Exploring Code based cryptography using LDPC Codes

A Thesis Submitted for the Partial Fulfillment of the Requirements for the degree of Master of Technology

in

Computer Science and Technology (Specialization: Computer Science and Engineering)

by

Anupam Chanda
Enrollment no.: 2023CSM003

Under the guidance of

Dr. Abhik Mukherjee

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY
INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR - 711103

Department of Computer Science and Technology
Indian Institute of Engineering Science and Technology, Shibpur,
India - 711103

CERTIFICATE

This is to certify that we have examined the thesis entitled "**Exploring Code based cryptography using LDPC Codes** ", submitted by **Anupam Chanda** (Roll Number: *2023CSM003*), a postgraduate student of **Department of Computer Science and Technology** in partial fulfillment for the award of degree of **Masters in Technology** with specialization of **Computer Science and Engineering**. We hereby accord our approval of it as a study carried out and presented in a manner required for its acceptance in partial fulfillment for the post graduate degree for which it has been submitted. The thesis has fulfilled all the requirements as per the regulations of the institute and has reached the standard needed for submission.

.....
Head of Department
Dr. Apurba Sarkar,
Dept. of C.S.T.,
IIEST, Shibpur.

.....
Supervisor
Dr. Abhik Mukherjee,
Dept. of C.S.T.,
IIEST, Shibpur.

Examiners:
1. .....
2. .....
3. .....

Place: Shibpur
Date:.....

Department of Computer Science and Technology
Indian Institute of Engineering Science and Technology, Shibpur,
India - 711103

CERTIFICATE OF APPROVAL

The forgoing thesis report is hereby approved as a creditable study of "**Exploring Code based cryptography using LDPC Codes** "carried out and presented satisfactorily to warrant its acceptance as a pre-requisite for the Degree of Master of Technology of University. It is understood that by this approval the undersigned do not necessarily approve any statement made, opinion expressed and conclusion drawn there in but approve the progress report only for the purpose for which it is submitted.

Examiners:

1. .....
2. .....
3. .....

Place: Shibpur
Date:.....

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere and deep gratitude to my supervisor, **Dr. Abhik Mukherjee**, Faculty, Department of Computer Science and Technology, for his kind and constant support during my post-graduation study. It has been an absolute privilege to work with Dr. Abhik Mukherjee for my master thesis dissertation. His friendly behavior, valuable advice, critical criticism and active supervision encouraged me to sharpen my research methodology and was instrumental in shaping my professional outlook.

I also want to express my gratitude towards **Dr. Apurba Sarkar**, Professor & Head, Dept. of Computer Science and Technology, IIEST, Shibpur for providing such a wonderful environment filled with continuous encouragement and support. I would also like to thank the entire faculty for their constant encouragement and assistance they have provided me.

Last, but definitely not the least, a very special note of thanks to my **parents and guardians**, who have helped me during this entire period directly or indirectly.

Contents

List of Figures

List of Tables

Chapter 1

Abstract

In the realm of digital communication, ensuring data integrity and security is paramount. This thesis explores the application of Low-Density Parity-Check (LDPC) codes in code-based cryptography, specifically within the McEliece cryptosystem. Traditional cryptographic systems face potential obsolescence with the advent of quantum computing, necessitating the development of post-quantum cryptographic solutions. LDPC codes, known for their sparse parity-check matrices and efficient decoding algorithms, offer a promising alternative to Goppa codes traditionally used in the McEliece cryptosystem.

This study delves into the fundamentals of error detection and correction, the principles of cryptography, and the specific advantages of LDPC codes over Goppa codes. Through a series of experiments, the implementation of LDPC codes in the McEliece cryptosystem is evaluated, highlighting the performance.

Future work will focus on optimizing LDPC code construction and decoding algorithms to further enhance security and performance.

Chapter 2

Basics

2.1 Basics of Error Detection, Error Correction

In the realm of digital communication, errors can creep into data during transmission due to various factors like noise, interference, or hardware malfunctions. To safeguard data integrity, error detection and correction techniques are employed to identify and rectify these errors.

2.1.1 Linear Block Code

Definition: An (n, k) code with length n and 2^k code words is linear iff its 2^k code words form a k dimensional subspace of the vector space of all n-tuples over the field $GF(2)$
This implies that a binary block code is linear iff modulo-2 sum of two code words is also a code word (closure property).
We can form generator matrix $G_{k\times n} = [P_{k\times(n-k)}|I_{k\times k}]_{k\times n}$, where P is the parity-checking part and I is the identity matrix.

2.1.2 Single Error Correcting Code

content.

Hamming Code: Hamming code can detect up to 2 bits errors and can correct up to 1 bit errors. This is also a linear block code.

If number of parity check bits is m
$n = 2^m - 1$;
$k = 2^m - m - 1$;
$n - k = m$;
$t = 1$,
$d_{min} = 3$

2.1.3 Multiple Error Correcting Code

BCH

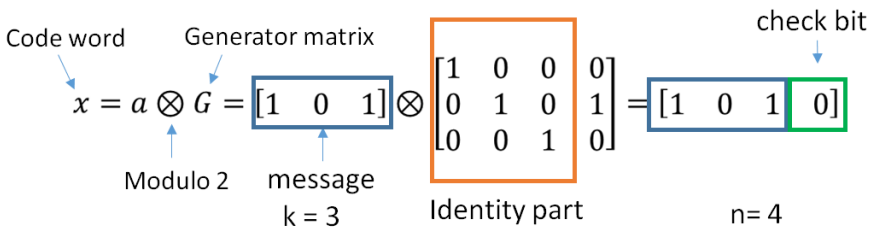

Figure 2.1: A block code (specifically a Hamming code) where redundant bits are added as a block to the end of the initial message [1]

Here we first fix the number of error we want to correct and then calculate other parameters accordingly.

Steps to generate g(x) for BCH Codes

1. Let α is the primitive element
2. Choose prime polynomial of degree m and construct $GF(q^m)$
3. Find $f_i(x)$, the minimal polynomial of α^i for all $i = 1, 2, ..., 2t$
4. The generator polynomial for the t error correcting code is simply

$$g(x) = LCM[f_1(x).f_2(x).....f_{2t}(x)] \tag{2.1}$$

2.1.4 LDPC(Low Density Parity Check)

LDPC codes are defined with respect to their parity check matrices. The term "low density"corresponds to a binary, sparse parity check matrix, $[H_G]_{M\times N}$, associated with this code. Now let's see the definition of the LDPC code originally defined by Gallager in his thesis [5].

Low Density Parity Check(LDPC) codes were first discovered by Robert Gallager in the early 1960s. Unfortunately, his work was mostly ignored by the coding theorists of that time due to the limitations of technology. This continued for 20 years until R. Michael Tanner's work in 1981 [6], where he provided a graph theoretic view of LDPC codes. His work was also ignored for another 14 years until in late 1990s some coding theorists began investigating graph based iteratively decodable codes. One of these pioneers was Dr. David J.C. MacKay. Their results led to the rediscovery of LDPC codes and further generalizations. Though Gallager did not provide specific algebraic methods for construction of good LDPC codes, he proposed a general method for constructing a class of pseudorandom LDPC codes. LDPC codes are a subclass of Block codes, in general.

Definition of Gallager Code: An (N, w_c, w_r) low density parity check code is a code with block length N and a parity check matrix H_G with M rows and N columns where the hamming weight of each row and each column are two fixed integers, w_r and w_c respectively, where $w_r > w_c >= 2$. Since the matrix is sparse, so $w_r << N$ and $w_c << M$.

The total number of 1's in H_G, when counted column wise and row wise, results to $(w_c * N)$ and $w_r * M$ respectively. So, $w_c * N = w_r * M$ i.e., the number of rows , $M = (N * w_c)/w_r$ which must be an integer.

In other words, this structure implies that every code bit participates in w_c parity check sums and each parity check sum involves w_c code bits.

Example: The following H_G is a 4×8 LDPC parity check matrix with $w_c = 2$ and $w_r = 4$.

$$[H_G]_{4\times 8} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}_{4\times 8}$$

Regular and Irregular LDPC Codes: In regular LDPC codes(like Gallager Code) the row weight and column weight are constant and predefined. But we can design LDPC H matrix with different row and column weights also. But this type of designs have to be optimized by experiments.

Decoders of LDPC Codes:

Hard Decision Message Passing(HMP) Decoder: The second type of iterative decoding algorithm that Gallager has proposed [5] for decoding LDPC codes was the Message Passing(MP) algorithm. He proposed both the hard decision and the soft decision variants of an MP algorithm. But before describing these algorithms, we need to know what a generic message passing decoder is.

The idea of a generic message passing decoder can be visualized more easily if we refer to the Tanner graph of the associated code. In the Tanner graph (example: 2.2) of a code, the code bits are called the message nodes and the parity check equations are called the check nodes. Messages are sent from message nodes to check nodes and vice-versa along the edges defined by the Tanner graph of the code.

Algorithm of HMP:

1. All the message nodes send their current values, 0 or 1 as messages to their connected check nodes.

2. On getting the messages from (1), every check nodes calculate an extrinsic message. This message is calculated using the parity-check equations which force all message nodes connected to a particular check node to sum to 0 (mod 2). If the syndrome becomes 0, then the algorithm terminates. Else, goto (3).
3. The message nodes use the messages they get from the check nodes to decide if the bit at their position is a 0 or a 1 by a majority rule.
4. Repeat (1) to (2) until, either exit at (2) or a certain maximum number of iterations has been passed.

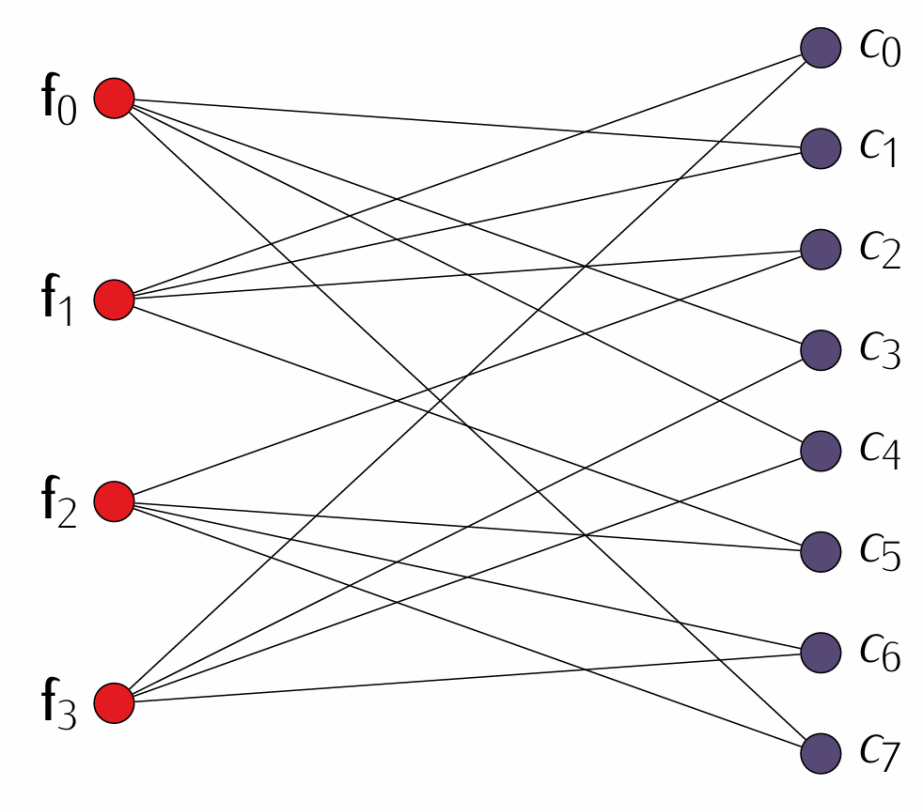

Figure 2.2: Example Tanner graph for the sample LDPC H matrix. c represents message node, f represents check node. 2.1.4

Soft Decision Message Passing Decoder: The soft decision MP algorithm is based on a technique called Belief Propagation. It is also called the Sum-Product algorithm(SPA). It operates with the same underlying structure as the hard decision decoder, except that now the messages propagated are some conditional probabilities, sent as a measure of the belief for each code bit to be a 0 or a 1, given the received vector, r. The exchange of the soft probabilities is termed as Belief Propagation.

We have used the LDPC API [4] to implement and test McEliece crypto system using LDPC.

2.2 Basics of Cryptography

2.2.1 Shared Key Cryptography

[2] Shared Key Cryptography is also known as Symmetric Key Cryptography.In 2.3, an entity, Alice, can send a message to another entity, Bob, over an insecure channel with the assumption that an adversary, Eve, cannot understand the contents of the message by simply eavesdropping over the channel. The original message from Alice to Bob is called plaintext; the message that is sent through the channel is called the ciphertext. To create the ciphertext from the plaintext, Alice uses an encryption algorithm and a shared secret key. To create theplaintext from ciphertext, Bob uses a decryption algorithm and the same secret key. We refer to encryption and decryption algorithms as ciphers. A key is a set of values (numbers) that the cipher, as an algorithm, operates on. Note that the symmetric-key encipherment uses a single key (the key itself may be a set of values) for both encryption and decryption. In addition, the encryption and decryption algorithms are inverses of each other. If P is the plaintext, C is the ciphertext, and K is the key, the encryption algorithm $E_k(x)$ creates the ciphertext from the plaintext; the decryption algorithm $D_k(x)$ creates the plaintext from the ciphertext. We assume that $E_k(x)$ and $D_k(x)$ are inverses of each other: they cancel the effect of each other if they are applied one after the other on the same input. We have, Encryption: $C = E_k(P)$, Decryption: $P = D_k(C)$, in which, $D_k(E_k(x)) = E_k(D_k(x)) = x$

We can prove that the plaintext created by Bob is the same as the one originated by Alice. We assume that Bob creates P_1; we prove that $P_1 = P$.

Alice: $C = E_k(P)$
Bob: $P_1 = D_k(C) = D_k(E_k(P)) = P$

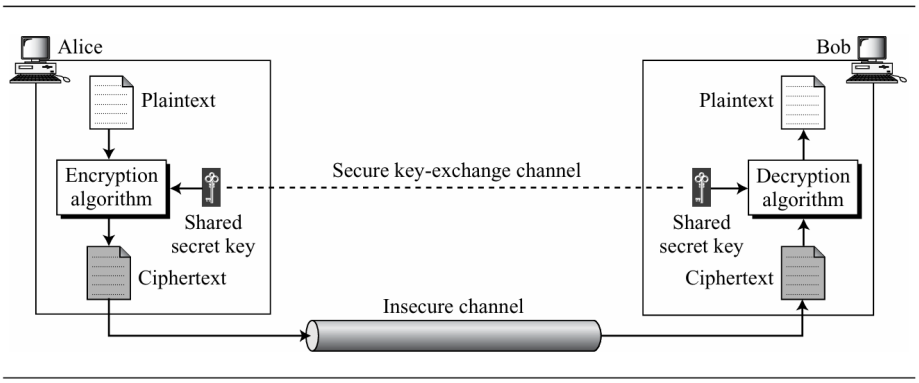

Figure 2.3: General idea of shared-key cipher [2]

2.2.2 Public Key Cryptography

[2] Public Key Cryptography is also known as Asymmetric Key Cryptography. Figure 2.5 shows the general idea of asymmetric-key cryptography as used for encipherment. We

will see other applications of asymmetric-key cryptography in future chapters. The figure shows that, unlike symmetric-key cryptography, there are distinctive keys in asymmetric-key cryptography: a private key and a public key. Although some books use the term secret key instead of private key, we use the term secret key only for symmetric-key and the terms private key and public key for asymmetric key cryptography. We even use different symbols to show the three keys. One reason is that we believe the nature of the secret key used in symmetric-key cryptography is different from the nature of the private key used in asymmetric-key cryptography. The first is normally a string of symbols (bits for example), the second is a number or a set of numbers. In other words, we want to show that a secret key is not exchangeable with a private key; there are two different types of secrets.

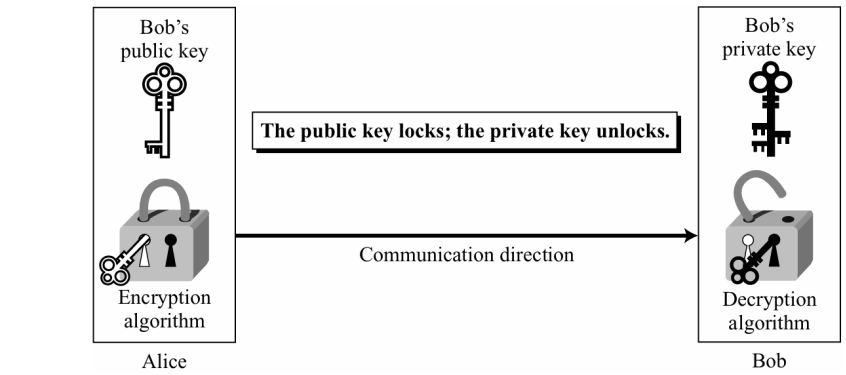

Figure 2.4: Locking and unlocking in public-key cryptosystem [2]

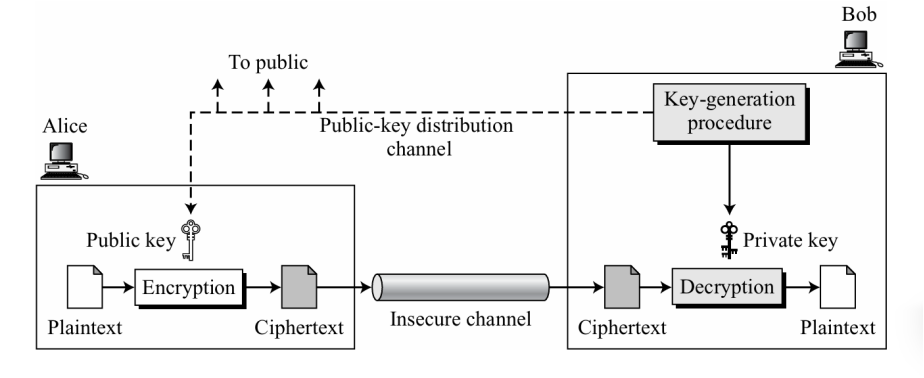

Figure 2.5: General idea of public-key cryptosystem [2]

Figure 2.5 shows several important facts. First, it emphasizes the asymmetric nature of the cryptosystem. The burden of providing security is mostly on the shoulders of the receiver (Bob, in this case). Bob needs to create two keys: one private and one public. Bob

is responsible for distributing the public key to the community. This can be done through a public-key distribution channel. Although this channel is not required to provide secrecy, it must provide authentication and integrity. Eve should not be able to advertise her public key to the community pretending that it is Bob's public key.

2.2.3 Need of Post Quantum Cryptography

The primary difference between post-quantum cryptography and traditional cryptography lies in their resilience against quantum computer attacks.

Traditional Cryptography: The traditional cryptography relies on complex mathematical problems. These problems, like integer factorization and discrete logarithm, are extremely difficult to solve with classical computers. Quantum computers, once sufficiently powerful, can efficiently solve these problems, rendering traditional cryptographic algorithms ineffective. Traditional cryptography is a well-established field that has been used for decades to secure digital communications. This types of cryptographic algorithms are faster and take less computational resources as compared to post quantum cryptography.

Peter Shor showed, in 1994, that using a Quantum Computer, it would be possible to solve the integer factoring problem in polynomial time, something which is believed to be impossible on classical computers. [7]

Post-Quantum Cryptography: This type of cryptography relies on different mathematical problems. These problems are believed to be resistant to attacks from both classical and quantum computers. Post-quantum algorithms are designed to withstand the computational power of quantum computers. In general, post-quantum algorithms are slower and require more computational resources than traditional algorithms. Post-quantum cryptography is a relatively new field that aims to address this threat by developing cryptographic algorithms that are secure against both classical and quantum computers.

Parameter	Traditional Cryptography	Post-Quantum Cryptography
Security against Quantum Computers	Vulnerable	Resistant
Mathematical Basis	Integer factorization, Discrete logarithm	Code-based, Lattice-based multivariate, hash-based
Computational Resource Consumption	Less	More
Maturity	More mature and widely deployed	Less mature, still under development and standardization

Table 2.1: Difference between Traditional and Post-Quantum Cryptography

A group of researchers have proposed a general quantum algorithm for integer factorization and have demonstrated the factor-ing principle for the algorithm on a superconducting quantum processor. The 48-bit integer 261980999226229 in their work is the largest integer factored by the general method in a real quantum system to date. [8]

Chapter 3

Code Based Cryptography

3.1 Introduction

Why code based cryptography? Code-based cryptography is a fascinating field that leverages the mathematical properties of error-correcting codes for secure communication. Unlike traditional cryptographic systems, it offers a potential solution to the threat posed by quantum computers, which could render many current cryptographic algorithms obsolete.

How Does It Work? At its core, code-based cryptography relies on the difficulty of decoding a random linear code. Here's a simplified overview:

Key Generation:

- Private Key: A secret error-correcting code with an efficient decoding algorithm.
- Public Key: A public matrix derived from the private key, which hides its underlying structure.

Encryption:

- The message is encoded as a codeword.
- Random errors are introduced to the codeword, making it appear random.
- The resulting corrupted codeword is sent to the recipient.

Decryption: The recipient uses the private key to efficiently decode the corrupted codeword and recover the original message using the private information.

3.2 Goppa Code in McEliece Cryptosystem

McEliece cryptosystem is also a code based cryptography. Generally Goppa code is used as the coding and decoding technique. In the introduction we have already seen all the processes, key generation, encryption, decryption. Now we will see how keys are generated in McEliece cryptosystem with Goppa code.

Parity Check Matrix(H) Generation:

Goppa Polynomial and Points:

- Define a Goppa polynomial $g(x)$ of degree t over a finite field F_{2^m}
- Choose a set of n distinct elements $L = x_1, x_2, ..., x_n$ from F_{2^m} as the Goppa points.

Constructing the Parity-Check Matrix:

- For each pair $(x_i, g(x))$, create a row in the parity check matrix H as follows: $H_i = [1/g(x_i), x_i/g(x_i), x_i^2/g(x_i)..., x_i^{t-1}/g(x_i)]$
- The complete parity-check matrix H is a $(t \times n)$ matrix over F_{2^m}

Example:

1. The Goppa code $\Gamma(L, g^{'}, z)$ is defined as:
   - $g(z) = (z + \alpha)(z + \alpha^{14}) = z^2 + \alpha^7 z + 1$
   - $L = \alpha^i | 2 <= i <= 13$
2. $m = 4, n = 12, t = 2$
3. $k >= n - mt; k >= 4$
4. $d = 2t + 1; d > 5$
5. Hence, the Goppa code has parameters $[12, >= 4, >= 5]$
6. $$H = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}_{8 \times 12}$$

3.3 LDPC Code in McEliece Cryptosystem

We have discussed on LDPC in section 2.1.4. The way Goppa code is used to form H and G matrices here in LDPC we follow the prescribed processes to form LDPC H and G matrices. And the rest of the processes 3.1 (encryption, decryption)are same.

We have also discussed on forming LDPC code using an example here 4.2

Why LDPC over Goppa: The McEliece cryptosystem, a public-key encryption scheme based on the difficulty of decoding general linear codes, has seen increased interest with the advent of quantum computing. While Goppa codes have traditionally been used, Low-Density Parity-Check (LDPC) codes offer several advantages:

Smaller Key Sizes:

- LDPC codes can be represented using sparse parity-check matrices, which significantly reduces the size of the public key compared to Goppa codes.
- Smaller key sizes translate to lower storage and transmission costs.
- The smaller key sizes and improved performance of LDPC-based McEliece make it suitable for resource-constrained devices like IoT devices.

Improved Performance:

- LDPC codes often have efficient decoding algorithms, leading to faster encryption and decryption processes.
- The reduced computational overhead enables higher throughput, making LDPC-based McEliece systems more suitable for real-world applications.

Enhanced Security:

- While the long-term security of both Goppa and LDPC-based McEliece against quantum attacks is still an active research area, LDPC codes have shown promising resistance to certain quantum attacks.
- LDPC codes offer more flexibility in choosing parameters, allowing for fine-tuning of security levels and performance characteristics.

3.4 Attack on McEliece Cryptosystem

Attack: Same Message Trouble-

- One person sends the same message to the same person more than once.
- We now have knowledge about error vectors so that for all these indices we know mG^*
- If we invert G^* in the column corresponding to these indices, we know the values of m in those indices

Solution:

- Usually Key Encapsulation Methods are used, even in RSA and ECC.
- Main idea is that instead of sending a message using the public-key cryptosystem, we send an encapsulation of a random string.
- Then there are no mathematical properties in the cipher texts anymore and no properties of the message, so that its hash can be used as a symmetric key for symmetric key encryption.

Attack: Guessing the S and P matrices- Attacker got an access generator polynomial g(z). Possible attack is to guess S and P

Attack: Exhaustive codeword comparison by distance metric- Exhaustive codeword comparison attacks pose a significant threat to code-based cryptographic systems. While the fundamental idea is to brute-force the codebook to find the closest codeword to a given ciphertext, several strategies can be employed to mitigate this risk:

Solution:

- A larger code length increases the search space for attackers, making exhaustive search computationally infeasible.
- A higher minimum distance allows for greater error correction capability and enhances the security of the code.

Attack: Syndrome decoding using trial errors-

Attacker uses the following technique by considering G^* is the generator matrix and guessing H^T.
$yH = (mG^* + e)H = (mG^*H^T + eH^T) = eH^T$
If syndrome is zero then the attacker have found the H.

Chapter 4

Experiments and results

4.1 Implementation Details

Through out the experiments we have used LDPC package [4] along with other in build Python libraries. To decode, LDPC Belief Propagation API of the mentioned package [4] is used. The Hard Message Passing algorithm is implemented and experimented the efficiency of it with various

4.2 Choosing a small LDPC code

We have already learnt about McEliece Cryptosystem in section 3.2. How Public Key pair is made, how a message is encrypted and then decrypted, etc. Here we will learn through an example and in the next section 4.3 we will practically experiment by implementing the example in a computer.

Choosing $\mathbf{H}_{m\times n}$(Parity Check) Matrix: We know, coding schemes like LDPC need parity check matrix(H). As we have started experimenting with the LDPC newly, we have started with small LDPC H matrix. As suggested in [3]*Short-length Low-density Parity-check Codes: Construction and Decoding Algorithms by Cornelius Thomas Healy*, we have taken H_a, H_b and I as follows.

$$H_a = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}_{5\times 5} \quad H_b = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}_{5\times 5} \quad I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{5\times 5}$$

And then H matrix is formed as follows.

$$H = \begin{bmatrix} I & 0 & H_a & H_b \\ H_a & H_b & I & 0 \end{bmatrix}_{n\times m}$$

Hence,

$n = 20$

$m = 10$
$k = n - m = 10$

So, the actual parity check matrix becomes as following.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{10\times 20}$$

Choosing $\mathbf{G}_{k\times n}$(Generator) Matrix: To generate code word from message we need generator matrix which is orthogonal to parity check matrix. We have used LDPC API [4] to generate G matrix. Following is the G matrix we have considered.

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{10\times 20}$$

Choosing $\mathbf{S}_{k\times k}$(Scramble) and $\mathbf{P}_{n\times n}$(Permutation) Matrices: Scrambling helps to change the error correcting code structure completely. P permutes the rows. It's not possible to get back the original G from G*($= S \times G \times P$) if we do not have S and P matrices. **S is random and invertible matrix**. **P is random permutation matrix**. We have taken

the following.

$$S = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}_{10\times 10}$$

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{20\times 20}$$

Calculating $\mathbf{G}^*_{k\times n}(= S \times G \times P)$ Matrix: We know,

$$G^* = SGP \tag{4.1}$$

$$G^* = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}_{10\times 20}$$

The Public Key: To encrypt sender needs G*. Hence G* is public here.

The Private Key: Receiver keeps S, P and G with him as they are private information.

So far: We have created the public key and private key. Now its time to take one message and encrypt it with public key, as per McEliece cryptosystem.

Choosing $\mathbf{M}_{1\times k}$(Message) Matrix: We can consider any $_{1\times k}$ matrix as message vector. Lets take the following message matrix.

$$M = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}_{1\times 10}$$

Calculating $(\mathbf{MG}^*)_{1\times n}$ Matrix:

$$MG^* = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}_{1\times 20}$$

Choosing $e_{1\times n}$(Error) Matrix: We studied, an error is added intentionally to obfuscate the error correcting code structure which has certain information like weight(number of 1's), etc. We can take any error vector. Lets take the following error.

$$e = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{1\times 20}$$

Calculating $y_{1\times n}$ Matrix: We know,

$$y = MG^* + e \tag{4.2}$$

$$y = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}_{1\times 20}$$

So far: After creating the public key and private key we have encrypted a message with public key, as per McEliece cryptosystem. Now lets decrypt it at the receiver end. Receiver is that who have created the public key pair. Meaning he has S, P and G matrices.

Calculating $y'_{1\times n}$ Matrix: $y^{'}$ is nothing but yP^{-1}. Multiplying y with P^{-1}

$$y^{'} = yP^{-1} = MSG + e^{'} \tag{4.3}$$

$e^{'}$ is the error which should be found after running decoding algorithm

In our case,

$$\begin{aligned} y^{'} &= yP^{-1} \\ &= MSG + e^{'} \\ &= \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{1\times 20} \end{aligned}$$

Calculating $M^{'}$ by Decoding $y^{'}$: Now it's the time to decode $MSG + e^{'}$ into $M^{'}$. As we are using LDPC, we decode $MSG + e^{'}$ through LDPC algorithm by using one of the methods Hard Message Passing, Belief Propagation). After decoding we will get,

$$M^{'} = MS \tag{4.4}$$

But to get $M^{'}$ two steps are involved. Step1: By decoding $MSG + e^{'}$ we will get a code word(say $C^{'}$). If that code word is valid and if decoding was successful then that valid code word must be MSG, we take help of some technique to find out $M^{'} = MS$ from that valid code word $C^{'} = MSG$.

In our implementation we have used lookup table having valid code word MSG to corresponding MS value.

But there are other efficient techniques (like this [9]) to find message from code word. In our next work we will try to implement such techniques to bring down the look up complexity.

Calculating M the original message: Clearly, the receiver has $M^{'}S$. To get M he multiplies it with S^{-1}.

$$M = M'S^{-1} \tag{4.5}$$

In our case,

$$MS^{-1} = \begin{bmatrix}1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}_{1\times 10}$$
$$\therefore M = \begin{bmatrix}1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}_{1\times 10}$$

This is the final decryption result.

Observations on the distance between original messages and their scrambled versions:

$$Message = \begin{bmatrix}1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}_{1\times 10}$$
$$ScrambledMessage = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1\end{bmatrix}_{1\times 10}$$

Distance = 5

$$Message = \begin{bmatrix}1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1\end{bmatrix}_{1\times 10}$$
$$ScrambledMessage = \begin{bmatrix}1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1\end{bmatrix}_{1\times 10}$$

Distance = 3

$$Message = \begin{bmatrix}0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1\end{bmatrix}_{1\times 10}$$
$$ScrambledMessage = \begin{bmatrix}0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0\end{bmatrix}_{1\times 10}$$

Distance = 7

Hence this observation implies that even if an attacker manages to decode the code word it is hard to find the actual message. Because what he gets is (MS) which is far away from M.

4.3 Experiments

BP Decoder parameters:

- Parity Check Matrix, H matrix 4.2 taken in example.
- error_rate = <Experiments are done using different configurations. In each configuration the error rates are mentioned.>
- max_iter = 10000: maximum number of iterations until which the BP decoder will go through if no valid code word is found.
- bp_method = < Any of two methods(**"product_sum"** or **"minimum_sum"**) are used. In each experiment the method is mentioned.>

Configuration 1:

- Parity Check Matrix(H)= example in section **4.2.**
- error_rate = **0.01**
- max_iter = **10000**
- bp_method = **product_sum**

Configuration 2:

- Parity Check Matrix(H)= example in section **4.2.**
- error_rate = **0.05**
- max_iter = **10000**
- bp_method = **product_sum**

Configuration 3:

- Parity Check Matrix(H)= example in section **4.2.**
- error_rate = **0.08**
- max_iter = **10000**
- bp_method = **product_sum**

Configuration 4:

- Parity Check Matrix(H)= example in section **4.2.**
- error_rate = **0.15**
- max_iter = **10000**
- bp_method = **product_sum**

Configuration 5:

- Parity Check Matrix(H)= example in section **4.2.**
- error_rate = **0.01**
- max_iter = **10000**
- bp_method = **minimum_sum**

Configuration 6:

- Parity Check Matrix(H)= example in section **4.2.**
- error_rate = **0.05**
- max_iter = **10000**
- bp_method = **minimum_sum**

Configuration 7:

- Parity Check Matrix(H)= example in section **4.2.**
- error_rate = **0.08**
- max_iter = **10000**
- bp_method = **minimum_sum**

Configuration 8:

- Parity Check Matrix(H)= example in section **4.2.**
- error_rate = **0.15**
- max_iter = **10000**
- bp_method = **minimum_sum**

Categories of Decoded Word:

- **Category 1**: If the decoder has successfully decoded the input(original code word + errors) word into the original code word.
- **Category 2**: If the decoder has decoded the input(original code word + errors) word into a valid code word but not same as the original code word.
- **Category 3**: If the decoder has not decoded the input(original code word + errors) word into any valid code word even after exhausting the maximum number of iterations allowed.

In example of LDPC we have taken a message 4.2 and after multiplying with the G^* we have added error 4.2 having only one 1(weight=1) in it. Meaning one bit error was added. In experiments we have experimented with different errors having different weight. For all those experiments we have taken the following message matrix.

$$M = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}_{1\times 10}$$

4.3.1 With 1 bit Errors

With the LDPC code we have taken, we have experimented by introducing ${}^{20}C_1 = 20$, 1 bit errors and found different results for Hard Message Passing(HMP) and Belief Propagation(BP) techniques. BP has decoded successfully, every time but HMP could not solve.

4.3.2 With 2 bits Errors

We have experimented by introducing ${}^{20}C_2 = 190$, 2 bits errors also and found different results for Hard Message Passing(HMP) and Belief Propagation(BP) techniques. BP has shown more promising result than HMP.

4.4 Results

Results are here.

Error Rate	Category	Product Sum(count)	Minimum Sum(count)
0.01	1	111	133
	2	19	57
	3	60	0
0.05	1	117	102
	2	7	76
	3	66	12
0.08	1	88	102
	2	5	76
	3	97	12
0.15	1	0	121
	2	0	63
	3	190	6

Table 4.1: Counts of Category of decoded results with respect to BP Method and Error Rate

Chapter 5

Conclusion

In our current work we have studied basics of different error detection and correction algorithms. We have implemented LDPC code based cryptography by forming short length LDPC code. We have experimented the decoders by adding different errors of 1 bit and 2 bits. Among Hard Decision Message Passing and Belief Propagation the later one has shown better results in terms of decoding received, error added code words.

5.1 Achievement

We have successfully implemented the LDPC Code base McEliece cryptography where maximum possible erroneous bit is 1. Our system is successful to encrypt and decrypt messages.

5.2 Limitation

The implemented cryptosystem is using such LDPC parity check matrix that Hard Decision Message Passing is unable to decode the received code words. Even Belief Propagation technique does not show impressive results from 2 bits errors.

5.3 Future Work

Our next target would be to form efficient LDPC code so that Hard Decision Message Passing can decode received code words. We may need to fine-tune the Hard Decision Message Passing algorithm. We will try to implement long LDPC codes to make the security of the cryptosystem more strong.

References

[1] https://en.wikipedia.org/wiki/Error_correction_code

[2] Behrouz A. Forouzan, "Introduction to cryptography and network security ", 1st Edition

[3] Short-length Low-density Parity-check Codes: Construction and Decoding Algorithms by Cornelius Thomas Healy

[4]
```
@ software{Roffe_LDPC_Python_tools_2022,
 author = {Roffe, Joschka},
 title = {{LDPC: Python tools for low density parity check codes}},
 url = {https://pypi.org/project/ldpc/},
 year = {2022}
}
```

[5] R. G. Gallager, Low Density Parity Check Codes,, MIT Press, Cambridge, 1963

[6] R. M. Tanner, "A Recursive Approach to Low Complexity Codes ", IEEE Trans. Information Theory, IT-27: 533-547, Sept 1981

[7] Peter W. Shor (AT&T Research), "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer ", 1996

[8] Bao Yan, Ziqi Tan, Shijie Wei, Haocong Jiang, Weilong Wang, Hong Wang, Lan Luo, Qianheng Duan, Yiting Liu, Wenhao Shi, Yangyang Fei, Xiangdong Meng, Yu Han, Zheng Shan, Jiachen Chen, Xuhao Zhu, Chuanyu Zhang, Feitong Jin, Hekang Li, Chao Song, Zhen Wang, Zhi Ma, H. Wang, Gui-Lu Long, "Factoring integers with sublinear resources on a superconducting quantum processor ", 2022

[9] A. Hasani, L. Lopacinski, S. Büchner, J. Nolte and R. Kraemer, "An Unnoticed Property in QC-LDPC Codes to Find the Message from the Codeword in Non-Systematic Codes," 2019 European Conference on Networks and Communications (EuCNC), Valencia, Spain, 2019, pp. 6-9, doi: 10.1109/EuCNC.2019.8801957.