_t y0[BYTES(k
    )], uint8_t cand[BYTES(k)], size_t block, size_t step)
{
    uint8_t x[BYTES(k)];
    memset(x, 0, sizeof(x));

        if (step < B_k[block]) {

                memcpy(x, cand, sizeof(x));
                size_t step_slice = sum(block, B_k) + step;
                uint8_t y0_ki_mask = 0x3 << (6 - step_slice
                    % 8);
                // e = 00
                x[step_slice / 8] ^= (y0[step_slice / 8] &
                    y0_ki_mask);
                extend(next, k, y0, x, block, step + 2);

                // e = 01
                x[step_slice / 8] ^= (0x55 & y0_ki_mask);
                extend(next, k, y0, x, block, step + 2);

                // e = 10
                x[step_slice / 8] ^= (0xff & y0_ki_mask);
                extend(next, k, y0, x, block, step + 2);
        } else {
                list_append(next, k, cand);
        }
}


/* ——————————— Signature/verification functions
   ——————————*/

static void sign_decode(size_t k, size_t n,
    uint8_t G[n][BYTES(k)], uint8_t y[BYTES(n)], uint8_t x[
        BYTES(k)]);

static bool survive_block(size_t k, size_t n, size_t Ki,
    size_t block,
    uint8_t G[n][BYTES(k)], uint8_t y[BYTES(n)], uint8_t y0
        [BYTES(k)], uint8_t x[BYTES(k)]);

static void expand(size_t k, size_t Ki, size_t ki, size_t
    ni,
    List *L, uint8_t y0[BYTES(k)], uint8_t x[BYTES(k)]);

static size_t exp_limit(size_t ni)
{
    return ceil(pow(2, ni) / pow(3, ni / 2));
}


unsigned int sign(size_t k, size_t n,
    uint8_t S_inv[k][BYTES(k)],
    uint8_t G[n][BYTES(k)],
    unsigned int inv_perm[n / 2],
    uint8_t z[BYTES(n)],
    uint8_t signature[BYTES(k)])
{
    uint8_t x[BYTES(k)], y[BYTES(n)];
    memset(x, 0, sizeof(x));
    memset(y, 0, sizeof(y));

        uint8_t ztemp[n], ytemp[n];

        // inverse permute z to obtain y
        size_t j = 0;
        for (size_t i = 0; i < BYTES(n); i++) {
        ztemp[j++] = get_bit(z[i], 0);
        ztemp[j++] = get_bit(z[i], 1);
        ztemp[j++] = get_bit(z[i], 2);
        ztemp[j++] = get_bit(z[i], 3);
        ztemp[j++] = get_bit(z[i], 4);
        ztemp[j++] = get_bit(z[i], 5);
        ztemp[j++] = get_bit(z[i], 6);
        ztemp[j++] = get_bit(z[i], 7);
    }

    for (size_t i = 0; i < n / 2; i++) {
        ytemp[2 * i] = ztemp[2 * inv_perm[i]];
        ytemp[2 * i + 1] = ztemp[2 * inv_perm[i] + 1];
    }

        for (size_t i = 0; i < BYTES(n); i++) {
        y[i] = (ytemp[8*i])<<7     | (ytemp[8*i + 1])<<6 \
                           | (ytemp[8*i + 2])<<5 | (ytemp[8*i
                              + 3])<<4 \
                           | (ytemp[8*i + 4])<<3 | (ytemp[8*i
                              + 5])<<2 \
                           | (ytemp[8*i + 6])<<1 | (ytemp[8*i
                              + 7]);
    }

        sign_decode(k, n, G, y, x);
    vector_matrix_mult(k, k, x, S_inv, signature);

    return 1;
}

static void sign_decode(size_t k, size_t n, uint8_t G[n][
   BYTES(k)], uint8_t y[BYTES(n)], uint8_t x[BYTES(k)])
```

```
{
    size_t list_decode_point = NUMBER_OF_BLOCKS - 1;
    uint8_t e0[BYTES(k)], y0[BYTES(k)];
    memcpy(y0, y, sizeof(y0));

        List L;
        list_init(&L, k);

        bool valid_cand = false;

    while (!valid_cand) {
                size_t Ki = B_k[0];
        memset(x, 0, BYTES(k));
        //random_error(0, Ki, e0);
        BytewiseOperation(xor, k, 0, Ki, y0, e0, x);

        if (valid_candidate(k, n, G, y, x, 0)) {
                        for (size_t block = 1; block <
                            list_decode_point; block++) {
                                if ( !survive_block(k, n,
                                    Ki, block, G, y, y0, x)
                                    ) {
                                        goto NEW_CANDIDATE;
                                }
                                Ki += B_k[block];
            }
            L.size = 0;
            list_append(&L, k, x);
            valid_cand = (list_decode(k, n, G, y, y0, &L,
                list_decode_point) > 0);
                }

        NEW_CANDIDATE:;
        }

    memcpy(x, list_get(&L, 0), BYTES(k));
    printf("sign_decode:␣L.size␣=␣%zd\n", L.size);
    printf("selected␣candidate:\n");
    print_vector(stdout, k, x);

    list_free(&L);
}

static bool survive_block(size_t k, size_t n, size_t Ki,
    size_t block, uint8_t G[n][BYTES(k)], uint8_t y[BYTES(n)
    ], uint8_t y0[BYTES(k)], uint8_t x[BYTES(k)])
{
        List L;
        list_init(&L, k);
        expand(k, Ki, B_k[block], B_n[block], &L, y0, x);

        for (size_t j = 0; j < L.size; j++) {
                if ( valid_candidate(k, n, G, y, list_get(&
                    L, j), block) ) {
                        BytewiseOperation(xor, k, Ki, Ki +
                            B_k[block], x, list_get(&L, j),
                            x);
                        list_free(&L);
                        return true;
                }
        }

        list_free(&L);

        return false;
}

static void expand(size_t k, size_t Ki, size_t ki, size_t
    ni, List *L, uint8_t y0[BYTES(k)], uint8_t x[BYTES(k)])
{
        uint8_t e_ki[BYTES(k)], expanded[BYTES(k)];
    for (size_t i = 0; i < exp_limit(ni); i++) {
        random_error(0, k, e_ki);
        memcpy(expanded, x, sizeof(expanded));
        BytewiseOperation(xor, k, Ki, Ki + ki, y0, e_ki,
            expanded);
                list_append(L, k, expanded);
    }
}

bool verify(size_t k, size_t n, uint8_t G_pub[n][BYTES(k)],
     uint8_t z[BYTES(n)], uint8_t sig[BYTES(k)])
{
        uint8_t y[BYTES(n)];
    memset(y, 0, sizeof(y));
        vector_matrix_mult(k, n, sig, G_pub, y);
        BytewiseOperation(xor, n, 0, n, y, z, y);
        for (size_t i = 0; i < n; i += 2) {
                uint8_t slice = (y[i / 8] >> (6 - i % 8)) &
                    0x3;
                if (slice == 0x3) {
            printf("Problem␣at␣i␣=␣%zd\n", i);
            print_vector(stdout, i + 2, y);

                        return false;
                }
        }

        return true;
}
```