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 Engineering

by

Anupam Chanda

Enrollment no.: 2023CSM003

Under the supervision 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.

29.7.25

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

Head of the Department
Computer Science & Technology
Indian Institute of Engineering
Science and Technology, Shibpur
Howrah-711 103 (India)

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

Examiners:

1. Sanjay Kumar Saha
2. .....
3. .....

Place: Shibpur

Date: 29/07/25

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. ..... 29/07/25

2. .....

3. .....

Place: Shibpur

Date: 29/07/2025

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.

Anupam Chanda.

.....

Anupam Chanda
Place: IIEST Shibpur
Date: 29/07/2025

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.

Contents

List of Figures

List of Tables

Chapter 1

Basics

1.1 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.

1.1.1 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.

1.1.2 Single Error Correcting Code

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
$n = 2^m - 1$;
$k = 2^m - m - 1$;
$n - k = m$;
$t = 1$,
$d_{min} = 3$

1.1.3 Multiple Error Correcting Code

BCH

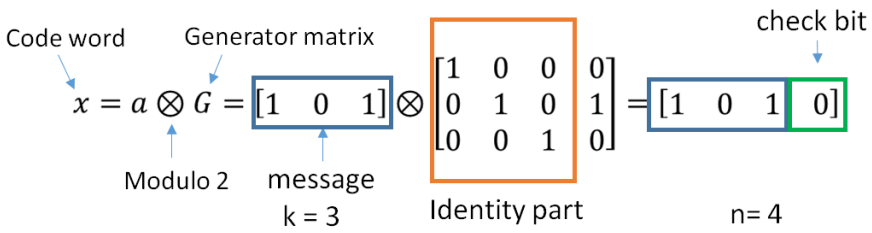

Figure 1.1: A block code (specifically a Hamming code) where redundant bits are added as a block to the end of the initial message [1]

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

$$g(x) = LCM[f_1(x).f_2(x).....f_{2t}(x)] \tag{1.1}$$

1.1.4 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: 1.2) 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.

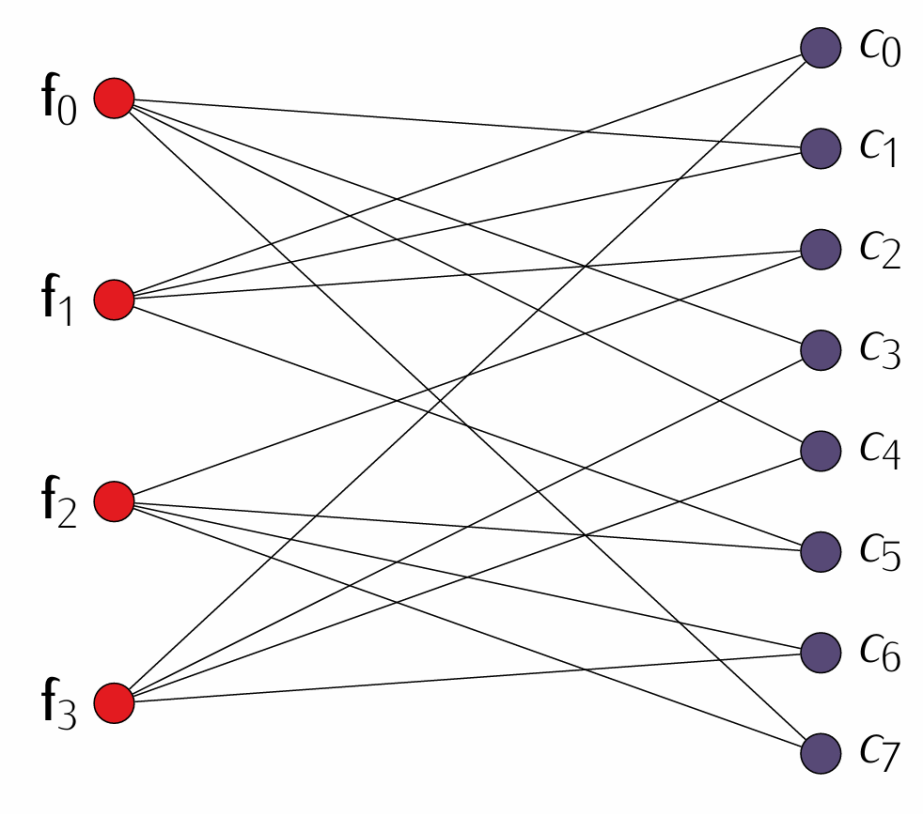

Figure 1.2: Example Tanner graph for the sample LDPC H matrix. c represents message node, f represents check node. 1.1.4, [13]

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.

1.2 Basics of Cryptography

1.2.1 Shared Key Cryptography

Shared Key Cryptography[2] is also known as Symmetric Key Cryptography. In 1.3, 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: $P_1 = D_k(C) = D_k(E_k(P)) = P$

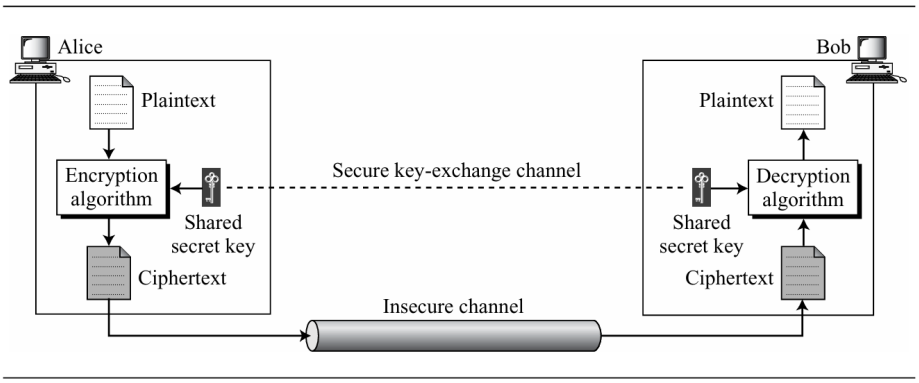

Figure 1.3: General idea of shared-key cipher [2]

1.2.2 Public Key Cryptography

Public Key Cryptography[2] is also known as Asymmetric Key Cryptography. Figure 1.5 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.

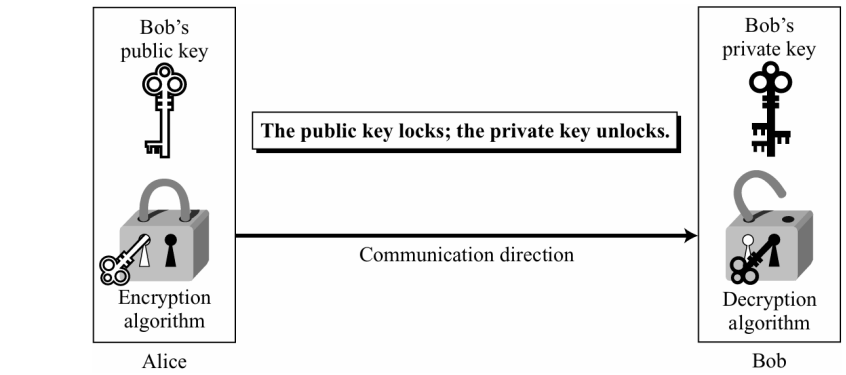

Figure 1.4: Locking and unlocking in public-key cryptosystem [2]

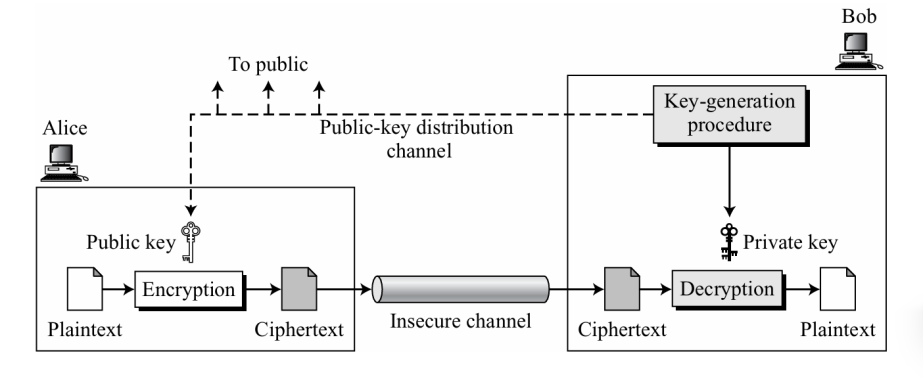

Figure 1.5: General idea of public-key cryptosystem [2]

Figure 1.5 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.

1.2.3 Need for 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 1.2: 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 2

Code Based Cryptography

2.1 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.

2.2 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:
Steps:

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. $$H = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}_{8 \times 12}$$

2.3 LDPC Code in McEliece Cryptosystem

We have discussed on LDPC in section 1.1.4. 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 2.1 (encryption, decryption)are same.

We have also discussed on forming LDPC code using an example here 3.1.2.

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.

2.4 Attacks on McEliece Cryptosystem

Attack 1: Same Message Trouble :-

- One person sends the same message to the same person more than once.
- If we bit-wise XOR the received erroneous code-words we will get error positions. Remaining positions are part of information bits.

Solution:

- Usually Key Encapsulation Methods are used, even in RSA and ECC.
- Extra padding or salt can be added in information bits.

Attack 2: Guessing the S and P matrices :- Attacker got an access generator polynomial g(z) or generator matrix. Possible attack is to guess S and P.

Solution:

- Considering long H matrix. In other words, taking larger code length.

Attack 3: 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 4: 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.

Solution:

- Considering long H matrix. In other words, taking larger code length.

Chapter 3

Experiments and results

3.1 Experiments With Small LDPC Code

3.1.1 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 it's efficiency is experimented.

3.1.2 Choosing a small LDPC code

We have already learnt about McEliece Cryptosystem in section 2.2. 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 3.1.3 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 *Short-length Low-density Parity-check Codes: Construction and Decoding Algorithms by Cornelius Thomas Healy*[3], 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,

$$G^* = SGP \tag{3.1}$$

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

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,

$$y = MG^* + e \tag{3.2}$$

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

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}

$$y^{'} = yP^{-1} = MSG + e^{'} \tag{3.3}$$

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

In our case,

$$\begin{aligned} y^{'} \\ &= yP^{-1} \\ &= MSG + e^{'} \end{aligned}$$

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

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,

$$M^{'} = MS \tag{3.4}$$

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}.

$$M = M'S^{-1} \tag{3.5}$$

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:

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

Distance = 5

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

Distance = 3

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

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.

3.1.3 Experiments

BP Decoder parameters:

- Parity Check Matrix, H matrix (3.1.2), taken in example. Common for all the configurations.
- 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:

- error_rate = **0.01**
- max_iter = **10000**
- bp_method = **product_sum**

Configuration 2:

- error_rate = **0.05**
- max_iter = **10000**
- bp_method = **product_sum**

Configuration 3:

- error_rate = **0.08**
- max_iter = **10000**
- bp_method = **product_sum**

Configuration 4:

- error_rate = **0.15**
- max_iter = **10000**
- bp_method = **product_sum**

Configuration 5:

- error_rate = **0.01**
- max_iter = **10000**
- bp_method = **minimum_sum**

Configuration 6:

- error_rate = **0.05**
- max_iter = **10000**
- bp_method = **minimum_sum**

Configuration 7:

- error_rate = **0.08**
- max_iter = **10000**
- bp_method = **minimum_sum**

Configuration 8:

- 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 3.1.2 and after multiplying with the G^* we have added error 3.1.2 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.

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

3.1.3.1 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.

3.1.3.2 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.

3.1.4 Results

outcomes are shown in tabular format as follows:

Error Rate	Category[3.1.3]	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 3.1: Counts of Category of decoded results with respect to BP Method and Error Rate

3.2 Experiments With NASA-LDPC-(8176, 7156) Code

3.2.1 Implementation Details

Goddard Space Flight Center, NASA published an LDPC code (8176, 7156) standard which was approved on 19th March-2008. The purpose of this standard is to establish a common channel coding protocol for bandwidth efficient spacecraft communications.

The LDPC code considered in this specification is a member of a class of codes called Quasi Cyclic codes. The construction of these codes involves juxtaposing smaller circulants (or cyclic sub-matrices) to form a larger parity check or base matrix.

Circulant Matrix: For example following figure 3.1 is a 5×5 circulant.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}_{5\times5}$$

Figure 3.1: A Circulant matrix

We can see that every row in figure 3.1 is one bit right shifted from its previous row.

Last bit is wrapped around to the beginning bit. In this example the weight of each row is 2. The entire circulant is uniquely identifiable by the first row.

Quasi-Cyclic parity check matrix: Following is an example of a quasi-cyclic parity check matrix. In this case, a quasi-cyclic 10×25 matrix is formed by an array of 2×5 circulant submatrices of size 5×5. To unambiguously describe this matrix, only the position of the 1's in the first row of every circulant submatrix and the location of each submatrix within the base matrix is needed.

$$\left[\begin{array}{ccccc|ccccc|ccccc|ccccc|ccccc}
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1
\end{array}\right]_{10\times25}$$

Figure 3.2: A Quasi-Cyclic parity check matrix

As mentioned in the paper, out of 7156×8176 only 7154×8176 part is used to implement the code standard for encoding and decoding.

3.2.2 NASA-LDPC-(8176, 7156) Specifications

Parity Check Matrix: Following is the base of the parity check matrix of the standard. The corresponding circulants are specified in appendix A.1 .

$$\begin{bmatrix}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} & A_{1,5} & A_{1,6} & A_{1,7} & A_{1,8} & A_{1,9} & A_{1,10} & A_{1,11} & A_{1,12} & A_{1,13} & A_{1,14} & A_{1,15} & A_{1,16} \\
A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} & A_{2,5} & A_{2,6} & A_{2,7} & A_{2,8} & A_{2,9} & A_{2,10} & A_{2,11} & A_{2,12} & A_{2,13} & A_{2,14} & A_{2,15} & A_{2,16}
\end{bmatrix}_{2\times16}$$

Figure 3.3: Base Parity Check Matrix of the (8176, 7156) LDPC code [10]

Each $A_{i,j}$ is a 511 x 511 circulant. The row weight of the each of the 32 circulants is 2, i.e. there are two 1's in each row. The total row weight of each row in the parity check matrix is 2 x 16 or 32. The column weight of each circulant is also 2, i.e. there are two 1's in each column. The total weight of each column in the parity check matrix is 2 x 2 or 4. The position of the 1's in each circulant is defined in table A.1.

Generator Matrix: Following is the base of the generator matrix of the standard. The corresponding circulants are specified in appendix A.2 .

$$\begin{bmatrix}
I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{1,1} & B_{1,2} \\
0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{2,1} & B_{2,2} \\
0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{3,1} & B_{3,2} \\
0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{4,1} & B_{4,2} \\
0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{5,1} & B_{5,2} \\
0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{6,1} & B_{6,2} \\
0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{7,1} & B_{7,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & B_{8,1} & B_{8,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & B_{9,1} & B_{9,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & B_{10,1} & B_{10,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & B_{11,1} & B_{11,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & B_{12,1} & B_{12,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & B_{13,1} & B_{13,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & B_{14,1} & B_{14,2}
\end{bmatrix}$$

Figure 3.4: Systematic Circulant Generator Matrix of 14 x 16 Circulants [10]

3.2.3 Experiments

Error Correction Capability(Approximate): To check error correction capability of the NASA-LDPC(8176, 7156)[10] using BP decoding[4] we have run the BP decoding algorithm using 3 different sets of error vectors having 3 different bit error probabilities.

Using numpy.random.rand(n) OR cupy.random.rand(n), n length array is made and in each position 0 or 1 is chosen as per the bit error probability. If the i^{th} element has the value greater than the specified bit error probability (refer 3.2.3) then 1 otherwise 0. In each set we have taken 1000 samples meaning total 3000 samples. In each set there is a range of number of error bits as outcomes.

For each sample error vector, we have run the BP decoding algorithm[4] with all zero message(refer 3.2.3) and tested the ability of decoding of NASA-LDPC(8176, 7156). For all 3000 samples we have concluded the result in table 3.4. The result is grouped by total error bits and the BP decoding result category(discussed here 3.1.3) .

Bit error probability
The probability of being erronious of a bit in code-word while being transmitted.

All zero message
All bits of a binary version of a message are zeros.

Set No.	Bit Error Probability	Mean of number of error bits	Median of number of error bits
1	82/8176	82.104	82
2	72/8176	72.483	72
3	92/8176	92.583	92.5

Table 3.3: No of error bits with respect to bit error probability

Table 3.4: Count of BP decoded categories for certain error bits

Total number of Error Bits	Decoded Category [3.1.3]	Count
49	1	4
50	1	1
51	1	2
52	1	3
53	1	2
54	1	6
55	1	9
56	1	11
57	1	8
58	1	12
59	1	20
60	1	13
61	1	18
62	1	33
63	1	25
64	1	30
65	1	35
66	1	49
67	1	37
68	1	56
69	1	79
70	1	70
71	1	69
72	1	86
73	1	82
74	1	81
75	1	79
76	1	97
77	1	86
78	1	84
79	1	91
80	1	87
80	3	4
81	1	84
82	1	80
82	3	2
83	1	75
83	3	3
84	1	112
84	3	6
85	1	80
85	3	5
86	1	64
86	3	10
87	1	56
87	3	10
88	1	73
88	3	18
89	1	58
89	3	17
90	1	52
90	3	26
91	1	46
91	3	36
92	1	35

92	3	46
93	1	29
93	3	36
94	1	23
94	3	36
95	1	9
95	3	47
96	1	11
96	3	48
97	1	5
97	3	54
98	1	6
98	3	39
99	1	3
99	3	30
100	1	2
100	3	42
101	3	32
102	3	24
103	3	24

104	3	18
105	3	19
106	3	19
107	3	15
108	3	6
109	3	17
110	3	12
111	3	8
112	3	4
113	3	7
114	3	5
115	3	3
117	3	1
119	3	1
121	3	2

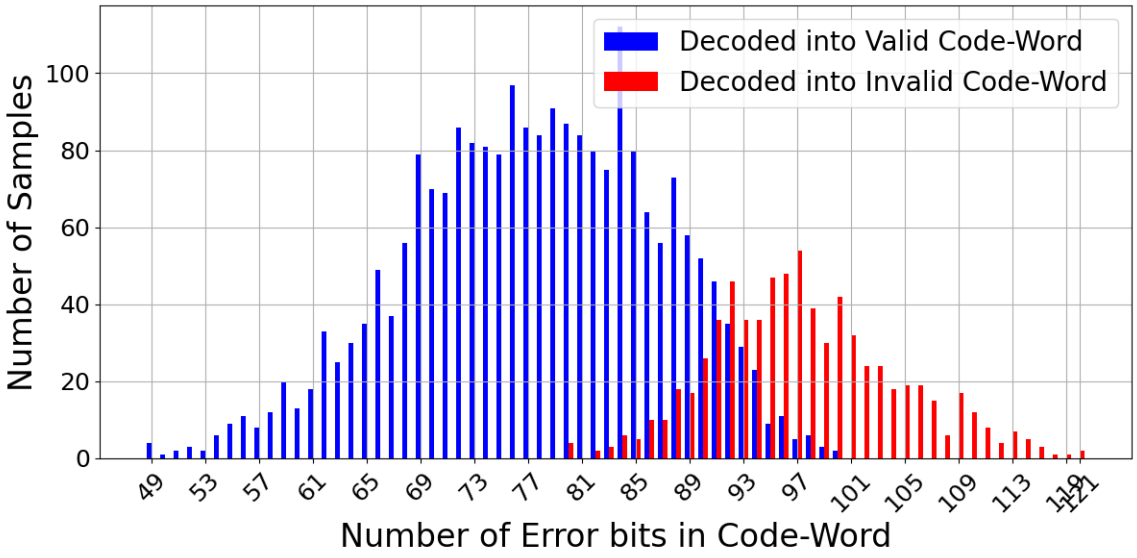

Figure 3.5: Count of BP decoded categories for certain error bits

From the table and graph, it is very clear that transmitted code-word with 81 or lesser error bits has highest chance of being corrected. Hence we have further investigated by pinpointing the error bits around 81 bits.

SNR(dB) vs Number of error bits: We have analyzed the approximate maximum SNR(dB) in a channel corresponding to the maximum number of error bits upto which the NASA LDPC decoder can correct using BP decoding package[4]. Results are shown in the table 3.5 and graph 3.6 .

Refer the Matlab code A.3 used for testing SNR(dB) vs Error Bits.

Following is the outcome in tabular format.

SNR(dB)	Mean of number of error bits	Median(# error bits)
17	195.042	195
17.25	169.426	169
17.50	145.492	146
17.75	122.776	123
18	102.804	102
18.25	84.716	85
18.29	82.439	82
18.30	81.346	81
18.50	68.6	68
18.75	54.812	55
19	43.051	43
19.25	33.042	33
19.50	25.042	25
19.75	17.864	18
20	12.58	12
20.25	8.83	9
20.50	5.82	6
20.75	3.673	4
21	2.364	2
21.25	1.298	1
21.50	0.655	1
21.75	0.363	0
22	0.15	0
22.25	0.075	0
22.50	0.029	0
22.75	0.006	0

Table 3.5: SNR(dB) VS Number of Error Bits

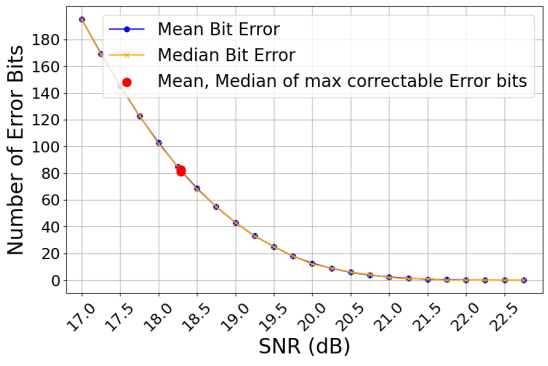

Figure 3.6: Mean and Median of Error Bits VS SNR

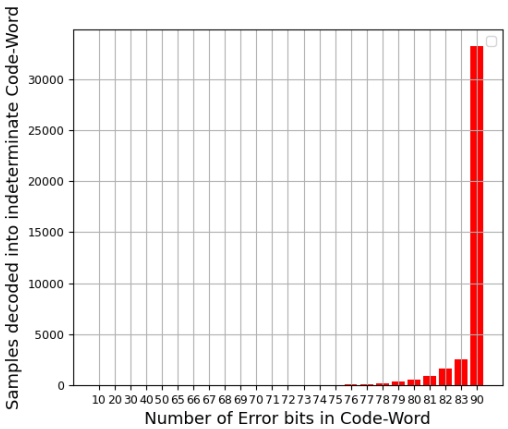

Figure 3.7: Count of BP decoded into indeterminate(category 3) code-word

For each error bits 100000 samples have been checked. To do so we have generated 10000 different information/actual message then generated code-words for them. After that same number of bits in same positions are introduced in each of those code-words and then decoded with BP. Same procedure is followed for different number of error bits. The result is showcased in the form of table 3.6 and graph 3.7 .

Total number of Error Bits	Count
10	0
20	0
30	0
40	0
50	0
65	1
66	1
67	1
68	1
69	2
70	2
71	8
72	2
73	9
74	21
75	24
76	47
77	92
78	163
79	317
80	562
81	947
82	1635
83	2561
90	33204

Table 3.6: Count of BP decoded indeterminate(category 3) code-word for certain error bits

What if a wrong but valid code-word(category 2) is generated by decoder? As we know that BP decoder can and may generate a valid code-word, from a given erroneous code-word, but that's not the actual code-word which was generated during encoding. In that case we don't have any mechanism to understand whether the receiver have received the correct message from sender or not. To mitigate this problem we have introduced hashing technique(sha256[11]) which generates 256 bits of hashed value.

In NASA(8176, 7156), 7154[10] bits are for information. As we are going to keep hash of raw message/data which will consume 256 bits of the whole information bits hence we have reserved 256 bits at the end of the information bits. Hence, 6898 bits remains.

To encode raw text message/data of sender into binary stream, we are following ASCII table which requires 8 bits for each character. So, out of 6898 bits we can use 6896 bits at max for keeping sender's raw message.

Remaining 2 bits can be used as salt to the encoder so than same message does not get converted into same code-word.

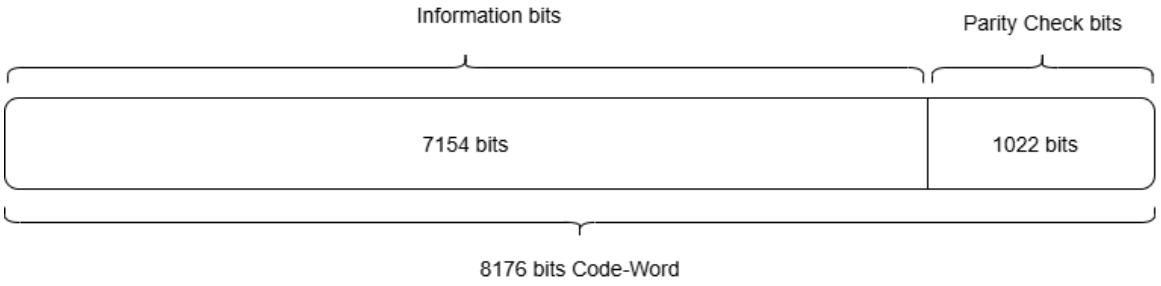

Figure 3.8: Breakup of Code-Word bits

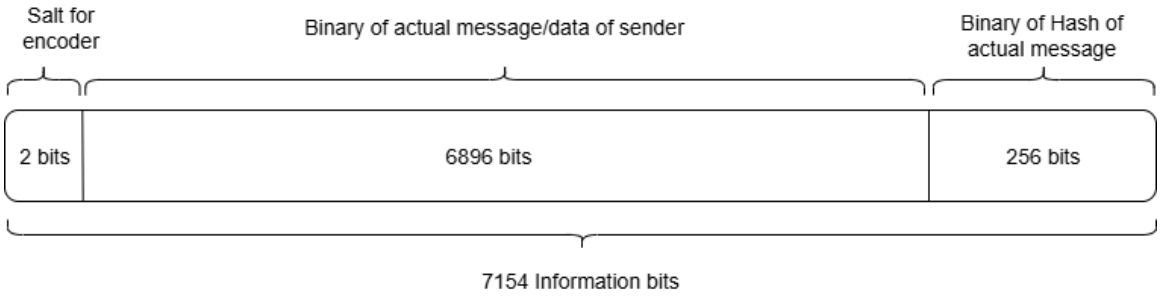

Figure 3.9: Breakup of Information bits

Lets see how a message is getting scrambled. The same message(7154 bits) (refer B.1) is multiplied by different scramble matrices (refer B.4, B.5, B.6) to see how the message is scrambled in different scenarios.

The observed hamming distance between the message B.1 and three different scrambled versions of the message are 3599, 3572, 3611 respectively. Hence it is confirmed that the trace of original information is nullified by scrambling the message. It is also obvious that finding these huge number of positions is matter of many years. Hence, it is highly unlikely that an intruder will be able to revert back to original message from a scrambled version.

It is also observed that the hamming distances between the scrambled versions of the message are 3609, 3594, 3563 respectively for Scrambled Message 1 and Scrambled Message 2, Scrambled Message 1 and Scrambled Message 3, Scrambled Message 2 and Scrambled Message 3.

To check whether same scramble matrix can scramble different message with same level of scrambling power, we have scrambled three different messages(B.1, B.2, B.3) with one scramble matrix B.4 . The distances are 3599, 3575, 3594 respectively.

3.2.4 Complexity Analysis

Complexity analysis of NASA-LDPC-(8176, 7156) Code: Graph 3.10 which is produced by taking 100000 erroneous code-words having 80 error bits depicts the variation of frequency of iterations. The reason behind analyzing 80 error bits code-words is that our implemented scheme uses fixed 80 error bits.

We see that maximum number of 80 bits error are corrected in 9 iterations. Which is a

good sign for this scheme to be fit in real time message transfer for such heuristic decoding.

From graph 3.11 we see that average number of BP iteration needed to decode certain number of error bits increases almost linearly up to 70 error bits. Then from 70 to 83 it starts accelerating. And from 83 onward it grows almost exponentially.

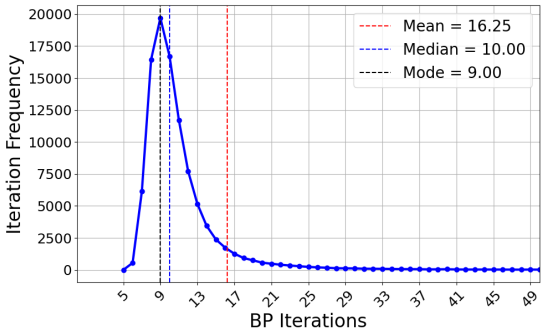

Figure 3.10: Iteration Frequency VS BP Decoder Iteration for 80 error bits

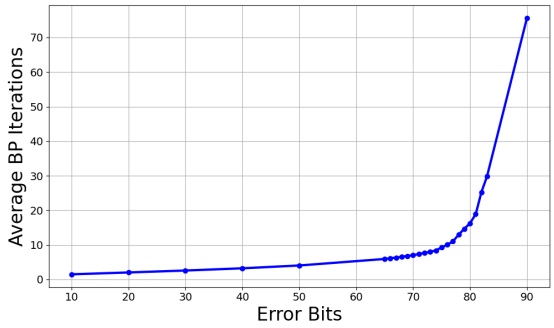

Figure 3.11: Average BP Iterations VS Error Bits

Why choose 80 error bits: The more error bits are pushed into code-word the more tough it is for an attacker to find the error positions to correct them. Hence pushing more errors will make the encryption stronger. But at the same time, more errors lowers the success rate of decoding. Hence the decision of choosing number of error bits depends on the trade off between strength of cryptography vs decoding success rate.

So, to get best success rate we may choose any number of error bits up to 70. But if we choose anything 83 bits onward, the decoding success rate becomes very poor. Choosing between 70 and 83 would give as good balance of strength of cryptography and decoding success rate. By compromising little in decoding success rate we can achieve more strength cryptographycally if we choose anything between 70 and 83. We have considered 80 error bits.

3.2.5 Requirements for Implementation

We have applied the encoder and decoder with the help of the generator matrix which is provided by NASA. Meaning, throughout our application the G and H matrix is fixed. Our work is not much with encoder, decoder or channel. Rather, we are focused on cryptography by leveraging the capability of coding theory. Hence, to implement the McEliece Cryptosystem we have used the given G, H. Apart from these we need S, P as secrete information. We have already discussed the idea here 2.1 and 3.1.2 .

3.2.6 Prototype Implementation

As McEliece cryptosystem with LDPC uses heuristic decoding, it may generate category 2 and category 3 results. Hence there must the a mechanism to re-send the code-word. But in computer communication(networking) re-send mechanism is present in transport layer.

But creating a new transport layer security with this McEliece using LDPC is very much time consuming and hence out of the scope of this MTech study.

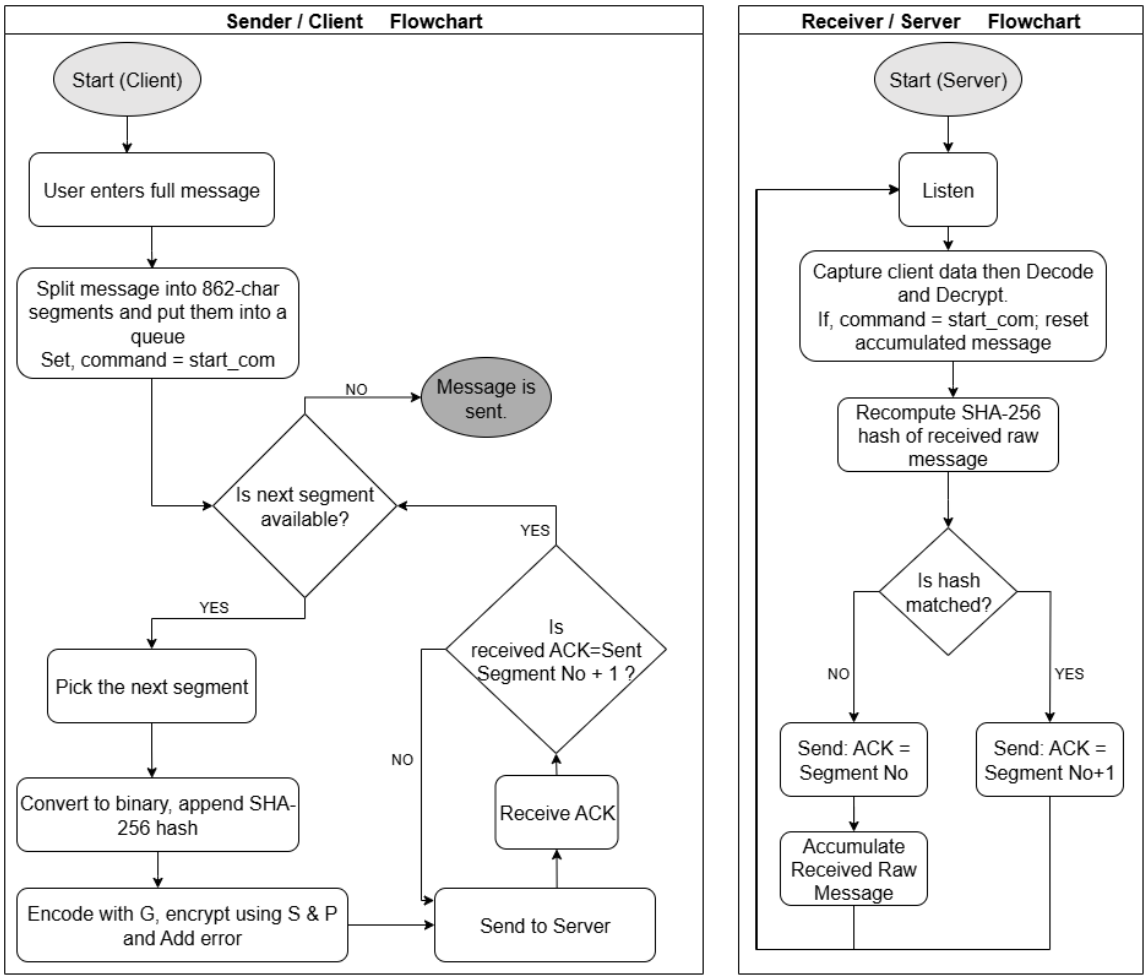

Figure 3.12: Implementation Flowchart

Just to realize the ability of our proposed crypto-system and to test how it is performing in message exchange, we have implemented the crypto-system in application layer by creating one server and one client. Client acts as sender and Server acts as receiver.

Sender split the entered message by user into segments with 862 characters. As 6896 bits = 862 Bytes = 862 Characters. Each segment has a segment number starting from 1. Salt, Raw Message, Hash of Raw Message are encapsulated and sent to Receiver and waits for acknowledgment. Acknowledgment is a number. First segment is sent by setting a parameter ”command” = ”start_com”.

After receiving a message segment, the Receiver decodes and decrypts the message and calculates the hash of the raw message portion. If the calculated hash value is matched with the embedded hash value then it sends (received segment number + 1) otherwise (received segment number) as acknowledgment number. The receiver keeps on accumulating the

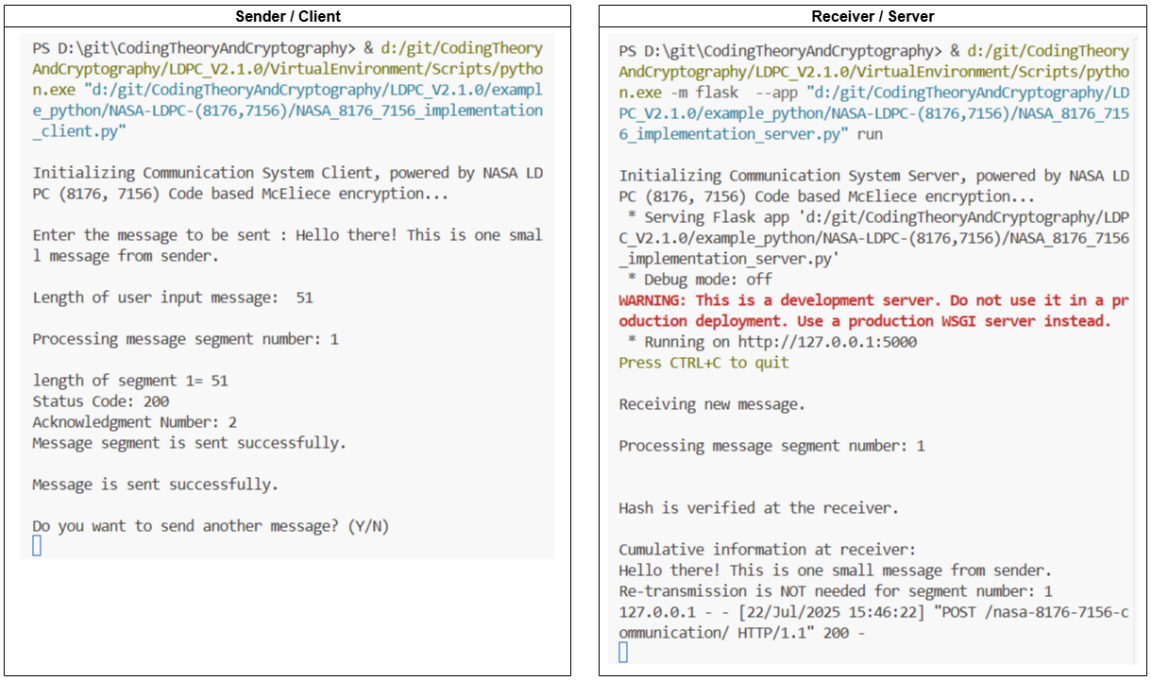

Figure 3.13: Implementation Flowchart

message segments. When the receiver receives a segment, it checks for value of ”command” parameter and if the value is ”start_com” it resets the accumulated message.

Chapter 4

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 as well as long length LDPC code. We have experimented the decoders by adding different errors of 1 bit and 2 bits in short length LDPC and 10 to 90 bits in long length LDPC. Among Hard Decision Message Passing and Belief Propagation the later one has shown better results in terms of decoding.

4.1 Achievement

We have successfully implemented the LDPC Code base McEliece cryptography with very long length LDPC block. We have successfully implemented a prototype in application layer to send message from a client(sender) to a server(receiver).

4.2 Limitation

- In the implemented cryptosystem the LDPC decoder can decode upto limited(around 100) error bits.
- Scheme is implemented in application layer but feasibility of using such scheme where re-transmission is required is less in application layer.
- Fixed number of errors are pushed in each encryption.

4.3 Future Scope of Work

- In our implementation scheme we have fixed the number of error bits which are being pushed into code-words. But pushing different number of error bits will give another layer of security as an attacker would not know how many number of errors he should try to correct. It would add another level of difficulty for sure. So, randomizing the error bits would be a good piece of work.

- For long block length in other words long length keys, Goppa Code's complexity arises tremendously. But LDPC works well. But this could be tried to analyze and compare the complexity of our proposed scheme with another one where both Goppa and LDPC is used together to get the strength of Goppa and decoding simplicity of LDPC [12].
- As this heuristic based decoding needs re-transmission of message, we need some transmission related mechanism in place. But application layer does not provide such inbuilt mechanism. But transport layer has that. So this scheme can be implemented in transport layer.

References

[1] Picture from Wikipedia 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 https://core.ac.uk/download/pdf/30267646.pdf

[4] Roffe, Joschka, "LDPC: Python tools for low density parity check codes", https://pypi.org/project/ldpc/, 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.

[10] GODDARD TECHNICAL STANDARD, GSFC-STD-9100; Goddard Space Flight Center, Greenbelt, MD 20771; Approved: 03-19-2008; Link 1: https://standards.nasa.gov/sites/default/files/standards/GSFC/A/0/gsfc-std-9100a-04-10-2017.pdf; Link 2: https://standards.nasa.gov/standard/GSFC/GSFC-STD-9100

[11] Tony Hansen and Donald E. Eastlake 3rd, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF", RFC 6234, doi 10.17487/RFC6234, https://www.rfc-editor.org/info/rfc6234, May 2011

[12] Pedro Branco, Paulo Mateus, Carlos Salema, André Souto, "Using Low-Density Parity-Check codes to improve the McEliece cryptosystem", 2019

[13] Monosij Maitra, Abhik Mukherjee, "Studies on Low Density Parity Check Codes and Decoder Dynamics at Waterfall SNR Region", ME Thesis 2014

Appendix A

NASA(8176, 7156) specifications

A.1 Circulants of Parity Check Matrix

Circulant	1's position in 1st row of circulant	Absolute 1's position in 1st row of Parity Check Matrix
$A_{1,1}$	0, 176	0, 176
$A_{1,2}$	12, 239	523, 750
$A_{1,3}$	0, 352	1022, 1374
$A_{1,4}$	24, 431	1557, 1964
$A_{1,5}$	0, 392	2044, 2436
$A_{1,6}$	151, 409	2706, 2964
$A_{1,7}$	0, 351	3066, 3417
$A_{1,8}$	9, 359	3586, 3936
$A_{1,9}$	0, 307	4088, 4395
$A_{1,10}$	53, 329	4652, 4928
$A_{1,11}$	0, 207	5110, 5317
$A_{1,12}$	18, 281	5639, 5902
$A_{1,13}$	0, 399	6132, 6531
$A_{1,14}$	202, 457	6845, 7100
$A_{1,15}$	0, 247	7154, 7401
$A_{1,16}$	36, 261	7701, 7926
$A_{2,1}$	99, 471	99, 471
$A_{2,2}$	130, 473	641, 984
$A_{2,3}$	198, 435	1220, 1457
$A_{2,4}$	260, 478	1793, 2011
$A_{2,5}$	215, 420	2259, 2464
$A_{2,6}$	282, 481	2837, 3036
$A_{2,7}$	48, 396	3114, 3462
$A_{2,8}$	193, 445	3770, 4022
$A_{2,9}$	273, 430	4361, 4518
$A_{2,10}$	302, 451	4901, 5050
$A_{2,11}$	96, 379	5206, 5489
$A_{2,12}$	191, 386	5812, 6007
$A_{2,13}$	244, 467	6376, 6599
$A_{2,14}$	364, 470	7007, 7113
$A_{2,15}$	51, 382	7205, 7536
$A_{2,16}$	192, 414	7857, 8079

Table A.1: Specification of Circulants [10]

A.2 Circulants of Generator matrix

Circulant	1st row of circulant
$B_{1,1}$	55BF56CC55283DFEEFEA8C8CFF04E1EBD9067710988E25048D67525 426939E2068D2DC6FCD2F822BEB6BD96C8A76F4932AAE9BC53AD20 A2A9C86BB461E43759C
$B_{1,2}$	6855AE08698A50AA3051768793DC238544AF3FE987391021AAF6383A6 503409C3CE971A80B3ECE12363EE809A01D91204F1811123EAB867D3 E40E8C652585D28
$B_{2,1}$	62B21CF0AEE0649FA67B7D0EA6551C1CD194CA77501E0FCF8C8586 7B9CF679C18BCF7939E10F8550661848A4E0A9E9EDB7DAB9EDABA 18C168C8E28AACDDEAB1E
$B_{2,2}$	64B71F486AD57125660C4512247B229F0017BA649C6C11148FB00B7080 8286F1A9790748D296A593FA4FD2C6D7AAF7750F0C71B31AEE5B400 C7F5D73AAF00710
$B_{3,1}$	681A8E51420BD8294ECE13E491D618083FFBBA830DB5FAF330209877 D801F92B5E07117C57E75F6F0D873B3E520F21EAFD78C1612C622811 1A369D5790F5929A
$B_{3,2}$	04DF1DD77F1C20C1FB570D7DD7A1219EAECEA4B2877282651B0FF E713DF338A63263BC0E324A87E2DC1AD64C9F10AAA585ED6905946 EE167A73CF04AD2AF9218
$B_{4,1}$	35951FEE6F20C902296C9488003345E6C5526C5519230454C556B8A04F C0DC642D682D94B4594B5197037DF15B5817B26F16D0A3302C0938341 2822F6D2B234E
$B_{4,2}$	7681CF7F278380E28F1262B22F40BF3405BFB92311A8A34D084C08646 4777431DBFDDD2E82A2E6742BAD6533B51B2BDEE0377E9F6E63DC A0B0F1DF97E73D5CD8
$B_{5,1}$	188157AE41830744BAE0ADA6295E08B79A44081E111F69BBE7831D07 BEEBF76232E065F752D4F218D39B6C5BF20AE5B8FF172A7F1F680E6 BF5AAC3C4343736C2
$B_{5,2}$	5D80A6007C175B5C0DD88A442440E2C29C6A136BBCE0D95A58A83B 48CA0E7474E9476C92E33D164BFF943A61CE1031DFF441B0B175209B 498394F4794644392E
$B_{6,1}$	60CD1F1C282A1612657E8C7C1420332CA245C0756F78744C807966C3E 1326438878BD2CCC83388415A612705AB192B3512EEF0D95248F7B73E 5B0F412BF76DB4
$B_{6,2}$	434B697B98C9F3E48502C8DBD891D0A0386996146DEBEF11D4B83303 3E05EDC28F808F25E8F314135E6675B7608B66F7FF3392308242930025 DDC4BB65CD7B6E
$B_{7,1}$	766855125CFDC804DAF8DBE3660E8686420230ED4E049DF11D82E357 C54FE256EA01F5681D95544C7A1E32B7C30A8E6CF5D0869E754FFDE 6AEFA6D7BE8F1B148
$B_{7,2}$	222975D325A487FE560A6D146311578D9C5501D28BC0A1FB48C9BDA 173E869133A3AA9506C42AE9F466E85611FC5F8F74E439638D66D2F00 C682987A96D8887C

Circulant	1st row of circulant
$B_{8,1}$	14B5F98E8D55FC8E9B4EE453C6963E052147A857AC1E08675D99A308 E7269FAC5600D7B155DE8CB1BAC786F45B46B523073692DE745FDF1 0724DDA38FD093B1C
$B_{8,2}$	1B71AFFB8117BCF8B5D002A99FEEA49503C0359B056963FE5271140E 626F6F8FCE9F29B37047F9CA89EBCE760405C6277F329065DF21AB3 B779AB3E8C8955400
$B_{9,1}$	0008B4E899E5F7E692BDCE69CE3FAD997183CFAEB2785D0C3D9CA E510316D4BD65A2A06CBA7F4E4C4A80839ACA81012343648EEA8DB BA2464A68E115AB3F4034
$B_{9,2}$	5B7FE6808A10EA42FEF0ED9B41920F82023085C106FBBC1F56B567A 14257021BC5FDA60CBA05B08FAD6DC3B0410295884C7CCDE0E5634 7D649DE6DDCEEB0C95E
$B_{10,1}$	5E9B2B33EF82D0E64AA2226D6A0ADCD179D5932EE1CF401B336449 D0FF775754CA56650716E61A43F963D59865C7F017F5383051430664982 2CAA72C152F6EB2
$B_{10,2}$	2CD8140C8A37DE0D0261259F63AA2A420A8F81FECB661DBA5C62D F6C817B4A61D2BC1F068A50DFD0EA8FE1BD387601062E2276A4987A 19A70B460C54F215E184
$B_{11,1}$	06F1FF249192F2EAF063488E267EEE994E7760995C4FA6FFA0E424182 5A7F5B65C74FB16AC4C891BC008D33AD4FF97523EE5BD14126916E0 502FF2F8E4A07FC2
$B_{11,2}$	65287840D00243278F41CE1156D1868F24E02F91D3A1886ACE906CE74 1662B40B4EFDFB90F76C1ADD884D920AFA8B3427EEB84A759FA02E 00635743F50B942F0
$B_{12,1}$	4109DA2A24E41B1F375645229981D4B7E88C36A12DAB64E91C764CC4 3CCEC188EC8C5855C8FF488BB91003602BEF43DBEC4A621048906A2 CDC5DBD4103431DB8
$B_{12,2}$	2185E3BC7076BA51AAD6B199C8C60BCD70E8245B874927136E6D8DD 527DF0693DC10A1C8E51B5BE93FF7538FA138B335738F4315361ABF8 C73BF40593AE22BE4
$B_{13,1}$	228845775A262505B47288E065B23B4A6D78AFBDDB2356B392C692EF 56A35AB4AA27767DE72F058C6484457C95A8CCDD0EF225ABA56B76 57B7F0E947DC17F972
$B_{13,2}$	2630C6F79878E50CF5ABD353A6ED80BEACC7169179EA57435E44411 BC7D566136DFA983019F3443DE8E4C60940BC4E31DCEAD514D755A F95A622585D69572692
$B_{14,1}$	7273E8342918E097B1C1F5FEF32A150AEF5E11184782B5BD5A1D8071 E94578B0AC722D7BF49E8C78D391294371FFBA7B88FABF8CC03A62 B940CE60D669DFB7B6
$B_{14,2}$	087EA12042793307045B283D7305E93D8F74725034E77D25D3FF043AD C5F8B5B186DB70A968A816835EFB575952EAE7EA4E76DF0D5F09759 0E1A2A978025573E

Table A.2: Table of Circulants for the Generator Matrix [10]

A.3 Matlab Code

:

```
% Parameters
M = 4; % Modulation order (4-PSK)
k = log2(M); % Bits per symbol
numBits = 8176; % Number of bits to transmit
%EbNoVec = 0:.5:10; % Eb/No values in dB
%EbNoVec = 9.12 * ones(1, 1000); % When EbNo = 10^(EbNoVec(i)/10);
%at 9.12 the channel produces 82(mean) no of errors

EbNoVec = 18.25 * ones(1, 1000); % When EbNo = 10^(EbNoVec(i)/20);
%at 23.50 most of the time the channel produces no error
%at 18.25 the channel produces 82(mean) no of errors

% Preallocate memory for BER results
berSim = zeros(size(EbNoVec));
numErrors = zeros(size(EbNoVec));

% Generate random data bits
dataBits = randi([0 1], numBits, 1);

for i = 1:length(EbNoVec)
% Calculate Eb/No in linear scale
EbNo = 10^(EbNoVec(i)/20);

% Modulate the data
modData = pskmod(dataBits, M);

% Add AWGN noise
noisyData = awgn(modData, EbNo, 'measured');
%noisyData = awgn(modData, EbNoVec(i), 'measured');

% Demodulate the received signal
demodData = pskdemod(noisyData, M);

% Calculate bit errors
[numErrors(i), berSim(i)] = biterr(dataBits, demodData, k);
end

% Theoretical BER for 4-PSK
berTheo = berawgn(EbNoVec, 'psk', M, 'nondiff');

```

Appendix B

Matrices Used in the encryption system

B.1 Message Matrix 1

https://github.com/anupamsworld/anupamsworld.github.io/blob/main/research_papers/mtech/message.txt

B.2 Message Matrix 2

https://github.com/anupamsworld/anupamsworld.github.io/blob/main/research_papers/mtech/message_2.txt

B.3 Message Matrix 3

https://github.com/anupamsworld/anupamsworld.github.io/blob/main/research_papers/mtech/message_3.txt

B.4 S Matrix 1

https://github.com/anupamsworld/anupamsworld.github.io/blob/main/research_papers/mtech/S_matrix.txt

B.5 S Matrix 2

https://github.com/anupamsworld/anupamsworld.github.io/blob/main/research_papers/mtech/S_matrix_2.txt

B.6 S Matrix 3

https://github.com/anupamsworld/anupamsworld.github.io/blob/main/research_papers/mtech/message_3.txt

B.7 P Matrix

https://github.com/anupamsworld/anupamsworld.github.io/blob/main/research_papers/mtech/P_matrix.txt