Code-Based Post-Quantum Cryptography

Spatially-Coupled MDPC codes as variant for the McEliece cryptosystem

Master's Thesis in Communication Engineering

ISSAM MAAROUF

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis EX061/2019

Code-Based Post-Quantum Cryptography

Spatially-Coupled MDPC codes as variant for the McEliece cryptosystem

ISSAM MAAROUF

Department of Electrical Engineering
Division of Communication Systems
Chalmers University of Technology
Gothenburg, Sweden 2019

Code-Based Post-Quantum Cryptography
Spatially-Coupled MDPC codes as variant for the McEliece cryptosystem
ISSAM MAAROUF

Supervisor and examiner:
Alexandre Graell i Amat, Department of Electrical Engineering

Co-supervisor:
Eirik Rosnes, Simula UiB, Bergen, Norway

Master's Thesis EX061/2019
Department of Electrical Engineering
Division of Communication Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Code-Based Post-Quantum Cryptography
Spatially-Coupled MDPC codes as variant for the McEliece cryptosystem
ISSAM MAAROUF
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The security of all the traditional cryptosystems used in practice will no longer be sufficient to protect user data with the introduction of quantum computers in the future. It has been proven that with using certain algorithms a quantum computer attack can easily break these cryptosystems. Hence, post-quantum cryptography (PQC) aims at studying and developing new cryptosystems that will be robust from attacks by a quantum computer. Several methods have been proposed in the literature, where one of them uses error-correction codes to construct the cryptosystem. This method is referred to as code-based post-quantum cryptography.

All code-based post-quantum cryptosystems are inspired by a cryptosystem introduced by Robert McEliece in 1978. The research in code-based PQC aims at finding error-correction codes as a variation for the codes proposed by McEliece, in an attempt to reduce the large key sizes produced by these codes.

In this thesis, findings and results on using quasi-cyclic (QC) medium density parity check (MDPC) codes as variants for the McEliece cryptosystem are reproduced. Furthermore, spatially-coupled (SC) QC-MDPC codes are proposed and tested in an attempt to use codes with better error correction capability than QC-MDPC codes.

Acknowledgements

I would like first to give special thanks to my two supervisors, Alexandre Graell i Amat and Eirik Rosnes. I enjoyed very much working with you, and I have learned a great deal from you as well. I am looking froward to continuing working under your supervision.

Second, I would like to thank my parents, Ahmad and Iman, for giving me the chance to study abroad and chase my dreams. You always had my back and always believed in me. I will never be able to repay all what you have done for me, but I hope I will always make you proud. I would like to also thank my siblings, Bilal, Ismat, and Reina for being my inspiration and motivation. Especially my older brother Bilal, for always being my number one fan.

Finally, I would like to thank my friends Katrin, Abed, and Rasika, who became my second family. Thank you so much for making the two years I spent in Sweden, the best years of my life. You will always have a special place in my heart.

Issam Maarouf, Gothenburg, June 2019

Contents

List of Figures

List of Acronyms

QC Quasi-cyclic

LDPC Low density parity check

CNs Check-nodes

VNs Variable-nodes

SPA Sum-product algorithm

BP Belief propagation

DE Density evolution

SC Spatial-coupling/Spatially-coupled

FW Full window

SW Sliding window

BSC Binary symmetric channel

WF Work factor

MDPC Medium density parity check

FER Frame-error rate

MF Miladinovic and Fossorier

REMP Random erasure message passing

MBP Masked belief propagation

ws Window span

BEC Binary erasure channel

PD Peeling decoder

BER Bit-error rate

List of Symbols

$\boldsymbol{C}(n,k)$	Linear code of length n and dimension k
n	Code length
k	Code dimensions
$\mathbb{F}_2^n$	Finite binary feild of dimensions n
$\boldsymbol{G}$	Code generator matrix
$\boldsymbol{m}$	Uncoded message or channel message probability vector
$\boldsymbol{c}$	Codeword of code $\boldsymbol{C}$
t	Error-correction capability of code $\boldsymbol{C}$
$\boldsymbol{C}_{\mathsf{null}}$	Dual code of $\boldsymbol{C}$
$\boldsymbol{H}$	Parity check matrix
d_v	VN degree
d_c	CN degree
m_{ch}	Channel message value
$m_{\mathsf{v}\to\mathsf{c}}$	VN to CN message value
$m_{\mathsf{c}\to\mathsf{v}}$	CN to VN message value
c_b	Cyphertext bit
$N(v)$	Neighborhood of VN v
$N(c)$	Neighborhood of CN c
$\hat{c}$	Decoded codeword
β	Channel parameter (cross-over probability, erasure probability etc.)
l	iteration counter
p	VN to CN message probability vector
q	CN to VN message probability vector
p_{min}	Minimum probability in DE
b	Base protograph matrix entry
α	Gallager B decoding algorithm threshold
L	Coupling length
m_{sc}	Memory constraint length
B_n	Number of VN types in base protograph matrix
B_k	Number of CN types in base protograph matrix
M	Lifting factor
η	Ratior between B_n and B_k
e	Error vector
c'	Received codeword
Q	Circulant size
ϵ	Code ensemble
$N(\epsilon)$	Key space of code ensemble
$I_{\max}$	Maximum number of iterations

1

Introduction

The goal of any cryptosystem is to ensure a secure exchange of information between two parties over an insecure channel. This insecure channel may contain several unwanted users (eavesdroppers) that will attempt to read the message exchanged between the two intended parties. As a result, a cryptosystem will need to come up with encryption and decryption techniques to guarantee that the eavesdroppers will not be able to recover the received messages, which is shown more clearly in figure 1.1. In cryptography, the original message is referred to as plaintext, and denoted by m. When the plaintext is encrypted by an encryption algorithm, the result will be a cyphertext, denoted by C. Then, the original message can be restored from the cyphertext by performing decryption using a decryption algorithm. The decoding algorithm will change the plaintext to a cyphertext in a controlled process. The input to this controlled process is referred to as a key. The decoding algorithm will give a specific cyphertext from a plaintext for a specific key.

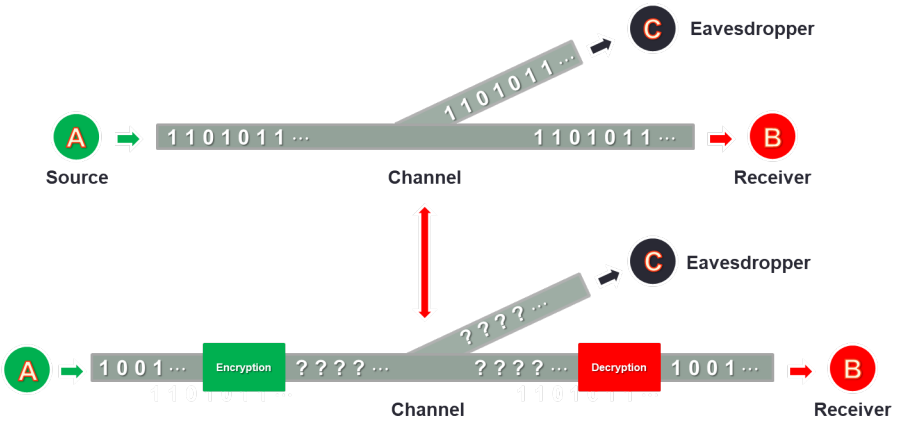

Figure 1.1: Basic Cryptography System.

Cryptosystems fall under two categories, the first referred to as secret-key cryptosystems, and the second referred to as public-key cryptosystems. Shannon is believed to be the first person to introduce the mathematical background of cryptography in the context of communication theory in his paper [1] back in 1949. It was not until 1974 where a global standard for secret-key algorithms was established and published by IBM [2]. IBM submitted a cryptogrpahic algorithm in response

to National Bureau of Standards (NBS) initiation program for developing a standard for the protection of online data [2]. This program was aided by the National Secretary Agency (NSA) of the US as well [2]. This standard was updated and enhanced in 2001, where the new algorithm is a subset of several algorithms developed by Vincent Rijmen and Joan Daemen in their paper [3], [4]. On the other hand, public-key cryptography was kick-started in 1976 by Whitfield Diffie and Martin Hellman, where they provided the mathematical background needed for public-key cryptosystems in their paper [5].

Both cryptosystem types are similar in security requirements, where it must be impossible for an attacker to gain any knowledge of the original message by knowing the cyphertext and the encryption/decryption algorithms. The main difference between the two types are the number of keys used in encryption/decryption, which will affect the way the message is encrypted. In secret-key cryptography, one key is used in encryption and decryption, referred to as the secret key. This key is only known and accessible to the source and receiver. As a result, only the source can encrypt the message, and only the intended receiver can decrypt it. Consequently, both source and receiver should have prior knowledge of the secret key. On the other hand, public-key cryptosystems use two keys, a public key, and a private key. The public key is accessible to everyone, while only the intended receiver will have access to the private key. The message is encrypted and decrypted by the public key and private key respectively. This cryptosystem can have several receivers for the same source, where each receiver will have a public and private key pair of their own. Now an eavesdropper may attempt to decrypt and recover the message; however, only by owning the private key, the decryption can be done efficiently, otherwise, it is nearly impossible to do. Figure 1.2 visualises the public-key cryptography system. Both cryptosystem types are still being used in practice depending on the security application, but in general no type offers better security than the other [6]. However, in this project, public-key cryptography is used.

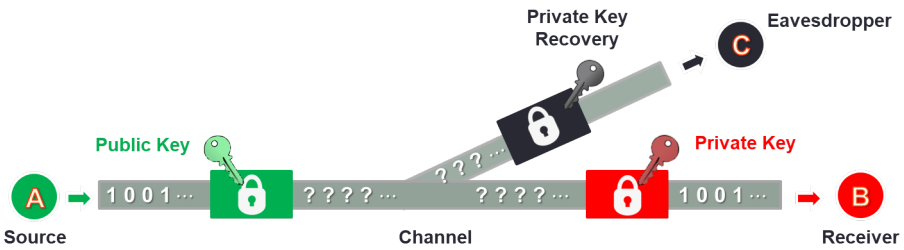

Figure 1.2: Basic Public Key Cryptography System.

As will be shown later, the security of all public-key cryptography systems rely on the high complexity of either prime factorization or computing discrete logarithms, depending on the encryption/decryption algorithm used. It is known that no algorithm up to date for performing these problems can solve them in reasonable

time [6]. However, with the introduction of quantum computers in the future, all these systems will no longer be safe. Since, by using Shor's algorithm [7], a quantum computer attack can solve these problems, and hence, break these cryptosystems in reasonable time [8]. As a result, new cryptosystems need to be developed that are robust against an attack using a quantum computer. These new cryptosystems need to provide security procedures that do not rely on the same mathematical principles used in public-key cryptography. Hence, the recent increase in interest in the field of post-quantum cryptography and its techniques, which provides security from attacks by a quantum computer.

Several post-quantum cryptography techniques have been proposed that will provide security against a quantum computer attack, and one of these techniques uses error-correction codes to perform the cryptography procedure. Using error-correction codes to perform post-quantum cryptography is a field of study referred to as code-based post-quantum cryptography. All current code-based cryptosystems are based on a cryptosystem developed by Robert McEliece, who introduced a cryptosystem that depends on the hardness of decoding a linear code [9].

1.1 In This Thesis

This thesis focuses on testing and analysing spatially-coupled (SC) quasi-cyclic (QC) medium density parity check (MDPC) codes , which will be explained later, as variant for the codes in the McEliece cryptosystem. This thesis builds on and extends the work done on this topic where several error-correction codes were introduced in the literature as variants as well. Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo Barreto used a basic QC-MDPC code ensemble as the variant in paper [8], and calculated its security level against standard attacks on the McEliece cryptosystem. Liva and Bartz enhanced the codes used in [8] by proposing other QC-MDPC code ensembles with better error-correction capabilities. The work done in these papers were studied and implemented, and their results were reproduced for validation. The system built for reproducing these results, will lay the foundation to test and analyze the required spatially-coupled codes.

1.2 Thesis Outline

The rest of the thesis report is divided as following. The second chapter introduces the state-of-the-art public-key cryptosystems along with their mathematical backhround, and also introduces the McEliece cryptosystem. The third chapter introduces and explains some modern coding theory concepts that will aid in understanding the topic and the work done in the thesis. The fourth chapter introduces and explains the work done in papers [8] and [10], where QC-MDPC codes are used as variants for the McEliece cryptosystem. The fifth chapter introduces and ex-

plains an attack designed to break the cryptosystem explained in chapter four, in addition to introducing modifications for these codes to counter-act the attack. The sixth chapter introduces the proposed spatially-coupled codes of this thesis. The seventh chapter gives some numerical results. Finally, the eighth chapter concludes and gives the required future work for the project and the topic.

2

Public-Key Cryptosystems

2.1 State-of-the-Art Public-Key Cryptosystems

Any application or system that requires using a public-key cryptosystem needs to include three main functions. These three functions are 1) Encryption/decryption, 2) Digital signature, and finally 3) Key exchange [6].

As the name suggests, the Encryption/decryption function needs to successfully encrypt the message with the public key, and successfully decrypt it with the private key. Furthermore, a digital signature on the message can be achieved by using the private key. Since, only the intended user will have access to the private key, only they can encrypt and decrypt messages with it; hence, authenticating (signing) their message. Finally, the key-exchange function helps in establishing a suitable private key between the sender and receiver [6]. Different public-key cryptography algorithms provide different functions, some can only be used for one function, and some are suitable for all. The state of the art public-key algorithms/cryptosystems include Diffie-Helman key exchange introduced by Whitfield Diffie and Martin Hellman in 1976 [5], El Gamal (DSS) cryptosystem introduced by Taher Elgamal in 1985 [11], RSA cryptosystem introduced by Rivest, Shamir and Adleman in 1978 [12], and finally the Elliptic Curve Cryptography (ECC) introduced by Victor Miller in 1985 [13] and by Neal Koblitz in 1987 [14]. Table 2.1 shows what functions are suitable for each public-key cryptosystem [6].

Algorithms	Encryption/Decryption	Key Exchange	Digital Signature
Diffie-Hellman	No	Yes	No
DSS	No	No	Yes
RSA	Yes	Yes	Yes
ECC	Yes	Yes	Yes

Table 2.1: Suitable functions for each public-key cryptosystem/algorithm

These algorithms have several conditions and requirements that need to be met to be used in practice. Furthermore, only these algorithms satisfy all these requirements, where many other developed algorithms failed to meet them [6]. These requirements are as following [6]:

1) It should be computationally easy to generate the public and private key.
2) It should be computationally easy to generate a cyphertext from a message knowing the public key. In other words, encryption should be computationally easy.
3) It should be computationally easy to recover the original message from the received cyphertext knowing the private key. In other words, decryption should be computationally easy.
4) It should be computationally very difficult to recover the private key knowing the public key.
5) It should be computationally very difficult to recover the original message knowing only the public key and the received cyphertext.

2.1.1 RSA Cryptosystem

Since the RSA cryptosystem is suitable for all the mentioned functions, it is the most commonly used out of all the other public-key cryptosystems. The message in the RSA cryptosystem is encrypted in several blocks, where each block is of size n in bits. Moreover, for an original message m and cyphertext C, the message encryption and decryption are done as following:

$$\begin{aligned} C &= m^e \mod n \\ m &= C^d \mod n \;=\; m^{e \times d} \mod n \end{aligned}$$

Where e, d and n are integers. Both the source and the receiver know the value of n. Furthermore, the source also knows the value of e, while only the receiver knows the value of d. Consequently, the public key is $\{e, n\}$ and the private key is $\{d, n\}$ for this cryptosystem. The block length n is calculated by privately choosing two prime numbers p and q and multiplying them, i.e. $n = p \times q$. Then, e and d, are chosen such that they satisfy the following equation:

$$ed = 1 \mod (p-1)(q-1)$$

An attacker will need to factor n into its prime factors p and q in order to retrieve the original message in reasonable time. As result, the security of the RSA cryptosystem relies on the difficulty of performing prime factorization, which is know to be extremely difficult [6].

2.1.2 Other Public-Key Cryptosystems

Unlike the RSA cryptosystem where its security relies on the difficulty of prime factorization, both the Deffie-Hellman and El Gamal algorithms' security rely on the difficulty of calculating discrete logarithms which will be shown later [6]. For both these algorithms, there are two integers that are accessible to everyone, there is the prime number p, and an integer α which is the primitive root of p.

Now for the Deffie-Hellman algorithm specifically, a key between users A and B is exchanged by the following steps. First, user A randomly selects an integer $X_A < p$ which is kept secret, and computes its public value $Y_A = \alpha^{X_A} \mod p$ and sends it to B. Meanwhile, user B also randomly selects a secret integer $X_B < p$, and computes its public value $Y_B = \alpha^{X_B} \mod p$ and sends it to A. Afterwards, user A computes the key by $K = (Y_B)^{X_A} \mod p$, and B computes the same key by $K = (Y_A)^{X_B} \mod p$, since both calculations lead to the same result. Hence, for this algorithms the public key is p, α, and Y_A and Y_B, and the private key is X_A and X_B.

For El Gamal algorithm, almost the same procedure is used, but in this case also an encrypted message is sent. User A generates random integer X_A, where $1 < X_A < p - 1$, then computes $Y_A = \alpha^{X_A} \mod p$. User A will have a public key $\{p, \alpha, Y_A\}$, and private key X_A. Any user B who knows user A's public key can send them an encrypted message m as following. First, user B selects a random integer k, where $1 \leq k \leq p - 1$. Then, the key is computed by $K = (Y_A)^k \mod p$, and the message m is encrypted with a pair of integers C_1 and C_2, where $C_1 = \alpha^k \mod p$ and $C_2 = K \times m \mod p$. Finally, user A can retrieve the original message by first computing the key by $K = (C_1)^{X_A} \mod p$, and computing the message as $m = (C_2 \times K^{-1}) \mod p$.

An attacker will need to recover the key K or user A's private key to break the cryptosystem. As a result, the attacker will either need to compute the discrete logarithm of Y_A for the first case or compute the discrete logarithm of C_1 for the second case. Both of these computations are known to be infeasible for a good choice of p [6].

Finally, the ECC algorithm was introduced to mitigate the problems that aroused from RSA cryptosystem in recent years. The required key size for a given level of security for RSA cryptosystem increased in recent years, which lead to shifting interest to the ECC cryptosystem. Since the ECC cryptosystem can achieve the same required level of security for a lower key size compared to the RSA cryptosystem [6].

2.2 McEliece Cryptosystem

As mentioned earlier, all public-key cryptosystems will no longer be safe with the introduction of quantum computers in the future. Since a quantum computer attack using shor's algorithm can break these cryptosystems easily. One cryptosystem that is known to be safe from any attack that uses shor's algorithm is the McEliece cryptosystem [15]. This cryptosystem was introduced by Robert McEliece in 1978 [6], and all of the post-quantum cryptosystems that uses error-correction codes are based on it.

The McEliece cryptosystem is a public key cryptosystem, with its own public and private keys. The cryptosystem provides an encryption and decryption algorithm that is based on the hardness of decoding a liner code $\mathcal{C}$. McEliece proposed the use

of Goppa codes as the linear code $\mathcal{C}$.

2.2.1 Coding Theory Preliminaries

Before going into more detail on the McEliece cryptosystem, some few concepts and definitions of coding theory need to be introduced. Coding theory in the context of communication theory, is used to add redundancy to the message sent over the noisy communication channel, where this redundancy can be exploited at the receiver to minimize the introduced errors [16]. In other words, the added redundancy will average out the noise; hence, reducing the probability of message error. The message will be encoded into a codeword by a code, then sent over the channel where it will be received and decoded. An error correcting code with code length n and dimension k, denoted as $\mathcal{C}(n,k)$, consists of 2^k codewords each of length n, that are binary tuples, all existing in binary field $\mathbb{F}_2^n$.

Definition (Linear Block Code): A code is said to be linear if all its 2^k codewords are a subspace of the binary field $\mathbb{F}_2^n$.

An encoder for $\mathcal{C}(n,k)$ will take a message m and encoded into one of the codewords of $\mathcal{C}(n,k)$, using an $n \times k$ generator matrix $\boldsymbol{G}$. $\boldsymbol{G}$ consists of k linearly independent vectors that span $\mathcal{C}(n,k)$, and a message of length k, can be encoded into a length n codeword as following, $\boldsymbol{c} = \boldsymbol{mG}$. The error correction capability of a code $\mathcal{C}$ is denoted by t, and it refers to the maximum number of bit errors $\mathcal{C}$ is guaranteed to correct.

All linear block codes $\mathcal{C}$ have a dual (null) space $\mathcal{C}_{\mathsf{null}}$ that is a $(n-k)$ dimensional subspace of field $\mathbb{F}_2^n$. The generator matrix of $\mathcal{C}_{\mathsf{null}}$ is an $(n-k) \times n$ matrix denoted by $\boldsymbol{H}$, and referred to as the parity-check matrix of code $\mathcal{C}$. Most importantly, if a codeword $\boldsymbol{c}$ or generator matrix $\boldsymbol{G}$ of code $\mathcal{C}$ are multiplied by the transpose of $\boldsymbol{H}$, we get the all zero vector and all zero matrix respectively. That is, $\boldsymbol{cH}^T = \boldsymbol{0}$, and $\boldsymbol{GH}^T = \boldsymbol{0}_{k\times k}$

Definition (Quasi-Cyclic Code): A linear block code is said to be quasi-cyclic (QC) if any cyclic shift of a codeword $\mathcal{C}$ by a positive integer n_0 is another codeword. The generator matrix of a QC-code will have the following form:

$$\boldsymbol{G}_{\mathrm{QC}} = \begin{pmatrix} \boldsymbol{P}_0 & \boldsymbol{P}_1 & \dots & \boldsymbol{P}_{n_0-1} \end{pmatrix}$$

where each $\boldsymbol{P}_i$, is a cyclic matrix, which means each row is cyclically shifted one unit to the right compared to the previous one. The corresponding $\boldsymbol{H}$ matrix will have the same format.

2.2.2 McEliece Algorithm Description

This section includes a more detailed description of the encryption and decryption algorithm introduced by Robert McEliece, where the key generation and the message encryption and decryption procedures are explained.

The message encryption of the McEliece cryptosystem is done by encoding the required message with a scrambled and permuted version of the code $\mathcal{C}$ generator matrix $\boldsymbol{G}$. In addition, the code $\mathcal{C}$ should have a well known and efficient decoding algorithm for the cryptosystem to work well as will be shown later. Furthermore, what the algorithm does is that it introduces intentional random errors in the encoded message, then sends the message to the decoder. The decoder, which also performs the decryption, will try to correct the errors, introduced in the encryption phase, using the known decoding algorithm of the code $\mathcal{C}$. If the decoding algorithm is not known, the message decoding can only be done by brute-force searching of the codeword from $\boldsymbol{G}$ that is closest in hamming weight to the received message. For codes with large dimensions, this process cannot be performed in reasonable time [9].

2.2.2.1 Key Generation

The public and private keys of this cryptosystem are generated as following. First, select an (n, k) binary linear code $\mathcal{C}$, with generator matrix $\boldsymbol{G}$, and error correction capability t. Second, select a random $n \times n$ matrix $\boldsymbol{P}$ and $k \times k$ matrix $\boldsymbol{S}$, where $\boldsymbol{S}$ is non-singular scrambler matrix, and $\boldsymbol{P}$ is a permutation matrix. Then compute the scrambled and permuted generator matrix $\hat{\boldsymbol{G}} = \boldsymbol{S}\hat{\boldsymbol{G}}\boldsymbol{P}$. Consequently, the public key is $(\boldsymbol{G}, t)$, and the private key is $(\boldsymbol{S}, \hat{\boldsymbol{G}}, \boldsymbol{P})$.

2.2.2.2 Message encryption

The message encryption is done int two steps. The first step is to encode the message $\boldsymbol{m}$ into codeword $\boldsymbol{c}'$ using the public key, i.e. $\boldsymbol{c}' = \boldsymbol{m}\hat{\boldsymbol{G}}$. Then a random error sequence $\boldsymbol{e}$ of weight$(\boldsymbol{e}) \leq t$ is generated and added to $\boldsymbol{c}'$ to produce $\boldsymbol{c}$, i.e. $\boldsymbol{c} = \boldsymbol{c}' + \boldsymbol{e}$. In this step, the algorithm adds intentional errors by adding the error sequence to the encoded message, then sends $\boldsymbol{c}$ to the decoder. This way of introducing errors can be modeled as a binary symmetric channel (BSC), that always introduces t or less errors, rather than flipping the message bit with a probability equal to the channel cross-over probability. The errors introduced in the encoded message can be corrected in a reasonable time using a decoding algorithm of code $\mathcal{C}$. Doing the following will guarantee $t-$bit level security against any attack that tries to decode the received message without knowing the decoding algorithm.

2.2.2.3 Message Decryption

The message decryption is done in two steps as well, and it can be only done by knowing the private key mentioned earlier. The first step is to compute $\hat{\boldsymbol{c}} = \boldsymbol{c}\boldsymbol{P}^{-1}$, and decode $\hat{\boldsymbol{c}}$ into $\hat{\boldsymbol{m}}$ using the decoding algorithm. The second step is to retrieve the original message as $\boldsymbol{m} = \hat{\boldsymbol{m}}\boldsymbol{S}^{-1}$.

2.2.3 Attacks on the McEliece Cryptosystem

General attacks on the McEliece cryptosystem fall under two types, message attacks, and key attacks [8] [9]. Message attacks try to recover the original message from the encoded and distorted version of it. The only way to perform the message attack on the McEliece cryptosystem is to perform a decoding attack on the message, where the attacker will try to decode a massage with t errors from linear code $\mathcal{C}$ [8]. Key attacks on the other hand, try to recover the private key instead of the message. Performing a key attack can be done by either performing a key distinguishing or a key recovery attack. In the key distinguishing attack, the attacker tries to distinguish the generator matrix $\boldsymbol{G}$ from the public key $\hat{\boldsymbol{G}}$ [9]. If the attacker manages to produce a codeword of weight w from the the dual code $\mathcal{C}_{\text{null}}$, where w is the row weight of $\mathcal{C}_{\text{null}}$, then this attack is considered successful [8]. Moreover, in the key recovery attack, the attacker tries to recover all parity check equations of low weight from an equivalent private key [8].

Since the message attacks rely on decoding a message with t errors, the security of the McEliece cryptosystem against these attacks depend on the $t-$bit security level guaranteed from $\mathcal{C}$. The higher the error-correction capability t of $\mathcal{C}$, the better the security from a message attack. Furthermore, the security of the cryptosystem against the key attacks depends on the different number of dual codes $\mathcal{C}_{\text{null}}$ that can be generated from $\mathcal{C}$. The bigger the space of dual codes $\mathcal{C}_{\text{null}}$, the better the security against a key attack [8].

The best known technique to perform any of the above attacks uses information set decoding (ISD) procedure [8]. The computational cost of performing any of the attacks using ISD is denoted by the work factor WF_{ISD}, where it computes the bit combinations required to be performed.

2.2.4 Advantages and Disadvantages of the McEliece Cryptosystem

There are two main advantages of the McEliece cryptosystem that can be deduced. The first is that encryption and decryption implementation can be performed fast and efficiently, since fast decoding algorithms exist for Goppa codes [9]. The second advantage is that performing the attacks mentioned earlier on Goppa codes is extremely difficult [8]. This is due to the fact that distinguishing $\hat{\boldsymbol{G}}$ from $\boldsymbol{G}$ is almost impossible, because of all the possible $\boldsymbol{G}$, $\boldsymbol{S}$ and $\boldsymbol{P}$ matrices that attacker has to go

through [9]. In addition, the problem of decoding a message with t from a linear code is proved to be $NP-$complete [9]. However, using Goppa codes results in very large key sizes, and there exists a high rate Goppa code distinguisher, which can be used to make the attack on the system easier [8].

As a result, the focus of the current research on code-based post-quantum cryptography is to find codes with good error correction capabilities that can reduce the key sizes as much as possible, dropping the need for using Goppa codes, while still maintaining an acceptable security level.

3

Basics of Modern Coding Theory

This chapter introduces and explains few concepts in modern coding theory that are needed for the rest of the report. First, low density parity check (LDPC) codes are introduced. Then, the decoding algorithms used for LDPC codes along with their density evolution (DE) equations are introduced and explained. Finally, spatially-coupled (SC) are introduced and explained as well.

3.1 LDPC Codes

Low Density Parity Check (LDPC) codes are commonly used in modern coding theory and code based systems. The main concept behind LDPC codes is their sparse parity-check matrix $\boldsymbol{H}$, which means that all rows have at least J ones (entries) where J is a small integer [17]. Since $\boldsymbol{H}$ is sparse, it can be easily represented as a bipartite (tanner) graph from graph theory, where this graph consists of $(n-k)$ check-nodes (CNs); corresponding to the rows in $\boldsymbol{H}$, and n variable-nodes (VNs); corresponding to the columns of $\boldsymbol{H}$. A VN i will be connected to a CN j in this graph via an edge, if there exists an entry (a one) between row j and column i in $\boldsymbol{H}$. Figure 3.1 shows a parity-check matrix for a (7×3) Hamming code, $\boldsymbol{H}_{\text{Hamming }(7,3)}$, with its corresponding tanner graph for visualization. The sparse $\boldsymbol{H}$ is mostly used in decoding, where several decoding algorithms exploit the VNs and CNs connections to iteratively pass reliabilities of the message received between them, until the message is recovered successfully or a maximum number of iteration has been reached.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

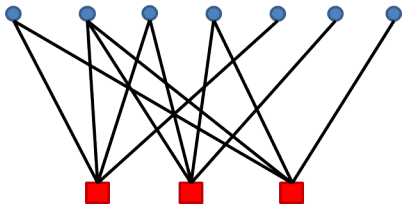

Figure 3.1: Parity-Check matrix $\boldsymbol{H}$ of $(7, 3)$ Hamming code with its corresponding Tanner graph.

An LDPC code is said to be regular, if all rows have the same number of ones

J, and all columns have the same number of ones K as well. The number of ones that the rows and columns in a regular code have is referred to as the CNs degree and VNs degree, denoted by d_{c} and d_{v} respectively.

3.2 Decoding Algorithms for LDPC Codes

In this section, all the equations used in performing the decoding procedure for each decoding algorithm are introduced. The decoding algorithms include Gallager A, B, algorithm E, and SPA. All the LDPC codes used, and hence the decoders, are assumed to operate under the BSC channel model.

3.2.1 Gallager A

The values that the passed messages have in Gallager A are either -1, or a +1. The message value coming from the channel is denoted by m_{ch}. The message value passed from a VN to a neighboring CN is denoted by $m_{\mathrm{v}\to\mathrm{c}}$, while that from a CN to a neighboring VN is denoted by $m_{\mathrm{c}\to\mathrm{v}}$. The cyphertext bit is denoted by c_{b} , and for Gallager A, the cyphertext bit can either be a -1, or a +1. The Gallager A decoding algorithm works as following [16]:

Algorithm 1: Gallager A Decoding Algorithm

1. **Initialization:** For all VNs, initialize all the VNs to CNs messages with their corresponding channel message value (cyphertext bit value -1,+1), i.e. $m_{\mathrm{v}\to\mathrm{c}} = m_{\mathrm{ch}}$.

2. **CN update:** For all CNs, the CN to VN message becomes:

$$m_{\mathrm{c}\to\mathrm{v}} = \prod_{v' \in N(c) \neq v} m_{\mathrm{v}'\to\mathrm{c}} \,,$$

3. **VN update:** A VN v to CN c message will change its original cyphertext value if all its neighboring CNs ($CN \neq c$) disagree with it. Otherwise, the message value will stay the same. i.e:

$$m_{\mathrm{v}\to\mathrm{c}} = \begin{cases} -c_{\mathrm{b}} & \text{if } m_{\mathrm{c}'\to\mathrm{v}} = -c_{\mathrm{b}} \text{ for } c' \in N(v) \neq c \\ c_{\mathrm{b}} & \text{otherwise.} \end{cases}$$

4. **Final Bit Decision:** For Gallager A the final bit decision is a majority based decision. If the degree of the VN is odd, then the bit value is the majority of the neighboring CN messages. If the VN degree is even, the the bit value is the majority of CN messages plus the channel message. i.e:

$$\hat{c} = \begin{cases} \text{majority of } m_{\mathrm{c}\to\mathrm{v}} \text{ if } d_v \text{ is odd} \\ \text{majority of } m_{\mathrm{c}\to\mathrm{v}} + c_{\mathrm{b}} \text{ if } d_v \text{ is even.} \end{cases}$$

5. **Stopping Criteria:** If $\hat{c}\,\boldsymbol{H}^T = \mathbf{0}$ i.e. a valid codeword was obtained, or maximum number of iterations I_{max} has been reached, exit the decoder; otherwise, go back to step 2 for another iteration.

3.2.2 Gallager B

Similar to Gallager A, The values that the passed messages have in Gallager A are either -1, or a +1. Hence, the same notation is used. The Gallager B decoding algorithm is as following:

Algorithm 2: Gallager B Decoding Algorithm

1. **Initialization:** For all VNs, initialize all the VNs to CNs messages with their corresponding channel message value (cyphertext bit value -1,+1), i.e. $m_{\mathrm{v}\to\mathrm{c}} = m_{\mathrm{ch}}$.

2. **CN update:** For all CNs, the CN to VN message becomes:

$$m_{\mathrm{c}\to\mathrm{v}} = \prod_{v' \in N(c) \neq v} m_{\mathrm{v}'\to\mathrm{c}} \,,$$

3. **VN update:** A VN v to CN $\mathcal{C}$ message will change its original cyphertext value if the number of neighboring CNs ($CN \neq c$) that disagree with it exceeds a threshold α. Otherwise, the message value will stay the same. i.e:

$$m_{\mathrm{v}\to\mathrm{c}} = \begin{cases} -c_{\mathrm{b}} & \text{if } |\{c' \in N(v) \neq c \,:\, m_{c'\to v} = -c_{\mathrm{b}}\}| \geq \alpha \\ c_{\mathrm{b}} & \text{otherwise.} \end{cases}$$

4. **Final Bit Decision:** The cyphertext value of a bit will flip if the number of incoming messages from all neighboring CNs that disagree with it exceeds a threshold α. i.e:

$$\hat{c} = \begin{cases} -c_{\mathrm{b}} & \text{if } |\{c \in N(v) \,:\, m_{\mathrm{c}\to\mathrm{v}} = -c_{\mathrm{b}}\}| \geq \alpha \\ c_{\mathrm{b}} & \text{otherwise.} \end{cases}$$

5. **Stopping Criteria:** If $\hat{c}\,\boldsymbol{H}^T = \mathbf{0}$ i.e. a valid codeword was obtained, or maximum number of iterations I_{max} has been reached, exit the decoder; otherwise, go back to step 2 for another iteration.

3.2.3 Algorithm E

The values that the passed messages have in Algorithm E are -1, 0, or a +1; where a zero represents an erasure. This algorithm can be used on codes constructed from protographs with state VNs as will be shown later.

Algorithm 3: Gallager E Decoding Algorithm

1. **Initialization:** For all VNs, initialize all the VNs to CNs messages with their corresponding channel message value, i.e. $m_{\mathrm{v}\to\mathrm{c}} = m_{\mathrm{ch}}$. The channel message will be the corresponding cyphertext bit value $m_{\mathrm{ch}} = c_{\mathrm{b}}$ for a normal VN, whereas it will be equal to an erasure $m_{\mathrm{ch}} = 0$, for a state VN.

2. **CN update:** For all CNs, the CN to VN message becomes:

$$m_{\mathrm{c}\to\mathrm{v}} = \prod_{v' \in N(c) \neq v} m_{\mathrm{v}'\to\mathrm{c}} \,,$$

3. **VN update:** For a scaling factor ω, the VN to CN message becomes:

$$m_{\mathrm{v}\to\mathrm{c}} = \mathrm{sign}\left[\omega m_{\mathrm{ch}} + \sum_{c' \in N(v) \neq c} m_{\mathrm{c}'\to\mathrm{v}}\right]$$

4. **Final Bit Decision:** The final bit decision is calculated as following:

$$\hat{c} = \mathrm{sign}\left[m_{\mathrm{ch}} + \sum_{c \in N(v)} m_{\mathrm{c}\to\mathrm{v}}\right]$$

where the dependence on the scaling factor is dropped.

5. **Stopping Criteria:** If $\hat{c}\,\boldsymbol{H}^T = \mathbf{0}$ i.e. a valid codeword was obtained, or maximum number of iterations I_{max} has been reached, exit the decoder; otherwise, go back to step 2 for another iteration.

3.2.4 SPA (Belief Propagation)

The passed messages can have any value in SPA, so the decoding algorithm becomes:

Algorithm 4: Gallager Sum-Product Algorithm for Decoding

1. **Initialization:** Initialize all the channel messages values as $m_{\text{ch}} = c_{\text{b}} \ln \frac{n-e}{e}$, where n is the code length, and e is the weight of the error vector pattern. Then for all VNs, set $m_{\text{v}\to\text{c}} = m_{\text{ch}}$.

2. **CN update:** For all CNs and for a scaling factor ω, the CN to VN message becomes:

$$m_{\text{c}\to\text{v}} = \omega 2 \tanh^{-1} \left[\prod_{v' \in N(c) \neq v} \tanh\left(\frac{m_{\text{v}'\to\text{c}}}{2}\right) \right],$$

3. **VN update:** For a scaling factor ω, the VN to CN message becomes:

$$m_{\text{v}\to\text{c}} = m_{\text{ch}} + \sum_{c' \in N(v) \neq c} m_{\text{c}'\to\text{v}}$$

4. **Final Bit Decision:** The final bit decision is calculated as following:

$$\hat{c} = \text{sign} \left[m_{\text{ch}} + \sum_{c \in N(v)} m_{\text{c}\to\text{v}} \right]$$

5. **Stopping Criteria:** If $\hat{c}\, \boldsymbol{H}^T = \mathbf{0}$ i.e. a valid codeword was obtained, or maximum number of iterations I_{max} has been reached, exit the decoder; otherwise, go back to step 2 for another iteration.

3.3 Density Evolution of Message Probability in LDPC Codes

One powerful tool to predict the performance and error correction capability of an LDPC code is referred to as Density Evolution (DE). DE measures and tracks the probability density function (pdf) of the messages being passed iteratively in the decoder, where the messages are modeled as random variables [16]. DE can predict at which channel conditions can the code best operate at. It should also be noted that DE measures the average performance of the ensemble that this code belongs to, and according to coding theory, for a large code length n, all the codes in the ensemble will approach this ensemble average [16].

A code ensemble can be represented by a base protograph, which is a condensed version of a tanner graph of an $\boldsymbol{H}$ matrix. The ensemble/protograph can be fully represented by the number of rows and columns its corresponding base matrix has, and the degree of each row and column. The columns of a base protograph are referred to as VN types, and its rows as CN types. Taking the simplest ensemble, the $(3, 6)$ LDPC code ensemble, as an example, the base protograph matrix and the

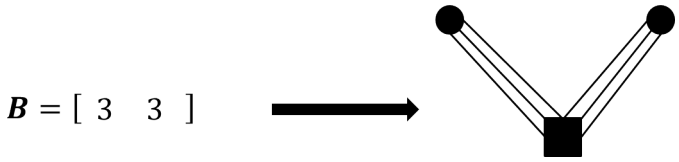

Figure 3.2: Base protograph and base protograph matrix of the (3,6) LDPC code ensemble.

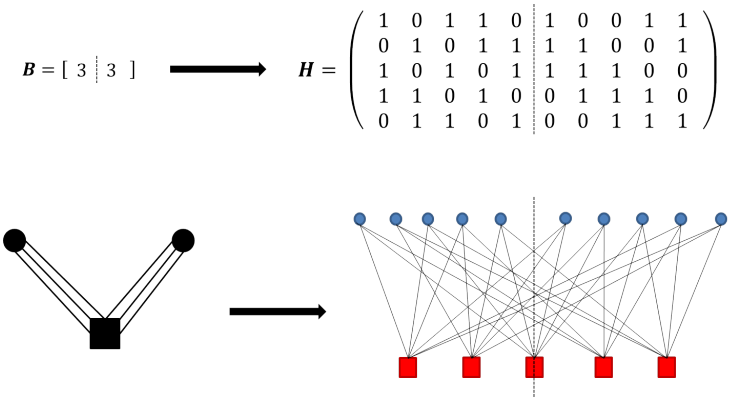

Figure 3.3: Base protograph and base protograph matrix expansion of a QC-(3,6) LDPC code.

base protograph are show in Figure 3.2.

Each VN type and CN type can be expanded to include several matrices inside them, and eventually, the entire base matrix can be expanded into the corresponding $\boldsymbol{H}$ matrix of the required code [16]. For a QC code, each entry in the base protograph matrix will be replaced by a circulant, and several constraints can be placed on these circulants to create the desired code $\mathcal{C}$. If it is required to create a QC Code from the the $(3, 6)$ LDPC code ensemble, the expansion will be as following: DE will track the evolution of the messages pdfs along the edges of these base protographs. In addition, the way the messages pdfs evolve are dictated by several equations that update the VNs and CNs messages probability values for each iteration in the decoding process.

DE on any code ensemble can be implemented using the following algorithm as a backbone [16]. The parameters of the algorithm are as following: β is the channel parameter (cross-over probability, erasure probability etc.), l is the iteration counter, $\boldsymbol{m}$ is the channel message probability, $\boldsymbol{p}$ is the VN to CN message probability, $\boldsymbol{q}$ is the CN to VN message probability, OS is the number of values a message can have (this is dependant on the decoding algorithm used), and finally p_{min} is a predetermined minimum probability.

Algorithm 5: Density Evolution Implementation

Assuming that the message and channel probabilities will only have two output states (OS) -1 , +1, which is the case for Gallager A and B.

1. Choose a value for β that is known to be lower than the threshold β^*. Set the iteration counter $l = 0$.
2. Set the channel probability value as $m_{\{-1,+1\}} = \{\beta, 1-\beta\}$.
3. Set the initial CN and VN message probabilities (iteration 0), $\boldsymbol{q}^{(0)} = \boldsymbol{m}$ and $\boldsymbol{p}^{(0)} = \boldsymbol{m}$ for each OS.
4. Compute $\boldsymbol{q}^{(l)}$ and $\boldsymbol{p}^{(l)}$ using the corresponding update equation for the decoding algorithm used.
5. Check which condition is satisfied:

if $l < l_{max}$ *and* $p_{-1}^{(l)} \leq p_{min}$ **then**
  Increment β by a small amount δ and go to step 2.
else if $l < l_{max}$ *and* $p_{-1}^{(l)} > p_{min}$ **then**
  Increment l and go to step 3.
else if $l = l_{max}$ *and* $p_{-1}^{(l)} > p_{min}$ **then**
  The most recent β before incrimination, is the decoding threshold β^*.
else
  exit.
end

Algorithm 1 will be altered depending on the decoding algorithm used, and whether the code ensemble being analyzed is regular or irregular.

3.3.1 Density Evolution for Gallager A

The output states available for the messages in Gallager A are -1 and +1, so no alteration is required for the number of output states in Algorithm 1. However, some notation alteration for the CN and VN update equations is required when moving from regular to irregular ensembles.

For regular ensembles, all CN types will follow the same update equation:

$$\left(q_{-1}^{(l)}, q_{+1}^{(l)} \right) = \frac{1}{2}\left(1 - \left(1 - 2p_{-1}^{(l-1)}\right)^{dc-1},\ 1 + \left(1 - 2p_{-1}^{(l-1)}\right)^{dc-1}\right)$$

and all VN types will have the following update equation:

$$\left(p_{-1}^{(l)}, p_{+1}^{(l)} \right) = \left(p_{+1}^{(0)}\left(q_{-1}^{(l)}\right)^{dv-1} + p_{-1}^{(0)}\left(1 - \left(q_{+1}^{(l)}\right)^{dv-1}\right),\right.$$

$$\left. p_{-1}^{(0)}\left(q_{+1}^{(l)}\right)^{dv-1} + p_{+1}^{(0)}\left(1 - \left(q_{-1}^{(l)}\right)^{dv-1}\right)\right)$$

For the irregular case, each VN type and CN type will have a different equation compared to the other VN and CN types. Furthermore, only the -1 output state is tracked in the update equations for simplicity. Denote by $p_{-1}^{(l)}(i,j)$ the probability that the message from VN type i to CN type j has a value of -1. Similarly, denote by $q_{-1}^{(l)}(i,j)$ the same for a message from CN type i to VN type j. Finally, denote by n_{v} and n_{c} be the number of VN types and CN types available in the photograph. The CN type and VN type update equations become:

$$q_{-1}^{(l)}(i,j) = \frac{1}{2}\left(1 - \prod_{\substack{m=1 \\ b_{i,m}\neq 0}}^{n_v} (1-2p_{-1}^{(l-1)}(m,i))^{b_{i,m}-1\{m=j\}}\right)$$

and

$$\begin{aligned} p_{-1}^{(l)}(i,j) = (1-\epsilon) & \prod_{\substack{m=1 \\ b_{i,m}\neq 0}}^{n_c} (q_{-1}^{(l)}(m,i))^{b_{i,m}-1\{m=j\}} \\ & + \epsilon \left(1 - \prod_{\substack{m=1 \\ b_{i,m}\neq 0}}^{n_c} (q_{+1}^{(l)}(m,i))^{b_{i,m}-1\{m=j\}}\right) \end{aligned}$$

where $\epsilon = m_{-1}$, and $b_{i,j}$ is the base protograph matrix value at edge position (i,j).

3.3.2 Density Evolution for Gallager B

Gallager B has the same output states as Gallager A, so the CN types update equations follow the same procedure:

$$\left(q_{-1}^{(l)}, q_{+1}^{(l)}\right) = \frac{1}{2}\left(1 - \left(1-2p_{-1}^{(l-1)}\right)^{dc-1},\ 1 + \left(1-2p_{-1}^{(l-1)}\right)^{dc-1}\right)$$

and all VN types will have the following update equation:

$$p_{-1}^{(l)} = \epsilon - \epsilon \sum_{k=\alpha^{(l)}}^{d_v-1} \binom{d_v-1}{k} (q_{+1}^{(l)})^k\, (q_{-1}^{(l)})^{d_v-1-k} + (1-\epsilon) \sum_{k=\alpha^{(l)}}^{d_v-1} \binom{d_v-1}{k} (q_{-1}^{(l)})^k\, (q_{+1}^{(l)})^{d_v-1-k}$$

where $\epsilon = m_{-1}$, α is an integer parameter special to Gallager B. However, an extra equation is required for the optimal value of α to be used in current iteration l. The optimal value of α, is the smallest integer that satisfies this equation:

$$\frac{1-p_{-1}^{(0)}}{p_{-1}^{(0)}} \leq \left[\frac{1+\left(1-2p_{-1}^{(l-1)}\right)^{dc-1}}{1-\left(1-2p_{-1}^{(l-1)}\right)^{dc-1}}\right]^{2\alpha-dv+1}$$

The same new notation is used for irregular code ensembles in Gallager B. The CN types update equation is the same as in irregular Gallager A, and the VN types update equation is as following:

$$p_{-1}^{(\ell)}(i,j) =$$
$$(1-\epsilon)\sum_{\alpha=\alpha_{i,j}^{(\ell)}}^{d_{v_i}-1}\sum_{\boldsymbol{v}:\sum_{k=1}^{n_c}v_k=\alpha}\prod_{\substack{m=1\\ b_{m,i}\neq 0}}^{n_c}\binom{b_{m,i}-1\{m=j\}}{v_m}\left(q_{-1}^{(\ell)}(m,i)\right)^{v_m}\left(q_1^{(\ell)}(m,i)\right)^{b_{m,i}-1\{m=j\}-v_m}$$
$$+\epsilon\left(1-\sum_{\alpha=\alpha_{i,j}^{(\ell)}}^{d_{v_i}-1}\sum_{\boldsymbol{v}:\sum_{k=1}^{n_c}v_k=\alpha}\prod_{\substack{m=1\\ b_{m,i}\neq 0}}^{n_c}\binom{b_{m,i}-1\{m=j\}}{v_m}\left(q_1^{(\ell)}(m,i)\right)^{v_m}\left(q_{-1}^{(\ell)}(m,i)\right)^{b_{m,i}-1\{m=j\}-v_m}\right)$$

where $\alpha_{i,j}^{(l)}$ is the threshold value at edge position (i,j) and iteration l, $b_{i,j}$ is defined earlier, and $\boldsymbol{v}=(v_1,\ldots,v_{n_c})$ with v_m being the number of incoming messages, from a CN type at position m sent to a VN type at position i, that disagree with the channel message.

3.3.3 Density Evolution for Algorithm E

One major alteration in DE implementation for algorithm E, is the output states for the CNs/VNs and channel message probabilities. Each CN and VN message can now have three possible values, -1, 0, and +1, where 0 represents an erasure. In addition, as will be seen later, the VN to CN update message in the decoding process, is multiplied by a scaling factor ω; as a result, the channel message will now have $2\omega+1$ output states. So, the channel message will be initialised as:

$$\boldsymbol{m}_{\{-\omega,-\omega+1,\ldots,+\omega-1,+\omega\}}=\{\boldsymbol{\beta},0,...,0,1-\boldsymbol{\beta}\}.$$

where $p_{-1}^{(0)}=m_{-\omega}$, and $p_{+1}^{(0)}=m_{+\omega}$.

The CNs update equations for all output states for regular codes will be as following:

$$q_{+1}^{(l)}=\frac{1}{2}\left[\left(p_{+1}^{(l-1)}+p_{-1}^{(l-1)}\right)^{d_c-1}+\left(p_{+1}^{(l-1)}+p_{-1}^{(l-1)}\right)^{d_c-1}\right].$$
$$q_{-1}^{(l)}=\frac{1}{2}\left[\left(p_{+1}^{(l-1)}+p_{-1}^{(l-1)}\right)^{d_c-1}-\left(p_{+1}^{(l-1)}+p_{-1}^{(l-1)}\right)^{d_c-1}\right].$$
$$q_0^{(l)}=1-\left(1-p_0^{(l-1)}\right)^{(d_c-1)}.$$

New notations for the CNs/VNs and channel message probabilities need to be introduced for the VNs update equations. Each probability will now be a vector, in which each element of that vector is the singular output state probability. The notation for the VNs, CNs, and channel message probabilities is as following respectively:

$$\boldsymbol{p}^{(l)}=\left(p_{-1}^{(l)},p_{+1}^{(l)},p_0^{(l)}\right)\qquad \boldsymbol{q}^{(l)}=\left(q_{-1}^{(l)},q_{+1}^{(l)},q_0^{(l)}\right)\qquad \boldsymbol{m}=(m_{-\omega},m_{-\omega+1},\ldots,m_{+\omega})$$

where the channel message probabilities values are the same as in the initialization earlier. Define an intermediate probability vector $\boldsymbol{z}^{(l)} = \left[\circledast_{d_c-1} q_{-1}^{(l)}(i)\right] \circledast \boldsymbol{m}$. In other words, the intermediate probability vector at iteration l, is the convolution of all CNs message probability vectors from $i = 0$ to $d_c - 1$, convolved with the channel message probability vector. Then the VNs message probability vector can be calculated as following:

$$p_{-1}^{(l)} = \sum_{n<0} z_n^{(l)} \qquad p_0^{(l)} = z_0^{(l)} \qquad p_{+1}^{(l)} = \sum_{n>0} z_n^{(l)}.$$

For irregular code ensembles, the same notation is used as in Gallagher A and B to differentiate between each VN type and CN type update equation. The CN types update equations, for all output states are as following:

$$\begin{aligned}
q_{-1}^{(l)}(i,j) &= \frac{1}{2}\left(\prod_{\substack{m=1\\ b_{i,m}\neq 0}}^{n_v} \left(p_{+1}^{(l-1)}(m,i) + p_{-1}^{(l-1)}(m,i)\right)^{b_{i,m}-1\{m=j\}}\right. \\
&\qquad \left. - \prod_{\substack{m=1\\ b_{i,m}\neq 0}}^{n_v} \left(p_{+1}^{(l-1)}(m,i) - p_{-1}^{(l-1)}(m,i)\right)^{b_{i,m}-1\{m=j\}}\right). \\
q_0^{(l)}(i,j) &= 1 - \prod_{\substack{m=1\\ b_{i,m}\neq 0}}^{n_v} \left(1 - p_0^{(l-1)}(m,i)\right)^{b_{i,m}-1\{m=j\}}. \\
q_{+1}^{(l)}(i,j) &= 1 - q_{-1}^{(l)}(i,j) - q_0^{(l)}(i,j).
\end{aligned}$$

and for the VN types update equations, a slight change in notation is required to differentiate between each VN type and each CN type. The notation for the VNs, CNs, and channel message probabilities is as following respectively:

$$\boldsymbol{p}^{(l)}(i) = \left(p_{-1}^{(l)}(i), p_{+1}^{(l)}(i), p_0^{(l)}(i)\right) \qquad \boldsymbol{q}^{(l)}(i) = \left(q_{-1}^{(l)}(i), q_{+1}^{(l)}(i), q_0^{(l)}(i)\right)$$

$$\boldsymbol{m} = (m_{-\omega}, m_{-\omega+1}, \ldots, m_{+\omega})$$

The intermediate probability vector becomes $\boldsymbol{z}^{(l)}(i) = \left[\circledast_{i\neq j} q_{-1}^{(l)}(j)\right] \circledast \boldsymbol{m}$, and the VN update equations for all output states are calculated as following:

$$p_{-1}^{(l)}(i) = \sum_{n<0} z_n^{(l)}(i) \qquad p_0^{(l)}(i) = z_0^{(l)}(i) \qquad p_{+1}^{(l)}(i) = \sum_{n>0} z_n^{(l)}(i).$$

3.4 Spatially-Coupled Codes

Performing the decoding on the tanner graph of the corresponding $\boldsymbol{H}$ matrix is known to be a sub-optimal procedure. In other words, the error-correction capability

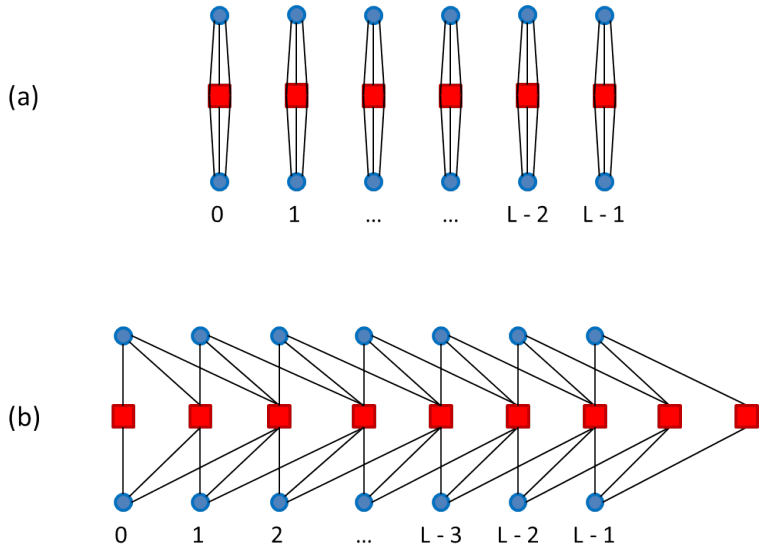

Figure 3.4: (a) L uncoupled $(3, 6)$ protographs (b) SC-$(3, 6)$ protographs with coupling length L, and memory length $m_{\text{sc}} = 2$.

of the code being used will be lower, much lower in this case, than the code capacity limit theorized by Shannon [18]. However, its has been proven that by introducing memory into the construction of the desired LDPC code, which is the idea behind convolutional codes (CC), it can almost achieve capacity even with sub-optimal decoding [19] [20]. The first description of LDPC convolutional codes (LDPCC) came in [21] and [22] back in 1999. Introducing memory to an LDPC code ensemble can be done by coupling (connecting) a protograph to several other protographs, depending on the memory length, in a process called Spatial-coupling (SC) [23].

Taking the $(3, 6)$ LDPC code ensemble as an example, an SC-$(3, 6)$ LDPC code ensemble can be created by first having L independent $(3, 6)$ protographs, where L is the coupling length. Then depending on the memory length m_{sc} , a protograph at position i can disconnect several of its edges form its corresponding VN types, and connect them to another VN type of another protograph as far as $i - m_{\text{sc}}$ positions before. As a result, an SC ensemble is created with L protographs interconnected with m_{sc} as the memory constraint length. Figure 3.4 visualises the process explained above.

A base protograph matrix of a SC code ensemble with coupling length L, B_n VN types, B_k CN types, and memory length of m_{sc} will have the following form:

$$\boldsymbol{B}_{SC} = \begin{bmatrix} \mathbf{B}_0 & & & \\ \mathbf{B}_1 & \mathbf{B}_0 & & \\ \vdots & \mathbf{B}_1 & \ddots & \\ \mathbf{B}_{m_{\text{sc}}} & \vdots & \ddots & \mathbf{B}_0 \\ & \mathbf{B}_{m_{\text{sc}}} & \ddots & \mathbf{B}_1 \\ & & \ddots & \vdots \\ & & & \mathbf{B}_{m_{\text{sc}}} \end{bmatrix}$$

where the empty spaces are zeros, and each $\boldsymbol{B}_i$ for $i = 0, \ldots, m_{\text{sc}}$, are each a composite base protograph matrix, when combined they form the original LDPC base protograph matrix. For the example in figure 3.4, we have $\boldsymbol{B}_0$, $\boldsymbol{B}_1$, $\boldsymbol{B}_2$, where the original $(3, 6)$ LDPC base protograph is $\boldsymbol{B} = \boldsymbol{B}_0 + \boldsymbol{B}_1 + \boldsymbol{B}_2$, i.e. $\boldsymbol{B} = [3\ 3] = [1\ 1] + [1\ 1] + [1\ 1]$ [23] [24].

By expanding each $\boldsymbol{B}_i$, in the SC base protograph matrix, the corresponding $\boldsymbol{H}$ matrix of the SC code can be constructed, and it will have the following form:

$$\boldsymbol{H}_{SC} = \begin{bmatrix} \mathbf{H}_0 & & & \\ \mathbf{H}_1 & \mathbf{H}_0 & & \\ \vdots & \mathbf{H}_1 & \ddots & \\ \mathbf{H}_{m_{\text{sc}}} & \vdots & \ddots & \mathbf{H}_0 \\ & \mathbf{H}_{m_{\text{sc}}} & \ddots & \mathbf{H}_1 \\ & & \ddots & \vdots \\ & & & \mathbf{H}_{m_{\text{sc}}} \end{bmatrix}$$

where each $\boldsymbol{H}_i$ will be made up of $M \times M$ matrices, where M is referred to as the lifting factor. After expansion (lifting), the SC code length will be $n = \eta \times L \times M$, where $\eta = \frac{B_n}{B_k}$ [23] [24].

The decoding of a SC LDPC code can be done in the same way as the block LDPC code. This is referred to as full window decoding of the SC code, since all VNs and CNs are used in the decoding process. However, sometimes L and M can be quite large, and the complexity of the decoder becomes too big to decode all VNs and CNs at once. Hence, a sliding window (SW) decoder can be used to implement the decoding procedure in parts. First, the window will include W CN type positions ($M \times W$ CN positions) and $\eta \times W$ VN type positions ($\eta \times W \times M$ VN positions), starting from the first CN and VN. W, is referred to as the window size [23] [24]. The window will perform the standard decoding procedure for all the CNs and VNs included in the window, along with all the VNs and CNs connected to them that are outside the window. The window will update the message values for all CNs and VNs included and moves M position in respect to the CNs, and $\eta \times M$ positions in respect to the VNs, and the decoding process is done again. The position of the sliding window keeps on shifting until all $(L + m_{\text{sc}}) \times M$ CNs are

processed and decoded [23] [24]. Figure 3.5 visualizes the sliding window decoding process.

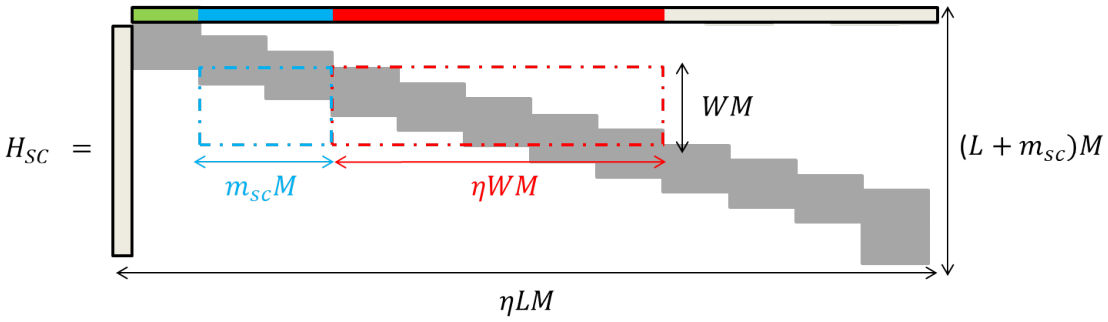

Figure 3.5: Sliding Window Decoder, where the VNs in green are already processed, the VNs in blue are not updated but they still pass messages to neighbouring CNs, and the VNs in red are being processed.

4

QC-MDPC Codes as Variant for the McEliece Cryptosystem

Many modifications of the McEliece cryptosystem have been proposed to minimize the required key size for encryption and decryption. One of these modifications was introduced by Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo Barreto in [8] and enhanced by Gianluigi Liva and Hannes Bartz in [10]. These modifications use QC medium density parity check (MDPC) codes as the linear code $\mathcal{C}$.

4.1 Key Generation, Encryption, and Decryption

The parity check matrix $\boldsymbol{H}$ of the QC-MDPC codes used in [8] and [10] have the following form:

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{h}_0 & \boldsymbol{h}_1 & \dots & \boldsymbol{h}_{n_0-1} \end{pmatrix} \tag{4.1}$$

where each $\boldsymbol{h}_i$ is a $Q \times Q$ circular matrix of weight$(\boldsymbol{h}_i) = d^{(i)}$, for $i = 0, 1, ..., n_0 - 1$. Therefore, the private key of this cryptography system is the $\boldsymbol{H}$ matrix itself, and the public key is the corresponding generator matrix $\boldsymbol{G}$. The message encryption follows the same procedure as the McEliece cryptosystem, and the message decryption (decoding) can be done using the standard LDPC codes decoding algorithms (Gallager A, B, algorithm E, or SPA) [18]. In [10], the authors use Gallager E and SPA as the decoding algorithms. While in [25], Gallager B, and modifications of B and E are used, as will be shown later. Since $\boldsymbol{G}$ is QC, it can be fully represented by its first column; hence, the size of the key is the same as the size of the first column $= 1 \times Q$, instead of being the entire $\boldsymbol{G}$ matrix in the non-QC case. Finally, the search or recovery of the private key requires the search for the correctly used $\boldsymbol{H}$ matrix, in a vector space composed of all possible $\boldsymbol{H}$ matrices for this code. This procedure is complex enough that the need for $\boldsymbol{S}$ and $\boldsymbol{P}$ matrices is dropped.

4.2 Protograph Types Used

As implied earlier, base protograph matrices can help in constructing the desired codes, in our case, it is the QC-MDPC codes. In [8] the authors introduce a base protograph matrix form that is used in the same paper, in [10], and in [25]. Furthermore, the authors in [10] introduced another form, which is an enhancement of the first form. The two base protograph matrix forms are as following respectively:

$$\boldsymbol{B} = \begin{pmatrix} b_{00} & b_{01} \end{pmatrix} \qquad \boldsymbol{B} = \left(\begin{array}{c|cc} 1 & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{array}\right)$$

where each $b_{i,j}$ represents the number of edge connections between each VN and CN type.

In the second base protograph matrix form, all VNs resulting form the first VN type (left of the line) are state VNs. The messages being passed by these state VNs are always erasures (zeros). This protograph type is mainly used while decoding with algorithm E or SPA decoding algorithms. From these protographs, three code ensembles were produced and analyzed in the paper:

$$\varepsilon_A = \begin{pmatrix} 45 & 45 \end{pmatrix} \qquad \varepsilon_B = \left(\begin{array}{c|cc} 1 & 8 & 8 \\ 5 & 5 & 5 \end{array}\right) \qquad \varepsilon_C = \left(\begin{array}{c|cc} 1 & 22 & 22 \\ 2 & 1 & 1 \end{array}\right)$$

Each ensemble has the following key space (number of possible $\boldsymbol{H}$ matrices/private keys):

$$N(\varepsilon) = \prod_{i,j} \binom{Q}{b_{ij}}$$

4.3 DE and Error-Correction Performance of the QC-MDPC Codes and Their Security Level

The implementation of the DE of the QC-MDPC code ensembles uses Algorithm 1, while the implementation of the error-correction performance of the codes uses an algorithm structure described in [10] and [25]. The algorithm structure contains a main algorithm that runs several sub-algorithms inside it. The main algorithm is as following:

Algorithm 6: Error-Correction Performance Implementation

Assuming a regular code ensemble is used with two VN types and one CN type (eg. (45,45)).

Define: Code length n, number of rows n_k, maximum number of iterations $I_{\max}$, maximum, minimum, and decrement of weight of error pattern $w_{\max}$, $w_{\min}$, and w_{dec}, number of frames in error required $F_{\max \text{ in error}}$, and number of columns and rows of base protograph, B_n and B_k.

for *current weight of error patter w, where* $w_{min} \leq w \leq w_{max}$ **do**
  for *frames in error* $F_{err} \leq F_{max\ in\ error}$ **do**

    Sub-Algorithm 1: Build the parity check matrix $\boldsymbol{H}$ for this frame.

    Encode message $\boldsymbol{m}$ into codeword $\boldsymbol{c}$. (In this project the all zero codeword was used).

    Sub-Algorithm 2: Generate error vector **e** and compute $\boldsymbol{c}' = \boldsymbol{c} + \boldsymbol{e}$.

    Decoding Algorithm: Using $\boldsymbol{H}$ decode $\boldsymbol{c}'$ into $\hat{\boldsymbol{c}}$.

    for *all elements in* $\boldsymbol{c}$ *and* $\hat{\boldsymbol{c}}$ **do**
      if $c_i \neq \hat{c}_i$ **then**
        $F_{\text{err}} = F_{\text{err}} + 1$
        exit
      else
        continue
      end
    end
  end
end

In sub-algorithm 1, a new parity check matrix is built for each frame, where its construction is taken from the description in [25] and [26]. By creating a new $\boldsymbol{H}$ every frame, this will guarantee calculating the FER as an ensemble average. It should be noted that the generated $\boldsymbol{H}$ will be QC, which means each entry in a row is shifted by one unit to the right from the previous row. This sub-algorithm works as following:

Sub-Algorithm 1: Build $\boldsymbol{H}$ for current frame

Input: Size of the circulant Q (for our example $Q = n_k$), protograph matrix to be used B (for our example B= (45, 45)).

Output: Parity Check Matrix $\boldsymbol{H}$.

for $\boldsymbol{B}_{00}$ *and* $\boldsymbol{B}_{01}$ **do**

Place $\boldsymbol{B}_{00}$ / $\boldsymbol{B}_{01}$ ones randomly across the first/second half of the first row of $\boldsymbol{H}$.

for *row* ***1*** *to* $\boldsymbol{Q}$ **do**

Shift each one in first row as following:

$Position_{\text{new}} = (Position_{\text{old}} + 1) \mod Q$.

end

end

Finally, the procedure of adding the error vector to the codeword follows the same procedure as the McEliece cryptosystem. In sub-algorithm 4, exactly w ones are placed randomly in error vector $\boldsymbol{e}$, and then added to codeword $\boldsymbol{c}$ to produce $\boldsymbol{c}'$.

4.3.1 Density Evolution and Error-Correction Performance

Table 4.1 shows the key space and decoding thresholds of SPA and algorithm E for each ensemble using density evolution (DE). In addition, Figure 4.1 shows the frame error probability for ε_A and ε_C with different algorithms and optimal ω values, produced by monte-carlo simulations [10].

Ensemble	Key Space $N(\varepsilon)$	$n\delta^*_{\text{SPA}}$ (ω)	$n\delta^*_{\text{E}}$ (ω)
ε_A	2^{715}	113(1) 112(0.5)	57(1) 106(13)
ε_B	2^{328}	132(1)	25(1) 57(4)
ε_C	2^{446}	171(1) 155(0.8)	43(1) 128(8)

Table 4.1: DE thresholds and key space sizes for different ensembles

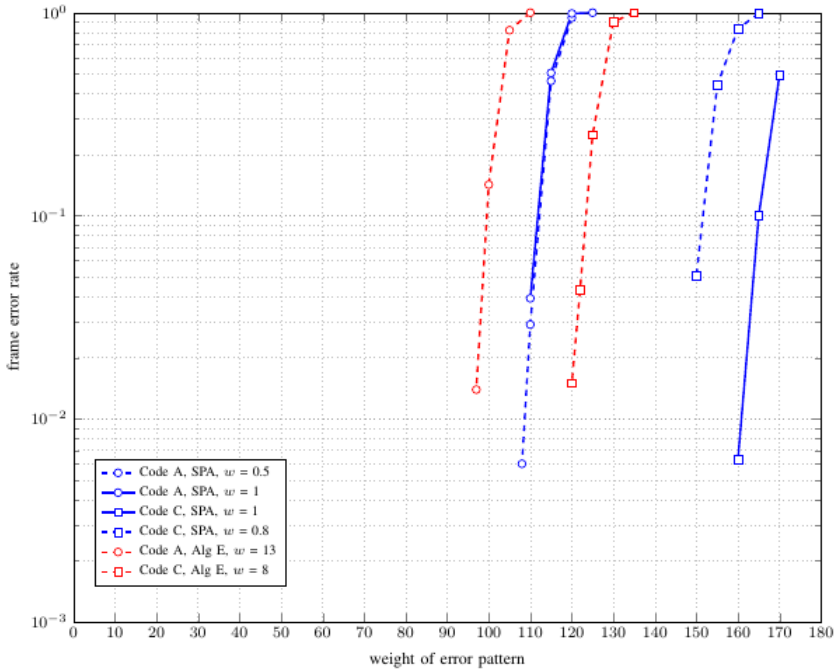

Figure 4.1: Frame Error Rate vs Weight of error pattern of ε_A and ε_C using SPA and algorithm E decoding algorithms

In both DE analysis and error-correction curves, a circulant of size $Q = 4801$ and block length of $n = 9602$ were used. A simulation point is validated after 100 frames in error are reached, and the maximum number of iterations used in the decoding algorithms is $I_{\max} = 100$. Regarding the DE analysis from Table 4.2, ε_A performs the worst with the SPA, while in algorithm E, ε_B performs the worst. The effect of the scaling factor is more prominent in the algorithm E case, where all ensembles performed better when the scaling factor was used. The monte-carlo simulations in Figure 4.1 agree with the above, as ε_C outperforms ε_A in all cases. In addition, the effect of introducing the scaling factor shown in the figure agrees with the DE analysis.

4.3.2 Security Level of the QC-MDPC Codes

As mentioned earlier, there are two attacks that can be performed to break the McEliece cryptosystem; message attacks and key attacks. The security of the QC-MDPC code ensembles against these attacks can be measured with the WFs required for performing them. Since the best known technique to per from this attack uses ISD, the WFs of these attacks will be compared to WF_{ISD}. So denote by $WF_{\text{ISD}}(n, Q, t)$ the bit combinations required for decoding a weight t error vector, using ISD. Furthermore, denote by $WF_{\text{dist}}(n, Q, t)$ the bit combination required for the key distinguishing attack. According to [8], the key recovery attack will have

the same WF as the key distinguishing attack. Finally, denote by $WF_{\text{dec}}(n, Q, t)$ the bit combinations required to perform the decoding attack. Now, according to [8] and [10], a quantum computer will have the following WFs:

$$WF_{\text{dist}}(n, Q, t) = \frac{WF_{\text{ISD}}(n, Q, t)}{\sqrt{Q}}$$

$$WF_{\text{dec}}(n, Q, t) = \frac{WF_{\text{ISD}}(n, Q, t)}{Q}$$

Table 4.2 shows the WFs for ε_A and ε_C. It is clear from the table that ε_C achieves the best security level.

Ensemble $\varepsilon_A(t = 84)$	Ensemble $\varepsilon_C(t = 102)$
$WF_{\text{ISD}}(9602, 4801, 84) = 2^{87.1}$	$WF_{\text{ISD}}(9602, 4801, 102) = 2^{104.4}$
$WF_{\text{dist}}(9602, 4801, 84) = 2^{81}$	$WF_{\text{dist}}(9602, 4801, 102) = 2^{98.3}$
$WF_{\text{dec}}(9602, 4801, 84) = 2^{74.9}$	$WF_{\text{dec}}(9602, 4801, 102) = 2^{92.2}$

Table 4.2: WFs for ε_A and ε_C.

Finally, Table 4.3 shows the required key sizes for the Goppa codes, MDPC codes, and the QC-MDPC codes. As expected, the QC-MDPC codes achieve the minimum key size.

QC-MDPC	MDPC	Goppa Codes
4800 bits	12096 bits	460647 bits

Table 4.3: Key sizes for QC-MDPC code, compared to regular MDPC code, and a Goppa code

5

The GJS Reaction-Based Key Attack

An attack to recover the private key of the cryptosystem is introduced and explined by Qian Guo, Thomas Johansson, and Paul Stankovski in [17], and is alos further explained and implemented by Liva and Bratz in [26]. The attack exploits a weakness noticed in the cryptosystem when QC-MDPC codes are used.

5.1 Attack Description

The aim of this attack procedure is to recover the first circulant of the secret matrix $\boldsymbol{H}$, which is denoted by $\boldsymbol{H}_0$, from its corresponding public matrix $\boldsymbol{G}$. Since the secret matrix $\boldsymbol{H}_0$ is QC, it is sufficient to only recover the first row of the corresponding matrix, $\boldsymbol{h}_0$. The main idea behind this attack is to check the number of decoding failures reported using a certain set of different error patterns. The attack generates these error patterns according to a certain criteria, which will be explained shortly.

Taking ε_{A} code ensemble as an example, the attack works on recovering the corresponding $\boldsymbol{H}$ matrix as following. Let $\boldsymbol{\Psi}_d$ be the set of all binary vectors of length $n = 2Q$, containing exactly t number of ones. These t ones, are placed in $t/2$ pairs randomly across the first half of the chosen error vector $\boldsymbol{v}$ where the distance between the ones of each pair is d. The second half of the error vector will be all zeros, since the attack only focuses on recovering the first circulant $\boldsymbol{H}_0$. The following equation gives a mathematical description of the generation of the error set $\boldsymbol{\Psi}_d$ and error vector $\boldsymbol{v}$:

$$\begin{aligned}\boldsymbol{\Psi}_d = \{\, \boldsymbol{v} = (\boldsymbol{e}, \boldsymbol{f}\,) \mid \omega_{\mathbf{H}}(\mathbf{f}\,) = 0, &\text{and all distinct } s_1, s_2, ..., s_t,\ s.t\ \boldsymbol{e}_{s_i} = 1, \\ &\text{and}\ \ s_{2i} = (s_{2i-1} + d) \bmod r \text{ for } i = 1, \ldots, \frac{t}{2}\Big\}.\end{aligned} \tag{5.1}$$

The attack will perform M decoding trials on the QC-MDPC code, where each trial picks an error vector from $\boldsymbol{\Psi}_d$. This will be done for $d = 1,\ldots,\frac{Q}{2}$, where $U = \frac{Q}{2}$ is an upper bound. Hence, a total of $M \times U$ trials will be performed, and a decoding error probability can be calculated for each trial.

The motivation behind this attack is that there is a correlation between the decoding error probability calculated and the frequency of occurrence of a distance

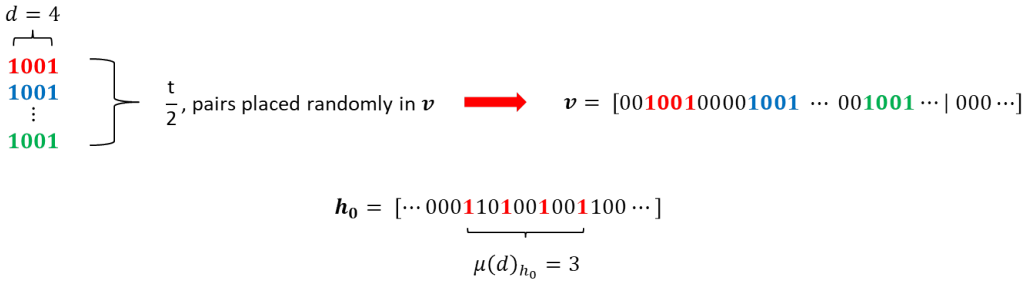

Figure 5.1: Generation of $\frac{t}{2}$ random pairs of ones in error vector $\boldsymbol{v}$, and its corresponding multiplicity in $\boldsymbol{h}_0$.

d in the first circulant $\boldsymbol{H}_0$. In other words, the more this distance occurs in the matrix $\boldsymbol{H}_0$, or vector $\boldsymbol{h}_0$, the lower the calculated decoding error probability will be. The number of times the distance occurs in a given vector is referred to as the multiplicity of this distance in that vector, and denoted by $\mu(d)$. As a result, a distance profile of $\boldsymbol{h}_0$ can be constructed, where it gives all the distances available in $\boldsymbol{h}_0$, and their corresponding multiplicities. The distance profile of $\boldsymbol{h}_0$ is denoted by $D(\boldsymbol{h}_0)$, and is classified as follows:

$$D(\boldsymbol{h}_0) = \{\ d \text{ repeated } \mu(d) \text{ times, for } d = 1, \ldots, U\}.$$

Finally, Figure 5.1 shows the way the error vector is generated for the attack for a given distance, along with how the multiplicity of this distance will appear in $\boldsymbol{h}_0$.

5.2 Secret Key H Reconstruction

Once the attack is done, and $D(\boldsymbol{h}_0)$ is successfully constructed, the reconstruction of $\boldsymbol{h}_0$, and hence $\boldsymbol{H}_0$, can be done easily. First, a one is placed in the first position of the reconstruction vector $\boldsymbol{h}_{\text{recons}}$. The second one is placed at a position i_0, where i_0 is equal to the first distance in $D(\boldsymbol{h}_0)$. The third one is placed at any position i_1 and the distance between this one and the previous two ones is calculated. If these calculated distances appear in the distance profile, then the third one is kept in its position; if not, then a new position is tested. This is repeated until the calculated distances, with this position i_1, appear in $D(\boldsymbol{h}_0)$. The same procedure is repeated for the fourth, fifth, sixth ones and so on, where all the calculated distances between the n-th one and the previous $n-1$ ones, should all appear in $D(\boldsymbol{h}_0)$.

In the end $\boldsymbol{h}_0$ will be fully reconstructed, and $\boldsymbol{H}_0$ can then be reconstructed by cyclically shifting each entry of $\boldsymbol{h}_0$ across all rows. Afterwards, the rest of the secret key $\boldsymbol{H}$ can be reconstructed form $\boldsymbol{H}_0$ using linear algebra [26] [25]. Figure 5.2 shows the $\boldsymbol{h}_0$ reconstruction procedure for a given distance profile.

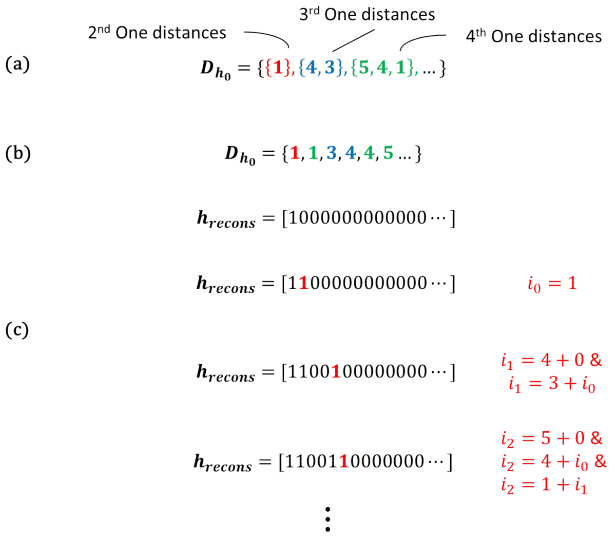

Figure 5.2: $\boldsymbol{h}_0$ reconstruction: (a) shows the distances for each one in the distance profile, (b) shows the actual distance profile, and (c) shows the way $\boldsymbol{h}_0$ is reconstructed using this distance profile.

5.3 Attack Implementation

The GJS reaction-based attack is explained and implemented in both [26] and [25]. The description of the attack in these papers explains the procedure to attack an actual code-based post-quantum cryptosystem, where the attacker has no knowledge of $\boldsymbol{H}$ or $D(\boldsymbol{h}_0)$. However, the described process can be tedious and time consuming to implement in its full scale; in addition, proving whether the attack works or not does not require the full process as well. As a result, [25] introduces a backward engineered implementation of the attack as will be shown later, to prove the effectiveness of the attack.

The algorithm used to implement the attack uses the same general procedure as algorithm 6, but with some alterations. The first main alteration is in the sub-algorithm for constructing $\boldsymbol{H}$, where this algorithm includes building the corresponding $D(\boldsymbol{h}_0)$. The second main alteration is in the sub-algorithm for generating the error vector, where this generation follows equation 5.1. The main algorithm for implementing the attack works as following:

Algorithm 7: GJS Attack Trials Implementation

Assuming a regular code ensemble is used with two VN types and one CN type (eg. (45,45)).

Define: Code length n, number of rows n_k, maximum number of iterations $I_{\max}$, required weight of error vector w, maximum multiplicity number $\mu_{\max}$, number of trials required M per multiplicity, maximum number of frames in error required $F_{\max \text{ in error}}$, and number of columns and rows of base protograph, B_n and B_k.

for *multiplicities from 0 to* μ_{max} **do**
  for *decoding trials from* 1 *to* M **do**
    for *frames in error* $F_{err} \leq F_{max\,in\,error}$ **do**

      Sub-Algorithm 3: Build the parity check matrix $\boldsymbol{H}$ for this frame with its corresponding $D(\boldsymbol{h}_0)$.

      Encode message $\boldsymbol{m}$ into codeword $\boldsymbol{c}$. (In this project the all zero codeword was used).

      Sub-Algorithm 4: Generate error vector $\boldsymbol{e}$ according to equation 5.1 and compute $\boldsymbol{c'} = \boldsymbol{c} + \boldsymbol{e}$.

      Decoding Algorithm: Using $\boldsymbol{H}$ decode $\boldsymbol{c'}$ into $\hat{\boldsymbol{c}}$.

      for *all elements in* $\boldsymbol{c}$ *and* $\hat{\boldsymbol{c}}$ **do**
        if $c_i \neq \hat{c}_i$ **then**
          $F_{\text{err}} = F_{\text{err}} + 1$
          exit
        else
          continue
        end
      end
    end
  end
end

Sub-algorithm 3 is identical to Sub-algorithm 1 in constructing $\boldsymbol{H}$; however, sub-algorithm 3 also stores the lee distances available in $\boldsymbol{h}_0$, along with the frequency of occurrence of these distances. In essence, this sub-algorithm builds $D(\boldsymbol{h}_0)$. Afterwards, M distances are chosen for each multiplicity to perform the decoding trials. This way of performing the attack is much more time efficient compared to the full

Sub-Algorithm 3: Build $\boldsymbol{H}$ and corresponding $D(\boldsymbol{h}_0)$ for current frame

Input: Size of the circulant Q (for our example $Q = n_k$), protograph matrix to be used $\boldsymbol{B}$ (for our example $\boldsymbol{B} = (45, 45)$).

Output: Parity Check Matrix $\boldsymbol{H}$ and corresponding distance profile $D(\boldsymbol{h}_0)$.

for $\boldsymbol{B}_{00}$ *and* $\boldsymbol{B}_{01}$ **do**

  Place $\boldsymbol{B}_{00}$ / $\boldsymbol{B}_{01}$ ones randomly across the first/second half of the first row of $\boldsymbol{H}$.

  for *row* $\boldsymbol{1}$ *to* $\boldsymbol{Q}$ **do**
    Shift each one in first row as following:
    $Position_{new} = (Position_{old} + 1) \mod Q$.
  end
end

for *all entries of first half of first row* $\boldsymbol{h}_0$ **do**
  Calculate all Lee distances from current entry i_n to all previous n - 1 entries.
  if *a distance d exists in* $\boldsymbol{h}_0$ **then**
    $\mu(\mathbf{d}) = \mu(\mathbf{d}) + 1$ in $D(\boldsymbol{h}_0)$.
  end
end

scale procedure explained in [26] and [25], in addition, it still manages to prove whether the attack is effective or not. It should be noted that for an actual code-based post-quantum cryptosystem, the only way to perform the attack is using the full scale procedure.

5.4 Reaction Key Attack Effectiveness vs the QC-MDPC Codes

Liva and Bratz implemented the attack described in [17] and [26] in their paper in [25]. The attack proved to be effective versus the QC-MDPC codes introduced in Chapter 4, as will be shown later. However, Liva and Bratz managed to counteract the attack by modifying their decoders.

5.4.1 GJS Reaction-Based Attack vs Regular Decoding Algorithms and a Modification of Gallager B

In [25], the attack managed to expose the structure of $\boldsymbol{H}$ for Gallager B, algorithm E, and SPA decoding algorithms, which proves that this proposed cryptosystem is easily breakable. However, Liva and Bratz also implemented the attack on a variation of Gallager B, introduced by Nenad Miladinovic and Marc Fossorier in [27], and the attack proved to be ineffective. The two modifications of Gallager B are reffered to as Miladinovic and Fossorier (MF)-1 and MF-2. These modification work by changing the VN to CN messages of Gallager B decoding algorithm at an iteration l, with a certain probability $p_e^{(l)}$. Defining the initial value of this probability at iteration zero as $p_e^{(0)} = p^*$, and a decrement $p_{dec} \leq p^*$, the probability update rule is as following:

$$p_e^{(l)} = \begin{cases} p_e^{(l-1)} - p_{dec} & \text{if } \; p_e^{(l-1)} \geq p_{dec} \\ 0 & \text{else} \end{cases} \tag{5.2}$$

The two MF variants update rules work as following:

Variant 1 (MF-1): If the number of incoming CN messages different from current CN message value c_b that do not agree with c exceeds the threshold b, i.e. if $|\{c' \in N(v) \neq c \; : \; m_{\mathrm{c}'\to\mathrm{v}} = -c_\mathrm{b}\}| \geq b$, the VNs send the messages

$$m_{\mathrm{v}\to\mathrm{c}} = \begin{cases} -c_\mathrm{b} & \text{with probability } \; 1 - p_e^{(l)} \\ c_\mathrm{b} & \text{with probability } \; p_e^{(l)} \end{cases} \tag{5.3}$$

and if they do not exceed b, then $m_{\mathrm{v}\to\mathrm{c}} = c_\mathrm{b}$.

Variant 2 (MF-2): For this variant, the iteration count needs to be introduced, and it is denoted by l. At iteration l, if the number of incoming CN messages different from current CN message value c_b that do not agree with c exceeds the threshold b, i.e. if $|\{c' \in N(v) \neq c \; : \; m^{(l-1)}_{\mathrm{c}'\to\mathrm{v}} = -c_\mathrm{b}\}| \geq b$, the VNs send the messages

$$m_{\mathrm{v}\to\mathrm{c}} = \begin{cases} -c_\mathrm{b} & \text{with probability } \; 1 - p_e^{(l)} \\ m^{(l-1)}_{\mathrm{v}\to\mathrm{c}} & \text{with probability } \; p_e^{(l)} \end{cases} \tag{5.4}$$

and if they do not exceed b, then $m^{(l)}_{\mathrm{v}\to\mathrm{c}} = c_\mathrm{b}$.

Figure 5.3 shows the GJS attack trials on the (45,90) QC-MDPC code ensemble using the MF-1, Gallager E, and SPA decoding algorithms. It is evident in the Figure that using any decoding algorithm other than MF, will result in private key $\boldsymbol{H}$ reconstruction and breaking the cryptosystem. Evidently having the VN to CN messages changing randomly instead of deterministically, will conceal the structure of the corresponding $\boldsymbol{H}$; hence, the ineffectiveness of the GJS attack on the MF decoding algorithms.

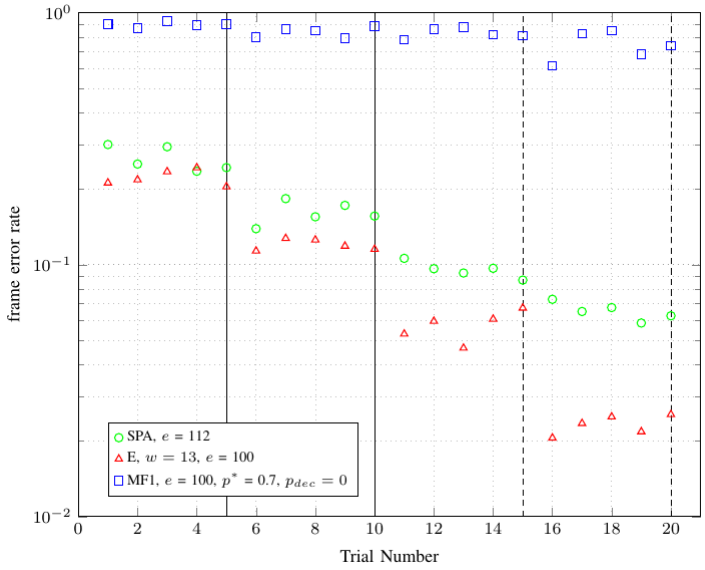

Figure 5.3: GJS reaction-based attack on Gallager E, MF-1, and BP decoding algorithms with their corresponding decoding parameters. All decoding algorithms, except for MF-1, have the pattern of decreasing FER for increasing multiplicity. This means that only MF-1 is safe from this type of attack.

5.4.2 GJS Reaction-Based Attack vs Modifications of Algorithm E

Liva and Bratz used the concept of introducing randomness to the VN to CN messages to conceal the structure of the corresponding $\boldsymbol{H}$, to introduce two modification for Gallager E in [25]. The two modifications use the random erasure message-passing (REMP) approach, where the VN to CN messages are erased (set to 0) under specific conditions with a probability $p_e^{(l)}$.

First modification of Gallager E (REMP-1): In this modification, the message from a VN to a CN is erased with a probability $p_e^{(l)}$, if this message is not an erasure. First, the original (temporary) message is computed as:

$$\tilde{m}_{\mathrm{v}\to\mathrm{c}} = \text{sign}\left[\omega m_{\mathrm{ch}} + \sum_{c'\in N(v)\setminus c} m_{\mathrm{c}'\to\mathrm{v}}\right] \tag{5.5}$$

Then if the temporary message is not an erasure, $\tilde{m}_{\mathrm{v}\to\mathrm{c}} \neq 0$, then the actual VN to CN message is computed as following:

$$m_{\mathrm{v}\to\mathrm{c}} = \begin{cases} \tilde{m}_{\mathrm{v}\to\mathrm{c}} & \text{with probability } 1-p_e^{(l)} \\ 0 & \text{with probability } p_e^{(l)} \end{cases} \tag{5.6}$$

and if $\tilde{m}_{\mathrm{v}\to\mathrm{c}}= 0$, then $m_{\mathrm{v}\to\mathrm{c}} = 0$. The update equation for the CNs and the final bit decision is the same as the original Gallager E. In addition, similar to the MF

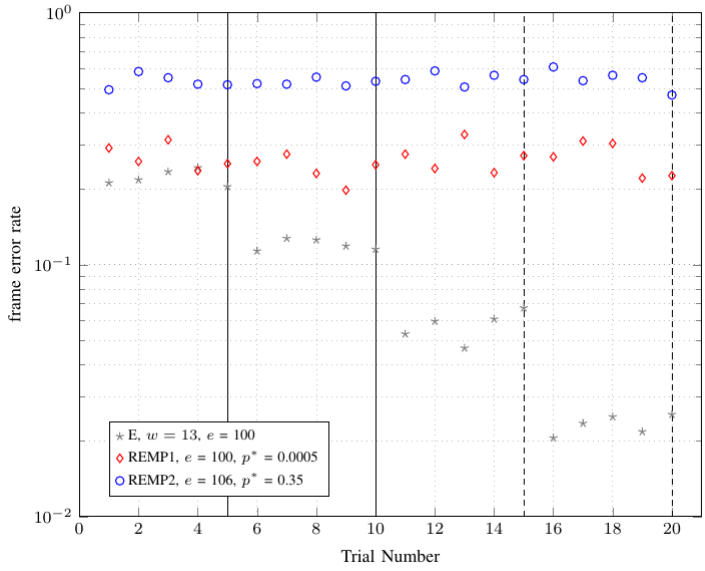

Figure 5.4: GJS reaction-based attack on REMP-1 and REMP-2, compared to Gallager E. Introducing random erasures to $m_{\mathrm{v}\to\mathrm{c}}$ will conceal the structure of the corresponding $\boldsymbol{H}$.

decoding algorithms, the probability $p_e^{(l)}$ is allowed to change after each iteration according to (5.4).

Second modification of Gallager E (REMP-2): Similar to REMP-1, the temporary message $\tilde{m}_{\mathrm{v}\to\mathrm{c}}$ is computed as in (10). However, in this modification, the VN to CN message is erased with a probability $p_e^{(l)}$, if the temporary message does not agree with the cyphertext bit c_{b} ; if $\tilde{m}_{\mathrm{v}\to\mathrm{c}} = -c_{\mathrm{b}}$:

$$m_{\mathrm{v}\to\mathrm{c}} = \begin{cases} \tilde{m}_{\mathrm{v}\to\mathrm{c}} & \text{with probability } 1 - p_e^{(l)} \\ 0 & \text{with probability } p_e^{(l)} \end{cases} \tag{5.7}$$

and if $\tilde{m}_{\mathrm{v}\to\mathrm{c}} = c_{\mathrm{b}}$, then $m_{\mathrm{v}\to\mathrm{c}} = \tilde{m}_{\mathrm{v}\to\mathrm{c}}$.

Figure 5.4 shows the GJS attack trials on REMP-1 and REMP-2, with their corresponding decoding parameters, compared to the original attack trials on Gallager E in Figure 5.3. It is evident that the attack is ineffective against REMP-1 and REMP-2, which further proves that introducing randomness to the VN to CN messages will conceal the structure of $\boldsymbol{H}$. Furthermore, a modification was proposed for BP decoding in [25], referred to as masked belief propagation (MBP), and it also managed to conceal $\boldsymbol{H}$ [25].

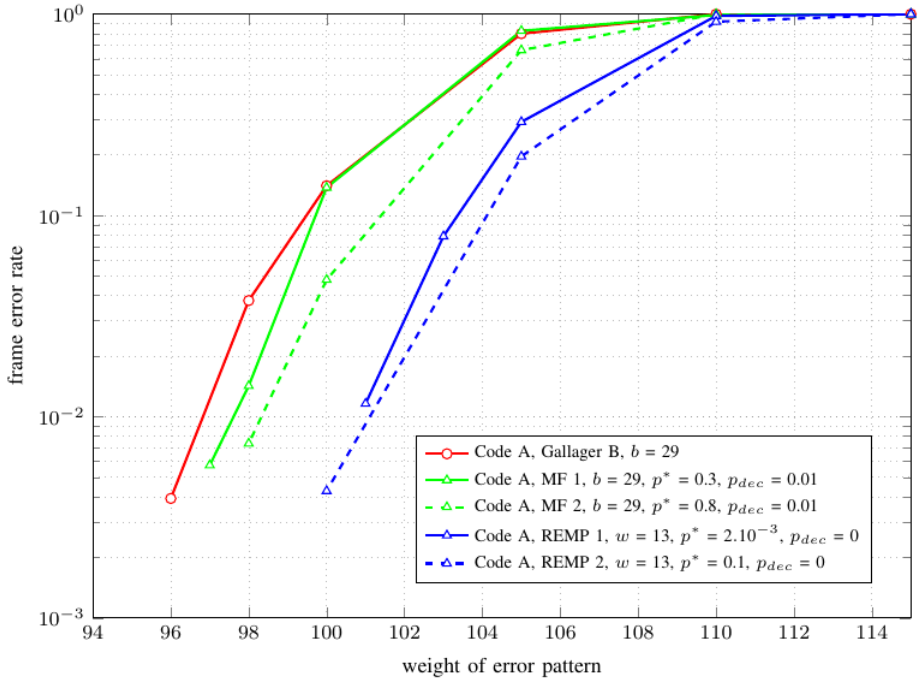

Figure 5.5: FER vs. Weight of error pattern of ε_A ensemble with Gallager B, MF-1, MF-2, REMP1, REMP2 decoding algorithms.

5.4.3 Error-Correction Performance of Gallager B Variants, and Algorithm E Modifications

Figure 5.5 shows the error-correction performance curves for ε_A ensemble with all decoding algorithms introduced in this chapter. A simulation point was validated with 100 frames in error, and a maximum number of iterations per frame as $I_{\max} = 50$. These curves show that this ensemble with these decoding algorithms can be used as codes for the McEliece cryptosystem.

6

Proposed SC QC-MDPC Code Ensembles

As mentioned earlier, SC codes will result in better error performance for a code ensemble when compared to the original block codes case. So it is worth testing and analyzing SC code ensembles as candidates for code-based post-quantum cryptography. As a result, in this project, SC-QC-MDPC code ensembles were proposed, that are the SC case for the code ensembles introduced in [2] and [3]. It is expected for the SC codes to outperform the block codes in terms of error performance. However, it is not clear whether the security level of the cryptosystem will be compromised. A better performing code will place less constraints on the total code length that needs to be used, which in turn will require a lower key size for a specific error correction capability. On the other hand, the structure of the SC code might compromise the security as the attack, whether using ISD or GJS reaction attack, might exploit this structure and easily break the system. As a result, this project analyses and tests some SC-QC-MDPC code ensembles.

6.1 Proposed SC-MDPC Code Ensembles

The SC-MDPC code ensembles tested in this project are as following:

$$\boldsymbol{\varepsilon}_{A\,:\,44\times 80} = \begin{pmatrix} \mathbf{9}\ \mathbf{9}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ 0\ 0\ \cdots\ 0\ 0 \\ 0\ 0\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \mathbf{9}\ \cdots\ 0\ 0 \\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ \mathbf{9}\ \mathbf{9} \end{pmatrix}$$

where this ensemble is a SC version of $\varepsilon_A = \begin{pmatrix} 45 & 45 \end{pmatrix}$, and has a coupling length of $L = 40$ and memory length of $m_{\text{sc}} = 4$.

$$\boldsymbol{\varepsilon}_{B\,:\,58\times 100} = \begin{pmatrix} \mathbf{5}\ \mathbf{5}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ 0\ 0\ 0\ \cdots\ 0\ 0 \\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ 0\ 0\ \cdots\ 0\ 0 \\ 0\ 0\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \mathbf{5}\ \cdots\ 0\ 0 \\ \vdots \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ \mathbf{5}\ \mathbf{5} \end{pmatrix}$$

where this ensemble is a SC version of $\varepsilon_A = \begin{pmatrix} 45 & 45 \end{pmatrix}$, and has a coupling length of $L = 50$ and memory length of $m_{\text{sc}} = 8$.

$$\boldsymbol{\varepsilon}_{C\,:\,84\times120} = \begin{pmatrix}
\mathbf{1\ 7\ 7}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{1\ 1\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 7\ 7\ 1\ 7\ 7}\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{1\ 0\ 0\ 1\ 1\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 8\ 8\ 0\ 7\ 7\ 1\ 7\ 7}\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
0\ 0\ 0\ \mathbf{0\ 8\ 8\ 0\ 7\ 7\ 1\ 7\ 7}\ \cdots\ 0\ 0\ 0 \\
0\ 0\ 0\ \mathbf{0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}\ \cdots\ 0\ 0\ 0 \\
\vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots \\
0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ \mathbf{0\ 8\ 8} \\
0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ \mathbf{0\ 0\ 0}
\end{pmatrix}$$

where this ensemble is a SC version of $\varepsilon_C = \left(\begin{array}{c|cc} 1 & 22 & 22 \\ 2 & 1 & 1 \end{array}\right)$, and has a coupling length of $L = 40$ and memory length of $m_{\text{sc}} = 5$. The first VN, fourth VN, etc. are state VNs.

$$\boldsymbol{\varepsilon}_{D\,:\,92\times120} = \begin{pmatrix}
\mathbf{1\ 3\ 3}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{1\ 1\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 3\ 3\ 1\ 3\ 3}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{1\ 0\ 0\ 1\ 1\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 3\ 3\ 0\ 3\ 3\ 1\ 3\ 3}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 1\ 3\ 3}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 1\ 3\ 3}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 1\ 3\ 3}\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 4\ 4\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 1\ 3\ 3}\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
\mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}\ 0\ 0\ 0\ \cdots\ 0\ 0\ 0 \\
0\ 0\ 0\ \mathbf{0\ 4\ 4\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 0\ 3\ 3\ 1\ 3\ 3}\ \cdots\ 0\ 0\ 0 \\
0\ 0\ 0\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}\ \cdots\ 0\ 0\ 0 \\
\vdots\ \vdots \\
0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ \mathbf{0\ 4\ 4} \\
0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \cdots\ \mathbf{0\ 0\ 0}
\end{pmatrix}$$

where this ensemble is a SC version of $\varepsilon_C = \left(\begin{array}{c|cc} 1 & 22 & 22 \\ 2 & 1 & 1 \end{array}\right)$, and has a coupling length of $L = 40$ and memory length of $m_{\text{sc}} = 13$. The first VN, fourth VN, etc. are state VNs.

6.2 Implementing DE, Error Performance, and GJS Attack Trials for Proposed SC Code Ensembles

Any SC code ensemble will always be irregular, even if the base protographs used are the regular ones. This is due to the fact that the first and last few CNs will have different degrees compared to the rest. As a result, the DE for any SC code ensemble will follow the same CN and VN update equations as the irregular case for all decoding algorithms.

The error-correction performance and the GJS attack trials implementation however needs to be modified slightly compared to the block codes case. Since in this case, a sliding window decoder needs to be implemented for the SC codes. As a result, both algorithms 6 and 7 need to be modified to accommodate the sliding

window decoding procedure. The algorithm implementing the error-correction performance, and the GJS attack trails of the SC codes are shown in Algorithm 8 and 9 respectively.

Algorithm 8: Error Performance Implementation for SC Codes

Assuming a regular code ensemble is used with two VN types and one CN type (e.g. (45,45)).

Define: Code length n, number of rows n_k, coupling length L, maximum number of iterations $I_{\max}$, maximum, minimum, and decrement of weight of error pattern $w_{\max}$, $w_{\min}$, and w_{dec}, number of frames in error required $F_{\max \text{ in error}}$, and number of columns and rows of base protograph, B_n and B_k.

for *current weight of error patter* w, *where* $w_{min} \leq w \leq w_{max}$ **do**
  for *frames in error* $F_{err} \leq F_{max\,in\,error}$ **do**

    Sub-Algorithm 1: Build the parity check matrix $\boldsymbol{H}$ for this frame.

    Encode message $\boldsymbol{m}$ into codeword $\boldsymbol{c}$. (In this project the all zero codeword was used).

    Sub-Algorithm 2: Generate error vector $\boldsymbol{e}$ and compute $\boldsymbol{c}' = \boldsymbol{c} + \boldsymbol{e}$.

    for *all different sliding window positions* **do**

      Decoding Algorithm: Using VNs and CNs of $\boldsymbol{H}$ that are within the window span (ws), decode $\boldsymbol{c}'_{\text{ws}}$ into $\hat{\boldsymbol{c}}_{\text{ws}}$.

    end

    for *all elements in* $\boldsymbol{c}$ *and* $\hat{\boldsymbol{c}}$ **do**
      if $c_i \neq \hat{c}_i$ **then**
        $F_{\text{err}} = F_{\text{err}} + 1$
        exit
      else
        continue
      end
    end
  end
end

Algorithm 9: CJS Attack Trials Implementation

Assuming a regular code ensemble is used with two VN types and one CN type (eg. (45,45)).

Define: Code length n, number of rows n_k, coupling length L, maximum number of iterations $I_{\max}$, required weight of error vector w, maximum multiplicity number $\mu_{\max}$, number of trials required M per multiplicity, maximum number of frames in error required $F_{\max\,\text{in error}}$, and number of columns and rows of base protograph, B_n and B_k.

```
for multiplicities from 0 to μ_max do
    for decoding trials from 1 to M do
        for frames in error F_err ≤ F_max in error do

            Sub-Algorithm 3: Build the parity check matrix H for this
             frame with its corresponding D(h_0).

            Encode message m into codeword c. (In this project the all zero
             codeword was used).

            Sub-Algorithm 4: Generate error vector e according to equation
             5.1 and compute c' = c + e.

            for all different sliding window positions do

                Decoding Algorithm: Using VNs and CNs of H that are
                 within the window span (ws), decode c'_ws into ĉ_ws.

            end

            for all elements in c and ĉ do
                if c_i ≠ ĉ_i then
                    F_err = F_err + 1
                    exit
                else
                    continue
                end
            end
        end
    end
end
```

7

Numerical Results

This chapter shows the numerical results of the proposed SC-QC-MDPC code ensembles. It includes the DE results, error-correction performance results, and the GJS attack trials results.

7.1 DE Results for Proposed SC-MDPC Code Ensembles

Density evolution for these ensembles were performed with Gallager A and algorithm E decoding algorithms. The thresholds achieved for the SC code ensembles are shown in table 4, compared to the block code ensemble. For algorithm E decoding algorithm, the scaling factor ω was optimized over all the positive integers from 1 to $\omega_{\max} = 14$. As expected, the SC code ensembles outperform block code ensembles. In Gallager A, code ensemble $\varepsilon_{A\,:\,44x80}$ threshold is more than double that of ε_A, and code ensemble $\varepsilon_{B\,:\,58x100}$ threshold is worse than that of $\varepsilon_{A\,:\,44x80}$, but still outperforms ε_A. In Algorithm E, all the SC code ensembles outperform the original ε_A, where the DE threshold more than doubles for all. As expected, the SC code ensembles with state VNs, $\varepsilon_{C\,:\,82x100}$ and $\varepsilon_{D\,:\,12x100}$, outperforms ε_A and $\varepsilon_{B\,:\,58x100}$, with $\varepsilon_{D\,:\,12x100}$ having the best threshold. From the DE results, we expect the SC-QC-MDPC codes to give a better error-correction capability, and hence, better t-bit security level when used in the McEliece cryptosystem.

Ensemble	δ_A^*	δ_E^* (ω_{max})
ε_A	0.000255	0.011(14)
$\varepsilon_{A\,:\,44x80}$	0.000557	0.023(14)
$\varepsilon_{B\,:\,58x100}$	0.000291	0.023(14)
$\varepsilon_{C\,:\,82x100}$	–	0.026(14)
$\varepsilon_{D\,:\,12x100}$	–	0.027(14)

Table 7.1: DE thresholds for proposed SC-MDPC codes

7.2 Error-Correction Performance Results

Initially, error-correction performance was implemented with the simple SC-(3,6) ensemble on the binary erasure channel (BEC). The error-correction performance of SC codes on the BEC is the easiest to study, and several literatures have done work in that area. As a result, error-correction curves for the SC-(3,6) ensemble on the BEC were simulated to gain better understanding of SC codes. This will later help in implementing the error-correction performance for the proposed SC ensembles.

The first SC code ensemble to be tested is as following:

$$\boldsymbol{\varepsilon}_{(3,6)\,:\,52\times100} = \begin{pmatrix} \mathbf{1}\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \cdots 0\ 0 \\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \cdots 0\ 0 \\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0 \cdots 0\ 0 \\ 0\ 0\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ 0\ 0\ 0\ 0 \cdots 0\ 0 \\ 0\ 0\ 0\ 0\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ 0\ 0 \cdots 0\ 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1} \cdots 0\ 0 \\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \cdots \mathbf{1}\ \mathbf{1} \end{pmatrix}$$

with coupling length of $L = 50$ and lifting factor of $M = 500$. This will result in a code length of $n = 50{,}000$ bits. Two decoding algorithms were used for the error-correction performance, the peeling decoder (PD) and BP decoder. Although both decoders perform the same over the BEC, both were used to validate the results achieved. Figure 7.1 shows the bit-error rate (BER) curves for the $\varepsilon_{(3,6)\,:\,52x100}$ ensemble using PD and BP decoder, for the full window case and a sliding window size of $W = 10$. In these curves, a simulation point is validated with 100 frames in error, and the maximum number of iterations used was $I_{\max} = 100$ iterations for the full window, and $I_{\max} = 10$ iterations for $W = 10$. It should be noted that a random code ensemble was used in this simulation, unlike the QC code ensembles used earlier.

7.3 GJS Reaction-Based Attack Results vs. Proposed SC-QC-MDPC Codes

GJS reaction-attack decoding trials have been performed in this project, and they use the same procedure as suggested in [26] and [25]. The attack has been performed on all the proposed SC-QC-MDPC code ensembles, using Algorithm E decoding algorithm. Figure 7.2 shows the GJS attack trails on these ensembles, with their corresponding decoding parameters, where Figure 7.2 (a) shows the full window decoding case, and Figure 7.2 (b) shows the sliding window decoding case. A lifting factor of $M = 120$ was used for all simulations. A simulation point is validated with 100 frames in error, and maximum number of iterations used in the decoding process varies according to the code ensemble used and whether FW or SW decoding was

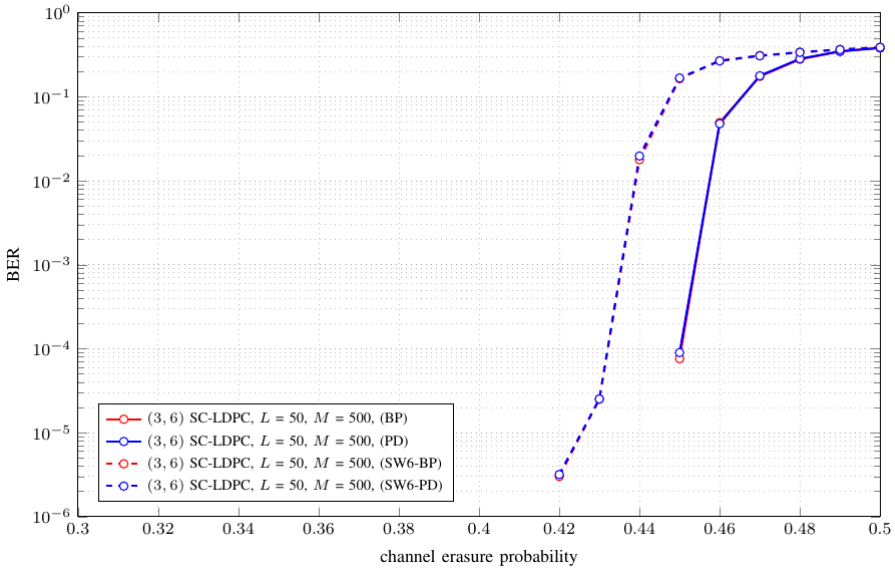

Figure 7.1: BER curves for $\varepsilon_{(3,6):52x100}$ on BEC, with full window decoding and SW decoder size $W = 10$.

implemented. The scaling factor ω was not optimized for the attack trials; never the less, the effectiveness of the attack is independent on the scaling factor used, since error corrections can still be made with sub-optimal scaling factors.

It appears from the results that the reaction key attack is ineffective when used on SC-QC-MDPC code ensembles used in this project, compared to the QC-MDPC ensembles case. As a result, no modifications are required to be done on the decoding algorithms.

The attack ineffectiveness is mainly caused by the fact that there will always be more than one circulant under a VN type. When the attack is performed on the block QC-MDPC ensembles used in this project, mainly the ε_{A} ensemble, it will only have one circulant, $\boldsymbol{H}_0$ to act on. So when decoding trials are performed, and a distance does appear to be present in $\boldsymbol{h}_0$ ($\mu(d) > 0$), it can only be present in $\boldsymbol{H}_0$. However, this is not the case for the SC-QC-MDPC ensembles used, since the attack will always have to act on m_{sc} extra circulants along with the targeted one.

However, the FER for distances with high multiplicities does appear to be lower than the rest of the multiplicities in some plots. This is due to the fact that $\mu(d) = 3$ and $\mu(d) = 2$ are much less frequent to appear in a vector (lower probability of appearing), compared to lower multiplicities. As a result, when a $\mu(d) = 3$ or a $\mu(d) = 2$ does appear to be present in h_0, it will be present in the targeted circulant $\boldsymbol{H}_0$ with high probability. Never the less, the security of the cryptosystem is still intact, since all the distances with lower multiplicities will still not be differentiated for each other (with high probability). Hence, the attacker will mistake the $\mu(d) = 3$ distances as $\mu(d) = 1$ or $\mu(d) = 2$ distances at best.

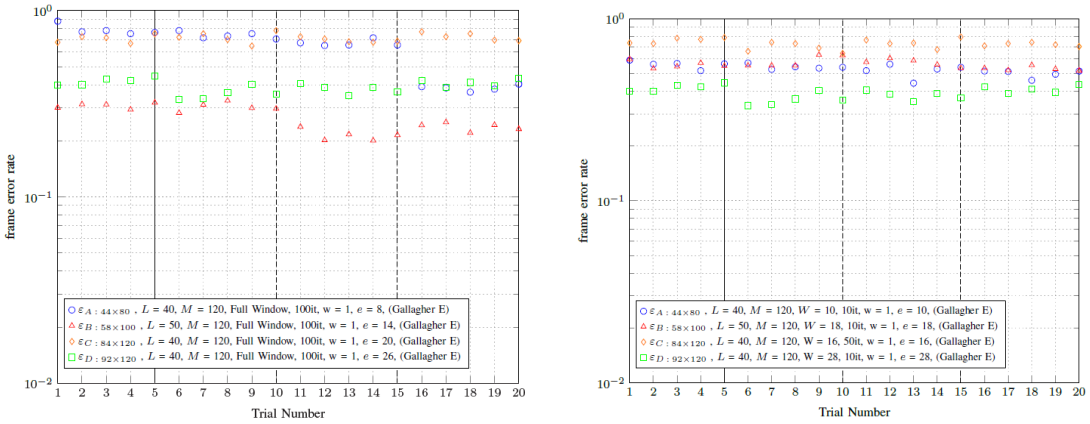

Figure 7.2: GJS reaction-based attack on $\varepsilon_{A:44x80}$, $\varepsilon_{B:58x100}$, $\varepsilon_{C:82x100}$, and $\varepsilon_{D:12x100}$, using algorithm E decoding algorithm with FW and SW decoding

8

Conclusion and Future Work

In this thesis, the error-correction and DE performance along with the security level of several coding schemes were tested as code variants for the McEliece cryptosystem. Investigating these parameters will first provide an insight on the t-bit security level expected from these schemes from message attacks designed for the McEliece cryptosystem. In addition, it will also give an indication on the security level against the key attacks designed for this cryptosystem. First, QC-MDPC codes as variants of the McEliece cryptosystem, which was proposed in [8] and enhanced in [10], were tested and implemented. The original QC-MDPC code ensemble introduced in [8] managed to achieve $t = 84$ error-correcting capability (80-bit security level) with private key size of 4800 bits, while the enhanced code ensemble introduced in [10] achieved an error-correcting capability of $t = 102$ with the same key size.

A key attack introduced in [17] that exploits a weakness in the QC structure of the codes proposed in [8] and referred to as the GJS reaction-based key attack, was tested and implemented as well. The attack works under the assumption that there is a relation between the probability of a decoding failure, of the cryptosystem, and the frequency of occurrence of a lee distance in the parity-check matrix of the code, when decoding an error vector containing many copies of this distance. It has been shown in [17] that a pattern of decreasing FER (probability of decoding failure) for an increasing multiplicity of a Lee distance, will be noticed when used on QC-MDPC codes. The attack was implemented on the QC-MDPC code ensembles in [8] using several decoding algorithms, and it proved to be effective. The attack was also implemented on a variation of Gallager B decoding algorithm introduced in [27], where this new decoding algorithm introduces randomness into the message passing of the decoder, and the attack was ineffective. Authors in [25] used the concept of introducing randomness to the message passing to modify decoding algorithm E, and the attack proved to be ineffective on the modification as well.

Finally, we proposed using SC-QC-MDPC codes as variants for the McEliece cryptosystem, since SC codes are known to have better performance compared to their block codes counterparts. DE was done on the proposed SC-QC-MDPC code ensembles, and as expected, the SC codes have better DE thresholds, and hence better error-correction capabilities, than the QC-MDPC codes. In addition, GJS reaction-based key attack decoding trials were performed on the SC codes, and the attack was ineffective; hence, no modifications were required on the decoding

algorithms.

Regarding the future work, error-correction plots for the SC-QC-MDPC code ensembles need to be simulated. This will give validation to the DE results and give the t-bit level security guaranteed by the cryptosystem. Furthermore, key sizes and key spaces for the SC-QC-MDPC codes need to be calculated and compared to the QC-MDPC codes, which will give validation to the practicality of using SC codes in the cryptosystem. Finally, the WFs of the McEliece cryptosystem attacks need to be calculated for the SC-QC-MDPC codes, which will give an indication on the security level of the cryptosystem compared to the QC-MDPC codes.

Bibliography

[1] C. E. Shannon, "Communication theory of secrecy systems," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[2] M. E. Smid and D. K. Branstad, "Data encryption standard: past and future," *Proceedings of the IEEE*, vol. 76, no. 5, pp. 550–559, 1988.

[3] J. Daemen and V. Rijmen, "The block cipher rijndael," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 1998, pp. 277–284.

[4] ——, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[5] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[6] W. Stallings, *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, 2017.

[7] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[8] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. Barreto, "MDPC-Mceliece: New Mceliece variants from moderate density parity-check codes," in *2013 IEEE international symposium on information theory*. IEEE, 2013, pp. 2069–2073.

[9] R. J. McEliece, "A public-key cryptosystem based on algebraic," *Coding Thv*, vol. 4244, pp. 114–116, 1978.

[10] G. Liva and H. Bartz, "Protograph-based Quasi-Cyclic MDPC Codes for Mceliece Cryptosystems," in *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*. IEEE, 2018, pp. 1–5.

[11] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[12] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[13] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques*. Springer, 1985, pp. 417–426.

[14] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[15] H. Dinh, C. Moore, and A. Russell, "Mceliece and niederreiter cryptosystems that resist quantum fourier sampling attacks," in *Annual Cryptology Conference*. Springer, 2011, pp. 761–779.

[16] W. Ryan and S. Lin, *Channel codes: classical and modern*. Cambridge university press, 2009.

[17] Q. Guo, T. Johansson, and P. Stankovski, "A key recovery attack on MDPC with CCA security using decoding errors," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2016, pp. 789–815.

[18] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[19] S. Kudekar, T. J. Richardson, and R. L. Urbanke, "Threshold saturation via spatial coupling: Why convolutional ldpc ensembles perform so well over the bec," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 803–834, 2011.

[20] S. Kudekar, T. Richardson, and R. L. Urbanke, "Spatially coupled ensembles universally achieve capacity under belief propagation," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 7761–7813, 2013.

[21] A. J. Felstrom and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 2181–2191, 1999.

[22] K. Engdahl, M. Lentmaier, and K. S. Zigangirov, "On the theory of low-density convolutional codes," in *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*. Springer, 1999, pp. 77–86.

[23] D. J. Costello, L. Dolecek, T. E. Fuja, J. Kliewer, D. G. Mitchell, and R. Smarandache, "Spatially coupled sparse codes on graphs: Theory and practice," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 168–176, 2014.

[24] D. G. Mitchell, M. Lentmaier, A. E. Pusane, and D. J. Costello, "Randomly punctured LDPC codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 408–421, 2015.

[25] H. Bartz and G. Liva, "On decoding schemes for the MDPC-Mceliece cryptosystem," in *SCC 2019; 12th International ITG Conference on Systems, Communications and Coding*. VDE, 2019, pp. 1–6.

[26] T. Fabšič, V. Hromada, P. Stankovski, P. Zajac, Q. Guo, and T. Johansson, "A reaction attack on the QC-LDPC Mceliece cryptosystem," in *International Workshop on Post-Quantum Cryptography*. Springer, 2017, pp. 51–68.

[27] N. Miladinovic and M. P. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1594–1606, 2005.