Republic of Iraq
Ministry of Higher Education and Scientific Research
Al-Nahrain University
College of Science

A Mechanism to Enhance McEliece Cryptosystem Performance

A Thesis

Submitted to College of Science, Al-Nahrain University in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

BY

Noor Redha ِِAbd Al-Razak Al-Kazaz

(B.Sc. 2003)

Supervisors

Dr. Sattar B. Sadkhan Dr. Jamal M. Kadhum

2006 **1427**

بسم الله الرحمن الرحيم

قَالُواْ سُبْحَانَكَ لاَ عِلْمَ لَنَا إِلاَّ مَا عَلَّمْتَنَا إِنَّكَ أَنتَ الْعَلِيمُ الْحَكِيمُ ۞

صدق الله العظيم

البقرة-٣٢

Supervisor Certification

We certify that this thesis was prepared under our supervision at the Department of Computer Science / College of Science / Al-Nahrain University, by **Noor Redha Abd Al-Razak Al-Kazaz** as partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Signature:	Signature:
Name: **Dr. Sattar B. Sadkhan**	Name: **Dr. Jamal M. Kadhum**
Title : **Assist. Prof.**	Title : **Lecturer**
Date : **/ /2006**	Date : **/ /2006**

In view of the available recommendations, I forward this thesis for debate by the examination committee.

Signature:

Name : **Dr. Taha S. Bashaga**

Title : **Head of the department of Computer Science, Al-Nahrain University.**

Date : / / **2006**

Certification of the Examination Committee

We chairman and members of the examination committee certify that we have studies this thesis "**A Mechanism to Enhance McEliece Cryptosystem Performance**" presented by the student **Noor Redha Abd Al-Razak Al-Kazaz** and examined her in its contents and that we have found it worthy to be accepted for the degree of Master of Science in Computer Science.

Signature:
Name: **Dr. Loay E. George**
Title : **Assist. Prof.**
Date : **/ /2007**

(Chairman)

Signature:	Signature:
Name: **Dr. Jane J. Stephan**	Name: **Dr. Sawsan K. Thamer**
Title : **Assist. Prof.**	Title : **Lecturer**
Date : **/ /2007**	Date : **/ /2007**
(Member)	**(Member)**
Signature:	Signature:
Name: **Dr. Sattar B. Sadkhan**	Name: **Dr. Jamal M. Kadhum**
Title : **Assist. Prof.**	Title : **Lecturer**
Date : **/ /2007**	Date : **/ /2007**
(Supervisor)	**(Supervisor)**

Approved by the Dean of the Collage of Science, Al-Nahrain University.

Signature:
Name: **Dr. LAITH ABDUL AZIZ AL-ANI**
Title : **Assist. Prof**.
Date : **/ /2007**

(Dean of Collage of Science)

Dedication

To my beloved family

Acknowledgment

I am indebted to the head of department of Computer Science of AL-Nahrain University *Dr. Taha S. Bashaga* for all things he did that enabled me to continue this work.

I would like to express my sincere gratitude and appreciation to my supervisors *Dr. Sattar B. Sadkhan* and *Dr. Jamal M. Kadhum* for their guidance, assistance and encouragement through the course of this project.

Special thanks to all my friends, group of M.Sc. laboratory. Finally, I want to express my gratitude to my beloved family for encouragement and understanding during the period of my study.

Noor Redha Abd Al-Razak

Abstract

Encoding and decoding of transmitted data through the communication system is considered as an important subject that paid great attention. It is well known that there are many types of error correction codes; some of these types are hamming and extended hamming codes. Hamming code can detect and correct single error while extended hamming code can detect two and correct one only.

This work is concerned with the implementation of McEliece cryptosystem by using hamming code and extended hamming code, evaluate the security of this cryptosystem and enhance the performance of it. McEliece cryptosystem is a public key cryptosystem based on algebraic coding theory.

The system design and implementation consists of four phases. The first phase is the key generation phase of McEliece cryptosystem using hamming code and extended hamming code. There are three secret keys participate in creating public key (generator matrix, non-singular matrix, and permutation matrix). The second phase is the encryption phase, in this phase the original message is converted to encrypted message. The original message is encrypted by using the public key generated in the first phase, and then adding the error to it. The third phase is the decryption phase. The secret keys are used in this phase to decrypt the encrypted message and to obtain the original one. Also, in decryption process the message is decoded from the errors added to it. The last phase is concerned with evaluating the security of this cryptosystem (McEliece cryptosystem). The evaluation was based on using brute force attack, and the security of this cryptosystem can be measured by its resistance to this type of attack. After applying this attack some weak points have been noticed. To overcome these weak points and to make this cryptosystem more secure, a modification was proposed an implemented. In this phase other cryptosystem parameters have been evaluated, they are: public key size, message expansion and information rate. The security, implementation and the use of this cryptosystem are issues significantly affected by these parameters.

All required programs, in this research project have been implemented by using Visual Basic (version 6) programming language working in Window XP operating system platform.

Table of Contents

List of Abbreviations

DES	Data Encryption Standard
ECC	Error Correction Codes
RSA	Rivesr Shamir Adleman
SECDED	Single Error Correcting / Double Error Detecting

List of Figures

List of Symbols

Symbol	***Description***
A	Side A
B	Side B
C	Code vector
C_1,C_2	Code vectors
C^T	Transpose code vector
Counter	Number of generation public keys
d	Hamming distance
d_{min}	Minimum hamming distance
E,e	Error vector
G	Public key
G_0	Private key (generator matrix)
$G_{0(k*n)}$	Generator matrix of k*n dimension
H	Parity check matrix
H^T	Transpose parity check matrix
$H^T_{(n*(n-k))}$	Transpose parity check matrix of n*n-k dimension
I	Identity matrix (identity part)
I_k	Identity matrix of order k
I_{n-k}	Identity matrix of order n-k
k	Number of original message bits
m	Number of parity check bits (n-k)
m_i	Message bits
me	Original message
N	Number of generated public keys
n	Number of cipher message bits

n-k	Redundant bits (check bits)
P	Private key (permutation matrix)
$P_{(n*n)}$	Permutation matrix of n*n dimension
P^{-1},Pinv	Inverse permutation matrix
p	Parity matrix (parity part)
p^T	Transpose parity matrix
R	The noise-corrupted vector
S	Private key (non-singular matrix)
$S_{(k*k)}$	Non-singular matrix of k*k dimension
S^{-1},Sinv	Inverse non-singular matrix
Sp	Number of probable secret keys when generating one public key
Spm	Number of all possible probable secret keys
Sy	Syndrome
t	Number of error
w	Hamming weight
y	Encrypted message
y_1	Encrypted message with error
y_2	The result of multiplication y_1 by Pinv

List of Tables

Chapter One

General Overview

1.1 Introduction

Transmitting messages across noisy channels is an important practical problem. Coding theory provides explicit ways of ensuring that messages remain legible even in the presence of errors. Cryptography on the other hand is the science of sending messages in disguise form, it makes sure that messages remain unreadable except to the intended recipient that means only the intended recipients can remove the disguise and read the message. These complementary techniques turn out to have much in common mathematically [Kor05].

It is pretty much taken for granted that data storage and communication are reliable. Increasingly, we expect or hope that our recorded and transmitted data are secure. The theories of coding and cryptography attend to data reliability and security, respectively [Hhl00].

Coding theory is concerned with finding explicit methods, called codes, of increasing the efficiency and fidelity of data communication over a noisy channel. The goal in coding theory has been to provide error-free and secure communication over noisy channels and devise efficient codes and their successful implementation by developing fast encoding and decoding procedures [Cha98]. These efficient codes are error correction codes (ECC). Error correcting codes are an essential part of modern communication and storage systems. ECC add redundancy to the original message in such a way that it is possible for the receiver to detect the error and correct it, recovering the original message. The study of error correction codes and the associated mathematics is known as coding theory [Fre00].

Algebraic coding theory is the theory of error correcting codes; it was originated in 1950 by Richard hamming. Algebraic coding theory is an area of discrete applied mathematics that is concerned with developing error correcting codes and encoding/decoding procedures [Spe00]. This coding theory (Algebraic coding) is differ from cryptography. Cryptography is the mathematical theory behind sending secret messages while algebraic coding theory is the mathematical theory of sending messages that arrive with the same content in which they were sent [Smi02].

The McEliece cryptosystem is a public key cryptosystem that provide secure transmission whose security rests on error correction codes and the difficult problem of decoding a message with random error; it is based on algebraic coding theory, whereas for most other public key systems, it is connected to algorithmic number theory (e.g. RSA, Elliptic Curve Cryptosystem). So in addition to its capability of offering secure transmission, it possesses the capability to correct communication channel error [Sua03].

In practice, the security of any cryptosystem can only be measured by its resistance to actual attempts to break it. Those that have been broken are obviously insecure [Mey98]. An encryption scheme can be broken by trying all possible keys to see which one is used by the communicating parties. This is called a brute force attack. It follows then that the number of keys should be large enough to make this approach computationally infeasible [Men96].

1.2 Literature Survey

- At 1992, Preneel et al, presented a software implementation of the McEliece public-key cryptosystem. The software performs encryption and decryption in a reasonable speed. All coding routines were written in assembly language to speed up the system. Also, this research the attack of recovering the plaintext from a ciphered message was discussed. The main idea is to select and solve k bits (number of original message bits) of n bits (number of cipher message bits) obtained from ciphertext and public key, it is found the probability of success is small [Pre92].
- At 1997, Berson indicated that the McEliece public-key cryptosystem fails to protect any message which is sent to a recipient more than once using different random error vectors; he called this attack a message-resend condition or a message-resend attack. And this cryptosystem fails to protect any messages sent to a recipient which have a known linear relation to one another; it called this a related-message condition or a related-message attack. These attacks are general attack on the class of public-key cryptosystems which use an error-correcting code. And to prevent these attacks; users of the McEliece public-key cryptosystem and of cryptosystems with similar structure, should guard against sending related messages. He presented that one countermeasure which comes to mind is to introduce an element of local randomness into any message before it is encrypted [Ber97].
- At 1998, Johansson offered many cryptosystems relying on the difficulty of the general decoding problem. The general decoding problem is the problem of decoding a received word to the closest codeword in an arbitrary code. This work is based on the observation that in many cryptographic applications it is possible to

create a list of received words in such a way that succeeded in decoding only one of them implies success in an attack of the underlying cryptosystem. He showed that this holds for Stern's identification scheme, McEliece public-key cryptosystem, and the stream cipher application [Joh98].

- At 1998, Lee et al, presented an attack on McEliece public key cryptosystem that is based on algebraic coding theory. This attack is to repeatedly select k bits randomly from an n-bit ciphertext vector, in hope that none of the selected k bits are in error until the cryptanalyst recovers the correct message of k bits, the probability of success of this attack is also small [Lee98].
- At 2000, SUN HUNG-MIN showed that the McEliece scheme suffers from two weaknesses they are; failure to protect any message which is encrypted more than once, and failure to protect any message which have a known linear relationship with one another. The researcher proposed variant McEliece scheme to prevent these attacks. The public key and the secret key in this variant scheme are same as those in the original McEliece scheme, the only difference is using a one-way hash function in encrypting messages, the lack knowledge about this hash function becomes the value of it is unknown. Thus, the message-resend and related message attacks fail [Sun00].
- At 2003, Jeroen introduced an attack against the McEliece public-key cryptosystem called an adaptive chosen ciphertext attack, which is based on the attacker which will continue to alter messages until retrieved enough secret information, and on the assumption that (ordinary) users may see no problem in revealing whether or not an encrypted message deciphers correctly. In the

case of the McEliece system it must repeat the attack for each ciphertext it wishes to decrypt. The aim of this attack is to recover the plaintext of a given ciphertext. And to prevent this attack; users should be alerted when many encrypted messages do not decrypted properly, also must come up with the idea of checking for repeated messages [Jer03].

- At 2003, Suanne et al, showed that there are two types of attacks on Mceliece cryptosystem, structural and decoding. A structural attack consists of attempts to reconstruct a decoder for the code generated by the public key, G. If such an attempt is successful, then the private key G_0 would be revealed and the cryptosystem would be completely broken. A decoding attack consists of decoding the intercepted ciphertext. In the case of a successful decoding attack, the plaintext message is recovered but the cryptosystem remains intact. They showed under different circumstances, many efficient decoding attacks are possible, but structural attacks remain infeasible in general [Sua03].

From the above review one can found that these most researches investigated different types of attacks on McEliece cryptosystem and evaluate the security of it by measuring the power of the system against different types of attack.

As a comparison between these previous efforts and the work done in this thesis, the current research study, implement, evaluate the security of McEliece cryptosystem using a brute force attack and enhance the performance of this cryptosystem. Also evaluates other cryptosystem parameters that affected the security of the system.

1.3 Aim of Thesis

- Study the public key encryption scheme based on algebraic coding theory. Design and implement McEliece public key cryptosystem with hamming and extended hamming code.
- Evaluate the security of McEliece public key cryptosystem type with hamming and extended hamming code by using a brute force attack and trying to overcome the weak points of this cryptosystem and enhance the performance of it.

1.4 Thesis Layout

The contents of individual chapters in the remainder part of this thesis are briefly reviewed:

- ***Chapter two:*** reviews the concept of error correction codes, and McEliece cryptosystem.
- ***Chapter three:*** the practical part of the work is presented in this chapter, and the algorithms that used to implement the system.
- ***Chapter four:*** the results of the system are shown here.
- ***Chapter five:*** introduces the derived conclusions of this work, with recommendations for future work.

Chapter One

General Overview

Chapter Two

An Overview of McEliece Cryptosystem

2.1 Introduction

In this chapter the principles of cryptography, cryptanalysis and the approaches of cryptanalysis attack were presented. The brute force attack is an important cryptanalysis attack, it is presented in details. Also the concepts of error correction codes were described. Finally McEliece public key cryptosystem which relate cryptography and coding theory is overviewed.

2.2 Cryptography

Cryptography is science of using mathematics to encrypt and decrypt data. Data that can be read and understood without any special measures is called plaintext [Sch97].

The process of encoding data to prevent unauthorized parties from viewing or modifying it is called encryption. The encryption process transforms plaintext into ciphertext, while the process of transforming ciphertext back into plaintext is called decryption. These two processes are shown in figure (2.1). A system for encryption and decryption is called a cryptosystem [Pfl89].

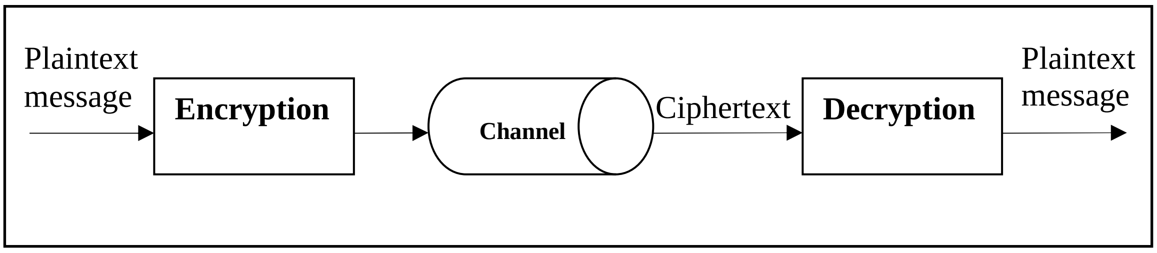

Figure (2.1): Encryption and decryption.

2.3 Types of Encryption Algorithm

Encryption components are an algorithm and key. Encryption algorithm is series of steps that mathematically transforms plaintext or other readable information into unintelligible ciphertext. Ciphertext that has been encrypted is unreadable until it has been decrypted by applying inverse mathematical transformation, which transforms the encrypted ciphertext back into something readable. Both encryption and decryption use a key [Ste99].

Encryption algorithms are classified into two main types, these types are [Sch97], [Men96]:

- Symmetric encryption (secret key encryption).
- Asymmetric encryption (public key encryption).

The characteristic features of symmetric encryption algorithms, is that both the encryption and decryption processes are accomplished by using the same key as known in figure (2.2).

The symmetric key algorithms can be further classified as: stream ciphers and block ciphers. A stream cipher is an encryption scheme which treats the plaintext as a block of unity length. A block cipher is an encryption scheme which breaks up the plaintext messages to be transmitted into strings called blocks (of a fixed length), and encrypts one block at a time. A good example of symmetric cryptosystem algorithm is Data Encryption Standard (DES).

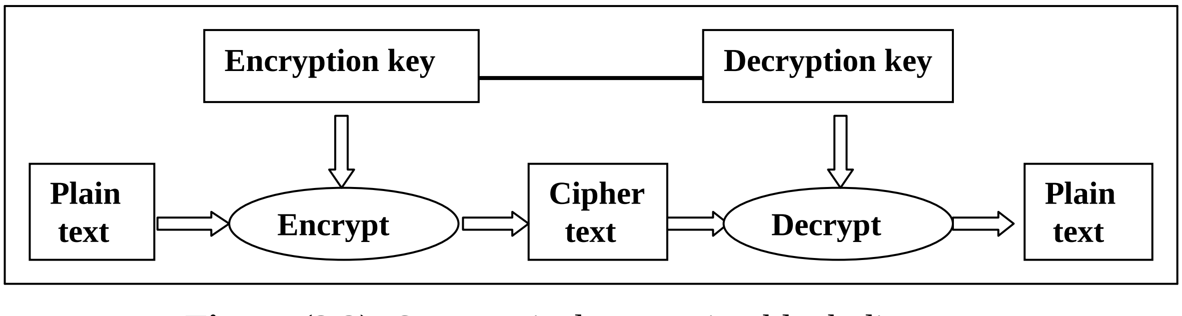

Figure (2.2): Symmetrical encryption block diagram.

Asymmetric encryption is more complex and more secure. The characteristic feature of public key encryption is that the encryption and decryption processes are accomplished by using different keys. More precisely, the encryption process is based on using a key that is easily available, while the decryption process is based on another key, which is only accessible to a specific entity. The key that is used for the encryption process is known as the public key, while the key that is used for the decryption process is known as the private key (secret key) as shown in figure (2.3). RSA and McEliece are two examples for such a system.

Symmetric encryption is fast but not as safe as asymmetric encryption because someone could intercept the key and decode the messages. But because of its speed, it’s commonly used for encoding-commerce transactions.

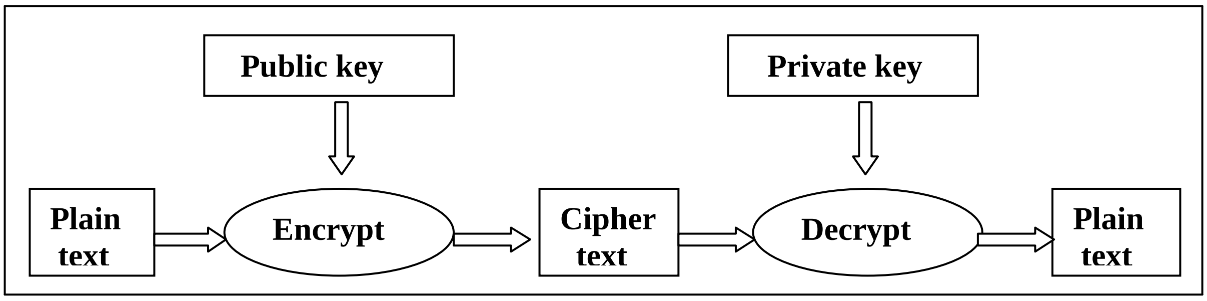

Figure (2.3): Asymmetrical encryption block diagram.

Cryptography algorithms are the heart of secure systems worldwide, providing encryption for millions of sensitive financial, government, and private transaction daily [Jam02].

2.4 Cryptanalysis

Cryptanalysis is the study of mathematical techniques for attempting to defeat cryptographic techniques [Men96]. In practice, the security of any cryptosystem can only be measured by its resistance to actual attempts to break it. Those that have been broken are obviously insecure. The central issue in assessing the resistance of an encryption algorithm to cryptanalysis is the amount of time that a given type of attack will take [Mey98]. Realistic system which means that it is theoretically breakable. The problem is to determine how long it would take to break the proposed cipher system and to estimate the involved cost (in terms of time and computation load). If the time is extremely huge then, for all practical purpose, we may regard the considered system as secure [You02]. So one can estimate the required time to compute the secret key, which is the minimum amount of work to compute the key, then for sufficiently large time the encryption system is, for all practical purpose, a secure system [Men96].

There are two approaches to attacking a conventional encryption scheme [Sta03]:

- **Cryptanalysis:** cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used. If the attack succeeds in deducing the key, the effect is catastrophic: all future and past messages encrypted with that key compromised [Men96]. There are various types of cryptanalytic attacks, based on the amount of information known to the cryptanalytic [Buc01]:

    a. Ciphertext only attack: the attacker knows ciphertexts and tries to recover the corresponding plaintexts or the key.

    b. Known plaintext attack: the attacker knows a plaintext and the corresponding ciphertext or several pairs. He tries to find the key used or to decrypt other ciphertexts.

c. Chosen plaintext attack: the attacker is able to encrypt plaintexts but does not know the key. He tries to find the key used or to decrypt other ciphertexts.

d. Adaptive chosen plaintext attack: the attacker is able to encrypt plaintexts. He is able to choose new plaintexts as a function of the ciphertexts obtained but does not know the key. He tries to find the key used or to decrypt other ciphertexts.

e. Chosen-ciphertext attack: the attacker can decrypt but does not know the key. He tries to find the key.

- **Brute Force Attack:** A brute force attack (also called exhaustive attack) is a method of defeating a cryptographic scheme by generating a large number of possibilities and trying it in order to recover the plaintext used to produce a particular ciphertext [Cla97]. The most difficult problem is presented when all that is available is the ciphertext only. In some cases, not even the encryption algorithm is known, but in general we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute force approach of trying all possible keys [Sta03]. The quicker the brute force attack, the weaker the cipher. Feasibility of brute force attack depends on the key length of the cipher (i.e. depends on the number of keys) and on the amount of computational power available to the attacker, thus the number of keys should be large enough to make this approach computationally infeasible [How00]. This type of attacks can take several hours, days, months, and even years to run. Brute forcing is one of the first methods approached to getting passwords. Even a lot of governments agencies and such whose main goal are to decrypt various coded passwords use brute force methods for one of these approaches. For example, a brute-force attack may have a dictionary of all words and/or a listing of commonly used passwords. To gain access to the account using a brute-force attack, the program would try all the available words it has to gain access to the account. So, brute forcing is a great method in the security world and must be clearly understood [Jar98].

2.5 Error Correcting Codes [Sha85]

Error detecting / correcting code is the calculated use of redundancy. The functional blocks that accomplish error correcting codes are the channel encoder and channel decoder. The channel encoder systematically adds digits to the transmitted message digits. These additional digits, while conveying no new information themselves, make it possible for the channel decoder to detect and correct errors in the information bearing digits. The purpose of error detecting / correcting codes is to reduce the chance of receiving messages which differ from the original message. A system that employs error correcting codes is shown in figure (2.4).

The channel encoder and decoder are functional blocks in the system that, by acting together, reduces the overall probability of error. The encoder divides the input message bits into blocks of k massage bits and replaces each k bit massage block with an n bit codeword by adding (n-k) check bits to each message block. The decoder looks at the received version of the codeword, which may occasionally contain error, and attempts to decode the k message bits. While the check bits convey no new information to the receiver, they enable the decoder to detect and correct transmission error and thereby lower the probability of error.

The design of the encoder and decoder consist of selecting rules for generating codeword from message blocks and for extracting message blocks from the received version of the codewords.

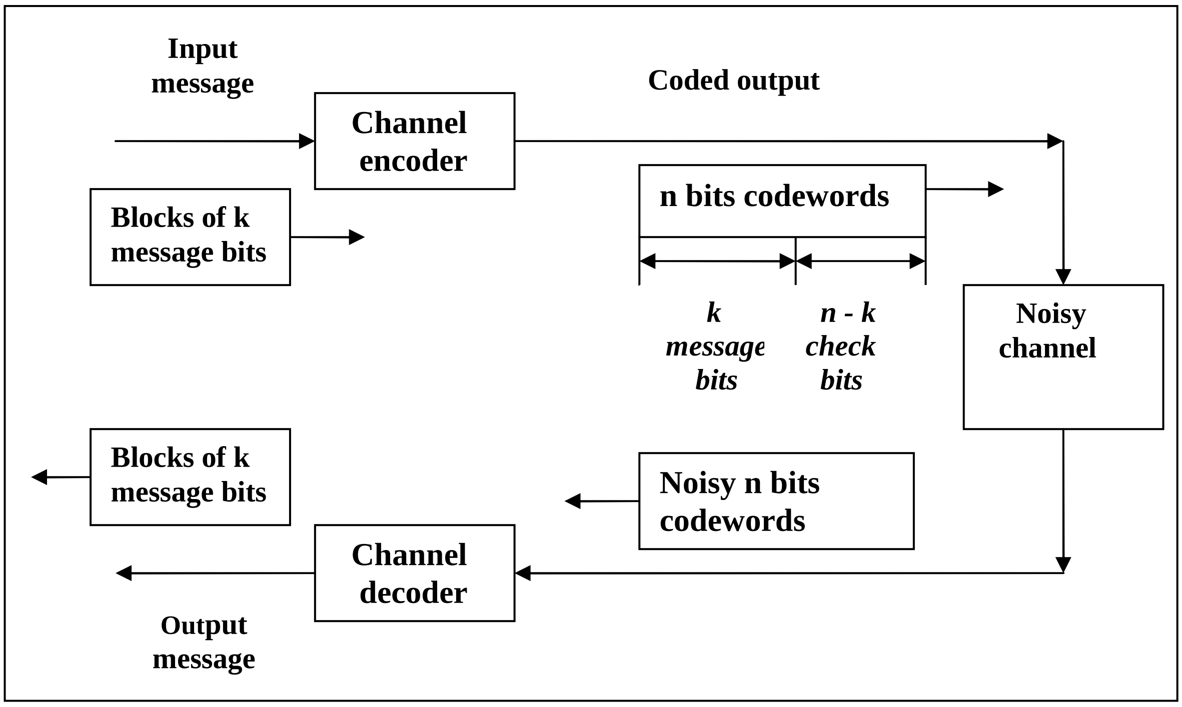

Figure (2.4): Block diagram of data communication system employing an error correcting code.

2.6 Types of Code

Error detecting/correcting codes are often divided into two broad categories: block codes and convolutional codes. In block codes, a block of k information bits is followed by a group of n-k check bits. At the receiver, the check bits are used to verify the information bits in the information block preceding the check bits. In convolutional codes, check bits are continuously interleaved with information bits; the check bits verify the information bits not only in the block immediately preceding them, but in other block as well. So a convolutional code, unlike a block code, the channel encoder accepts message bits as a continuous sequence and thereby generates a continuous sequence of encoded bits [Das86].

2.7 Linear Block Codes

A code is said to be linear if any two codewords in the code can be added in modulo-2 arithmetic to produce a third codeword in the code. For the purpose of encoding message for error protection, the long message is broken into message blocks consisting of, say, k of bits of information. Then redundant bits (normally known as parity bits) are added to these k bits according to certain rules of coding, and the codewords of length n bits, inclusive of (n-k) parity bits that shown in figure (2.5), are transmitted through the noisy channel. In the receiver, the k message bits are decoded from the erroneous received message blocks using suitable algorithms. With k bits of information per blocks, there are 2^k possible distinct messages (or codewords), out of the 2^n codewords that may be generated with n bits. This set of 2^k codewords called a block code [Hay01].

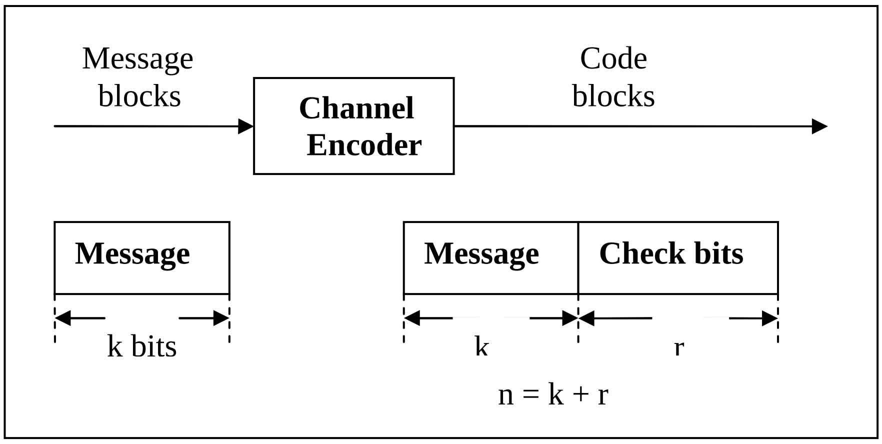

Figure (2.5): Structure of systematic codeword.

2.7.1 Error Detection and Correction Capabilities of Linear Block Codes [Mic85]

Some of the basic terminology that will be used in defining the error detection/correction capabilities of a linear block code. First, the Hamming weight of a code vector C is defined as the number of nonzero components of C. The Hamming distance between two vectors (C_1 and

C_2) having the same number of elements is defined as the number of positions in which they differ. Finally, the minimum distance of linear block code is the smallest distance between any pair of different codewords in the code.

eg. w(0010110) = 3

eg. d(0001111; 1000110) = 3

eg. For codewords = {1000011; 0100101; 0010110;0001111}

$d_{min} = 3$

The minimum distance of linear block code is equal to the minimum weight of any nonzero word in the code. The ability of linear block code to correct errors can be specified in terms of the minimum distance of the code.

A linear block code with a minimum distance d_{min} can correct $[(d_{min}-1)/2]$ errors and detect $[d_{min}-1]$ errors in each codeword, so

Error detection capabilities <= (d_{min}– 1).

Error correction capabilities <= (d_{min} – 1) / 2.

2.7.2 Matrix Description of Linear Block Codes

The encoding operation in a linear block scheme consists of two basic steps:

(1) The information sequence is segmented into message blocks, each block consisting of k successive information bits.

(2) The encoder transforms each message block into a larger block of n bits. We can describe the encoding operation using matrices.

We will denote the message block as a row vector or k-tuple (m= $(m_1, m_2, \ldots, m_k)$) where each message bit can be a 0 or 1. Thus we have 2^k distinct message blocks. Each message block is transformed to a codeword C of length n bits ($C=(c_1,c_2,\ldots,c_n)$) by the encoder and there are

2^k distinct codewords, one unique codeword for each distinct message blocks. This set of 2^k codewords, is called a (n,k) block code [Sha85].

In a linear block code, the first k bits of the codeword are the message bits, that is,

$$m_i\,, \quad i = 1, 2, \ldots, k$$

The last n-k bits in the codeword are check bits generated from the generator matrix is:

$$G = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & p_{11} & p_{12} & \ldots & p_{1,n-k} \\ 0 & 1 & 0 & \ldots & 0 & p_{21} & p_{22} & \ldots & p_{2,n-k} \\ 0 & 0 & 1 & \ldots & 0 & p_{31} & p_{32} & \ldots & p_{3,n-k} \\ : & : & & & : & & & & : \\ 0 & 0 & 0 & \ldots & 1 & p_{k1} & p_{k2} & \ldots & p_{k,n-k} \end{bmatrix} \quad \ldots\ldots\ldots\ldots(2.1)$$

and

$$C = mG \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2.2)$$

The (k*n) matrix G has the form

$$G = [I_k \mid P]_{k*n} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2.3)$$

The matrix I_k is the identity matrix of order k and P is an arbitrary k by (n-k) matrix. When P is specified, it defines the (n,k) block code completely [Sha85].

An important step in the design of a (n,k) block code is the selection of a P matrix that depends on error detecting and correcting capabilities, so that the code generated by G has certain desirable properties such as ease of implementation, ability to correct errors, high rate efficiency, and so fourth. As a result, for any (n,k) linear code there exists a (k*n) matrix G, whose rows are these k linearly independent codewords [Das86].

Example (2.1): the generator matrix for (4,7) block code is given below

$$G = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right)$$

The message block size k for this code is 4, and the length of the code vectors n is 7. The code vector of the message block m = (1110) is given by:-

$$C = mG = (1110) \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right)$$

$$= 1110100$$

This example illustrates how an encoder for the (4,7) code generates code vectors. The encoder essentially has to store the G matrix (or at least the sub matrix P of G). The complexity of the encoder increases as the block size n and the number of check bits (n-k) increase [Sha85].

Associated with each (n,k) block code is a parity check matrix H, which is defined as

$$H = \left(\begin{array}{cccc|cccccc} p_{11} & p_{21} & .... & p_{k1} & 1 & 0 & 0 & 0 & ... & 0 \\ p_{12} & p_{22} & .... & p_{k2} & 0 & 1 & 0 & 0 & ... & 0 \\ : & : & & & : & & & & & : \\ p_{1,n-k} & p_{2,n-k} & .... & p_{k,n-k} & 0 & 0 & 0 & 0 & ... & 1 \end{array}\right) \quad \text{.........(2.4)}$$

$$H = [P^T \mid I_{n-k}]_{(n-k)xn} \quad \text{...............................(2.5)}$$

The parity check mat rix can be used to verify whether a codeword C is generated by the matrix $G = [I_k \mid P]$. This verification can be done as follows; C is a codeword in the (n,k) block code generated by $G=[I_k\,|\,P]$ if and only if

$$CH^T=0 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2.6)$$

$$mGH^T=0$$

$$m\,0=0$$

Where H^T is the transpose of H matrix,

$$[G]=[I_k\,|\,P];\ \text{and}\ [H]=[P^T\,|\,I_{n\text{-}k}]$$

$$[G].[H^T]=[I_k\,|\,P].\begin{bmatrix} P \\ \text{------} \\ I_{n\text{-}k}\end{bmatrix}=[P\oplus P]=0 \quad \ldots\ldots\ldots\ldots\ldots(2.7)$$

The inner product of a vector in the row space of $[G]$ and a row in $[H]$ is zero. Also $[H].[C^T]=0$ [Das86].

While the generator matrix is used in the encoding operation, the parity check matrix is used in the decoding operation as follows:

Consider a linear (n,k) block code with a generator matrix $G=[I_k\,|\,P]$ and parity check matrix $H=[P^T\,|\,I_{n\text{-}k}]$. Let C be a code vector that was transmitted over noisy channel and let R be the noise-corrupted vector that was received. The vector R is the sum of the original code vector C and an error vector encoding, that is,

$$R=C+E \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2.8)$$

The receiver does not know C and E; its function is to decode C from R, and the message block m from C. The receiver does the decryption operation by determining an (n-k) vector Sy defined as

$$Sy=RH^T \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2.9)$$

The vector Sy is called the error syndrome of R

$$Sy=[C+E]H^T$$

$$=CH^T+EH^T$$

$$=0+EH^T$$

and obtain [Das86]:

$$Sy=EH^T \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2.10)$$

Since $CH^T = 0$. Thus the syndrome of a received vector is zero if R is valid code vector. If errors occur in transmission, then the syndrome Sy of the received vector is nonzero. Furthermore, Sy is related to the error vector. The row of the parity check matrix H^T serve as location vectors for bit positions where error occurred in a received word. The receiver can verify that, a signal error in the *i*-th bit of C would lead to a syndrome vector that would be identical to the *i*-th row of the matrix H^T. Thus signal errors can be corrected at the receiver by comparing Sy with the rows of H^T and correcting the *i*-th received bit if Sy matches with the *i*-th row of H^T [Das86].

2.8 Types of Linear Block Codes

There are many types of linear block codes, the most important types are Hamming code, extended hamming code, Golay code, Reed Muller codes and many other types [Pet72].

2.8.1 Hamming Code

Hamming code is a linear block codes capable of correcting single error. This code must have a minimum distance $d_{min} = 3$. We know that when a single error occurs, say in the *i*-th bit of the codeword, the syndrome of the received vector is equal to the *i*-th row of H^T. Hence, if we choose the n rows of the ((n)*(n-k)) matrix H^T to the distinct, then the syndrome of all single error will be distinct and we can correct single error. There are two points we must remember in chosen rows of H^T. First, we must not use a row of 0's since a syndrome of 0's corresponds to no error. Second, the last n-k rows of H^T must be chosen so that we have an identity matrix in H^T [Das86].

Hamming codes parameters are [Leh93]:

- Code length $n = 2^m - 1$, $m = (n-k)$
- Number of parity check bits $m = n-k$

- Number of information bits $k = (2^m - m - 1)$
- Minimum hamming distance of the generator matrix = 3
- Error correcting capability = 1

Each row in H^T has (n-k) entries, each of which could be a 0 or 1. Hence we can have 2^{n-k} distinct rows of (n-k) entries out of which we can select $2^{n-k}-1$ distinct rows of H^T (the row of 0's is the only one we cannot use). Since the matrix H^T has n rows.

Hamming codes are widely used in computing, telecommunication and other application including data compression, popular puzzle and turbo codes [Leh93].

Example (2.2): [Das86]

$$[G]= \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right)$$

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

The syndrome Sy is generated from:

$$[Sy]=[R].[H^T]=[R].\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

So for a message $m_1m_2m_3m_4$ = 0 0 1 0, the code C = 0 0 1 0 1 1 0. If the received vector R = 0 0 1 0 0 1 0, then the syndrome Sy = 1 0 0, and inspection of H^T, the error is in the 5th position, giving E = 0 0 0 0 1 0 0. So that can obtain C = 0 0 1 0 1 1 from R after correct error.

2.8.2 Extended Hamming Code

If multiple errors occur in the transmission of a word the hamming code cannot even detect them because the errors will produce a syndrome Sy which will either be zero or equal to some column of the parity check matrix. Thus for a hamming code decoding failure never occurs, it provides a complete decoding algorithm that codes can detect and correct one error and detect any two errors. This two bit error can be detected since they will correspond to a non-null syndrome which is not a row of H^T. The points that we must remember in chosen rows of H^T, we must not use a row of zeros since a syndrome of zeros corresponds to no error, and must be distinct. That is another code called the Extended hamming codes [Jon06]. The Extended hamming codes have three parameters [Leh93]:

- Length of codewords: $n = 2^m$
- Information symbols: $k = 2^m - m - 1$
- Minimum distance: $d = 4$

The extended hamming codes obtained by adding an additional redundant bit, this redundant bit called a parity check bit; the overall parity check bit is added to the end of each codeword [Leh93].

The minimum distance of this code is 4 since it is a linear block code whose non-zero codewords have minimum weight 4. Note that the extended hamming codes are still linear codes, since adding a one or zero at the end of each codeword is a linear process [Ric96].

Example (2.3): [Han06]

The generator matrix and parity check matrix of extended hamming code formed as:

Consider the following generator and parity check matrices of length 7 hamming code C.

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \qquad H^T = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The generator and parity check matrices of extended code (8,4) are:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \qquad H^T = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

So for a message equal to (1 0 1 0), the code C equal to (10100101), after insert the error the received word is:

$$R = 10100101 + 0010000$$
$$R = 10000101$$

The syndrome Sy is equal to (1011), and inspection of H^T, the error in the third position, so that can obtain C = 10100101 from R after correct the error.

At the same message and code, after insert two errors the received word is:

$$R = 10100101 + 01000001$$
$$R = 11100100$$

The syndrome Sy is equal to (1100), by inspection of H^T, find that no row in H^T equal to Sy, means that two errors added to the original code.

2.9 Error Correction Codes Applications [Mos01]

Error detecting / correcting codes are implemented in almost any application that includes the keywords “transmission” and “information”, taken in their broadest sense. For instance, transmission may refer to the storage of data on a computer hard disk and the retrieval of that data.
The use of error correcting codes is ever expanding, and it includes:

- Radio
- Long-distance telephony
- Television (High-Definition Television)
- Data storage systems
- Compact disk and Digital Versatile Disk
- International data networks
- Wireless communications
- Deep-space communications (satellites, telescopes, space probes)

Error detecting / correcting codes are widely used for improving the reliability of computer storage systems. The requirement for such systems, which at first used core memories, was for a single error correcting / double error detecting (SECDED) code. The first error control scheme to be implemented on computer memories were the hamming codes which have this error control capacity. Especially after core memories were replaced by semi-conductor memories, which are faster but their high density per chip induces more errors, error control codes became an essential design feature in computer storage systems.

2.10 Cryptosystems Based on Error Correcting Codes [Lou00]

Cryptosystems based on error correcting codes require a class of codes with some properties that are a good and efficient (fast) decoding algorithm and the type of codes must be large enough to avoid any enumeration. McEliece public key is the cryptosystem based on error correcting codes.

2.11 McEliece Public Key Cryptosystem

The McEliece public key cryptosystem was proposed nearly 20 years ago, by McEliece in 1978. The system is simple to explain and is very fast in execution. McEliece cryptosystem is a public key cryptosystem based on algebraic coding theory that provide secure transmission whose security rests on error correcting codes and the difficult problem of decoding plaintext from ciphertexts which the sender intentionally garbles with random error [Ber97].

There are many classes of linear codes which have very fast decoding algorithms. The basic idea of the McEliece system is to take one of these linear codes and disguise it, when trying to decrypt a message, is forced to use syndrome decoding. McEliece suggested using hamming codes and extended hamming codes, which are linear codes with a fast decoding algorithm, in the system, but any linear code with a good decoding algorithm can be used [Che00].

The McEliece public key cryptosystem is widely used especially after changes in technology and economics, for example the plummeting cost of storage, keep it on the list of candidates for some applications [Ber97].

The McEliece scheme uses a generator matrix and a parity-check matrix. Generally, the secret key to this kind of public key cryptosystems is the code itself, for which an efficient decoding algorithm is known [Lou00].

In this system, the public key, G, is a (k*n) matrix that is a product of three private keys: S, G_0 and P, where S is a (k*k) scrambling (non singular or invertible) matrix, G_0 is (k*n) generator matrix, and P is a (n*n) permutation matrix. Both S and P provide the required randomization effect on G.

$$G = S\,G_0\,P \quad \text{……………………………….....(2.11)}$$

S, G_0 and P are kept secret. For sender send a message to receiver, the sender blocks it into binary vectors of length k. If (m) is one such block, the sender randomly constructs a binary vector of weight t (that is, randomly places t 1's in a zero vector of length n), call it e that represent error vector (i.e. hamming code have ability to add one error and then can detect, correct it, extended hamming code also having ability to add one error and can detect, correct it and have ability to add two errors and then can only detect (not correct) their) and then sends to receiver the vector

$$y = mG + e \quad \text{……………………………….….....(2.12)}$$

The receiver receives y to decrypt the message. The receiver computes

$$y = mG + e$$

$$yP^{-1} = (mG + e)\,P^{-1} \quad \text{……………………….......(2.13)}$$

$$y` = (mSG_0P)P^{-1} + eP^{-1}$$

$$y` = mSG_0 + eP^{-1}$$

$$y` = mSG_0 + e`$$

Where e' is a vector of weight t (since P^{-1} is also a permutation matrix). The receiver now applies the fast decoding algorithm to strip off the error vector e' and get the codeword $(mS)G_0$. The vector mS can now be obtained from the first k positions of $(mS)G_0$, because G_0 matrix has been written in standard form $[I_k : P]$. Then receiver recovers m by multiplying mS by S^{-1} matrix [Che00].

There are major concerns with the McEliece public key cryptosystem, these are [Che00]; public key size, message expansion (message expansion is the ratio between the length of the encrypted message and the length of the secret message [Car05]), and information rate (this is a measure of how much information on the average is being carried by a symbol [Heu98]).

Brute force attack (2.4) was used to evaluate the security of McEliece public key cryptosystem.

Chapter Two

An Overview of McEliece Cryptosystem

Chapter Three
The Proposed System Design and Implementation

3.1 Introduction

The problem treated in this project implements the McEliece cryptosystem when using hamming and extended hamming code, evaluates the security of this cryptosystem and enhances the performance of it.

The proposed system of implementing McEliece public key cryptosystem, evaluating its security and enhancing it, consists of four phases. The first phase is concerned with generating the McEliece public key cryptosystem using two types of code that are hamming code and extended hamming code. The second phase is concerned with encrypting the message, and in this phase the error was added to it. The third phase is concerned with decrypting the encrypted message and obtained the original one. Also, in this phase the message was decoded from the error added to it. The last phase is concerned with evaluating the security of this system (McEliece cryptosystem). The evaluation phase based on attacking this cryptosystem using brute force attack and measures the power of it against this type of attack. Design and implement a mechanism that is used to overcome the weak points of this cryptosystem and to enhance the performance of it. Also, this phase is computing other cryptosystem parameters; key size, message expansion and information rate.

The general algorithm adopted to implement, evaluate and enhance this system was as follows:

1- Generate McEliece public key using hamming code and extended hamming code.

2- Get the message to be encrypted as a vector of binary data. Then, the encrypted message will be transmitted through a noisy channel, so noise will be added to the message.
3- The receiver will receive the encrypted message and starts to decrypt it. Decryption process is an important stage and the encrypted message requires several operations to be decrypted. The above steps are presented in figure (3.1).
4- The security of McEliece cryptosystem was evaluated by using a kind of attack called brute force attack. According to this attack, some weak points have appeared, and an adopted mechanism was used to overcome this weakness. The generated system according to the previous mechanism was more secure with most different value of cryptosystem parameter (m).
5- Computing and evaluating other cryptosystem parameters; the size of the public key, message expansion and information rate. The security of this cryptosystem was affected by these parameters.

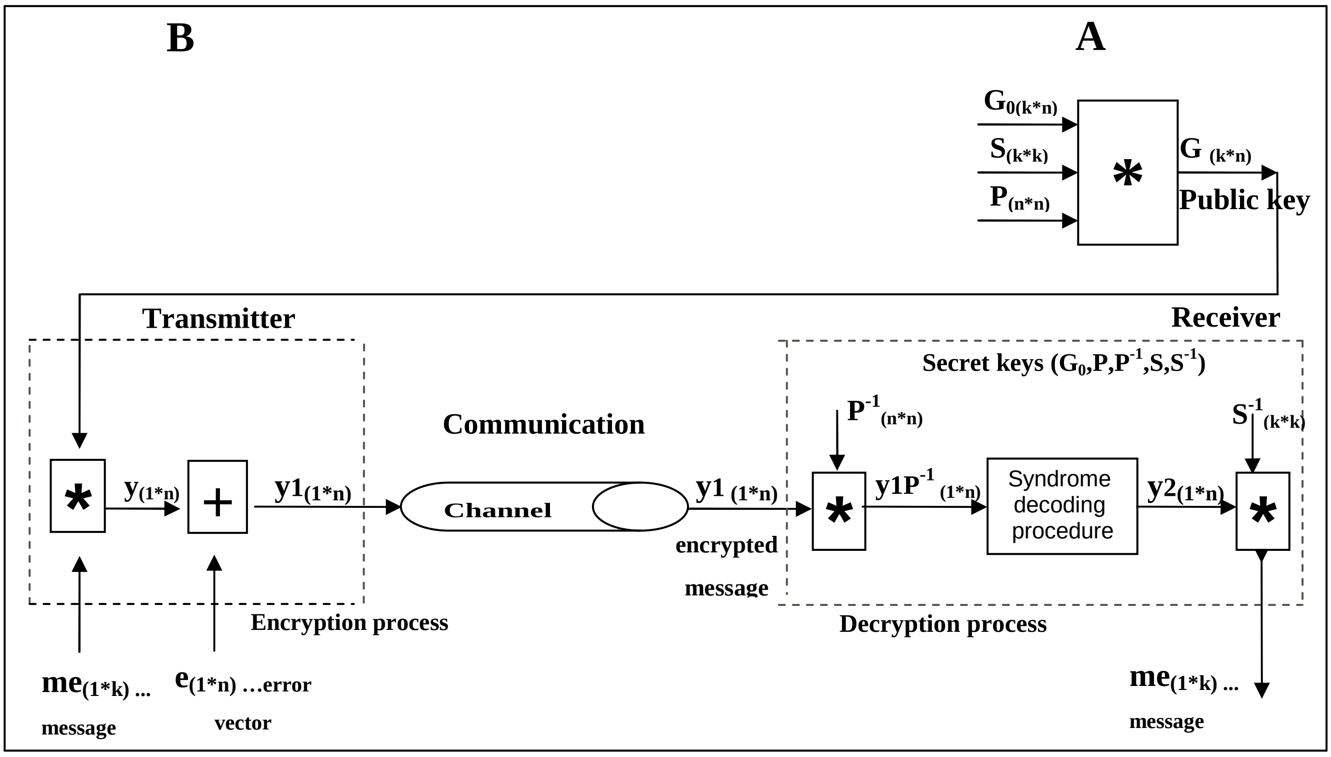

Figure (3.1): McEliece system model.

3.2 Key Generation Phase

a) Generate a public key of hamming code

The generation of the key here depends on the hamming code. Hamming code can correct single error that added to the code, this code has many parameters:

- Value of m: m = (n-k), m is limited between (2 to 10), (n > k).
- Value of n: n = 2^m-1, let n be a y-axes of the public key matrix.
- Value of k: k = 2^m - m - 1, let k be an x-axes of the public key matrix.

b) Generate a public key of extended hamming code

The generation of the key here depends on the extended hamming code. Extended hamming code corrects single error and detects two errors that added to the code. The algorithm of generating a public key of extended hamming code is the same of generating a public key of hamming code but the difference in creating the generator matrix of extended hamming code. Extended hamming code also has many parameters:

- Value of m: m = (n-k)-1, m is limited between (2 to 10), (n > k).
- Value of n: n = 2^m, let n be a y-axes of the public key matrix.
- Value of k: k = 2^m - m - 1, let k be an x-axes of the public key matrix.

The algorithm of generating a public key of hamming code and extended hamming code is:

i. Generate a binary generator matrix of hamming code and extended hamming code.

   The generator matrix of hamming code is a binary matrix of k*n dimension with two parts (identity part and parity part). The minimum distance between any two codewords equal to 3, and the minimum distance of linear block code is equal to the minimum weight of any nonzero word in the code. This matrix must be linear independent. The generator matrix was considered as a secret key. This random generation is performed as follows:

Algorithm (3.1): Generate a random generator matrix of hamming code.

Input:

n: an integer number represents dimension value (a y-axes)of the generator matrix //the value of n determines from algorithm (3.7)

k: an integer number represents dimension value (an x-axes) of the generator matrix //the value of k determines from algorithm (3.7)

Output:

G_0: a generator matrix of k*n dimension

HParity: a parity part of the generator matrix of k*(n-k) dimension

Procedure

```
1. Call Generate identity matrix that input (k) and output (id) // algorithm (3.2)
2. Call Generate parity matrix that input (k,(n-k)) and output (HParity) //
   algorithm (3.3)
3. For i ← 0 to k-1
      For j ← 0 to k-1
         Set G0(i, j) ← id(i, j)
      End loop (j)
   End loop (i)
4. For i ← 0 to k-1
   4.1 Set x ← 0
   4.2 For j ← k to n-1
          Set G0(i, j) ← HParity(i, x)
          Increment x by 1
       End loop (j)
   End loop (i)
```

End.

Algorithm (3.2): Generate identity matrix.

Input:

z: an integer number, represents dimensions values of the identity matrix

Output:

id: identity matrix of z*z dimension

Procedure

```
1. For i ← 0 to z-1
       For j ← 0 to z-1
           If (i = j) then Set id(i, j) ← 1 Else Set id(i, j) ← 0
       End loop (j)
   End loop (i)
End.
```

Algorithm (3.3): Generate a random parity matrix.

Input:

k: an integer number represents dimension value (an x-axes) of Parity matrix

z1: an integer number represents dimension value (y-axes) of Parity matrix

Output:

Parity: a parity matrix of k*z1 dimension

Procedure

```
1. Set x ← 0
2. Do
     2.1 Set count ← 0
     2.2 For j ← 0 to z1-1
             Set Parity(x, j) ← randomize value 0 or 1
             If Parity(x, j) = 1 Then increment count by 1
         End loop (j)
   Until (count≥2)
3. Do While (x < k-1)
     3.1 Do
             Set count ← 0
             For j ← 0 to z1-1
                 Set temp (j) ← randomize value 0 or 1
                 If temp (j) = 1 Then increment count by 1
```

```
                End loop (j)
            Until (count $\geq$ 2)
        3.2 For i $\leftarrow$ 0 to x
                Set co1 $\leftarrow$ 0
                For j $\leftarrow$ 0 to z1-1
                    If (temp(j) $\neq$ Parity(i, j)) then
                        Set co1 $\leftarrow$ 1
                        Exit for (j)
                    End if
                End loop (j)
                If (co1 $\neq$ 1) then exit for (i)
            End loop (i)
        3.3 If (co1 = 1) then
                Increment x by 1
                For j $\leftarrow$ 0 to z1-1
                    Set Parity(x, j) $\leftarrow$ temp(j)
                End loop (j)
            End If
        Loop
End.
```

Example (3.1.a):

Let m = 3

n = 7, k = 4

The generator matrix of hamming code is:

$$G_{0\,\mathbf{(4*7)}} = \begin{bmatrix} \underbrace{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}}_{I_{\mathbf{(4*4)}}} & \vdots & \underbrace{\begin{matrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}}_{P_{\mathbf{(4*3)}}} \end{bmatrix}$$

Matrix I is an identity matrix of k*k dimension and matrix P is a parity matrix of k*(n-k) dimension. The two conditions are the minimum distance between any two codewords in the generator matrix must equal to 3, and the minimum distance of linear block code is equal to the minimum weight of any nonzero word in the code. The generator matrix of hamming code depends on these two conditions (i.e., just only one must exist in each row of the first part (identity part) of the generator matrix. In the second part (parity part), each row of the matrix must have at least two ones. The minimum weight of each codeword in the generator matrix will equal to 3). And this parity matrix must be linearly independent.

The generator matrix of the extended hamming code is a matrix of k*n dimension with two parts (identity part and parity part). The algorithm of generating the generator matrix of the extended hamming code is the same of generating it of hamming code but the difference that the minimum distance between any two codewords equal to 4 not to 3, and this matrix obtained by adding an additional bit to the end of each codeword. This redundant bit called a parity check bit. The algorithm of creating this matrix is as follows:

```
Algorithm (3.4): Generate a random generator matrix of extended
hamming code.
Input:
    n: an integer number represents dimension value (a y-axes)of the generator
    matrix //the value of n determines from algorithm (3.7)
    k: an integer number represents dimension value (an x-axes) of the generator
    matrix //the value of k determines from algorithm (3.7)
Output:
    $G_0$: a generator matrix of k*n dimension
    ExParity: a parity part of the generator matrix of k*n-k dimension
Procedure
    1. Call Generate identity matrix that input (k) and output (id) // algorithm (3.2)
    2. Call Generate parity matrix that input (k,(n-k-1)) and output (Parity) //
```

```
algorithm (3.3)
3. For i ← 0 to k-1
      For j ← 0 to k-1
         Set $G_0(i, j) \leftarrow id(i, j)$
      End loop (j)
   End loop (i)
4. For i ← 0 to k-1
      4.1 Set x ← 0
      4.2 For j ← k to n-2
             Set $G_0(i, j) \leftarrow Parity(i, x)$
             Increment x by 1
          End loop (j)
   End loop (i)
5. For i ← 0 to k-1
      5.1 Set x ← 0
      5.2 For j ← 0 to n-2
             Set $x \leftarrow (x + G_0(i,j))$ and 1
          End loop (j)
      5.3 Set paritybit(i)←x
   End loop (i)
6. For i ← 0 to k-1
      6.1 Set $G_0(i,n-1) \leftarrow$ paritybit(i)
      6.2 Set x ← 0
      6.3 For j ← k to n-1
             Set $ExParity(i,x) \leftarrow G_0(i,j)$
             Increment x by 1
          End loop (j)
   End loop (i)
End.
```

Example (3.1.b):

Let m = 3

n = 8, k = 4

The generator matrix of the extended hamming code is:

$$G_{0\,(4*8)} = \left[\, \overbrace{\begin{matrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{matrix}}^{I_{(4*4)}} \;\vdots\; \overbrace{\begin{matrix}1&1&0&1\\1&0&1&1\\0&1&1&1\\1&1&1&0\end{matrix}}^{P_{(4*4)}} \,\right]$$

$$\Uparrow$$

parity bits

Matrix I is an identity matrix of k*k dimension and matrix P is a parity matrix of k*(n-k) dimension. A parity bit is a bit at the end of each codeword (i.e., these parity bits represent the summation of each codeword in binary representation). The only difference between generator matrix of extended hamming code and generator matrix of hamming code is the parity check bits at the end of each codeword. These parity check bits achieved that the minimum hamming distance between any two codewords equal to 4 not to 3.

ii. Generate randomly a non-singular binary matrix of k*k dimension. Non-singular matrix means that matrix must have an inverse matrix (its also called invertible matrix or scrambling matrix). This matrix was considered as the second secret key. The algorithm of creating the non-singular matrix illustrates in algorithm (3.5). But this random creating needs long time especially when the value of the system parameter ($m \geq 6$, $k \geq 57$) therefore instead of it, can read a non-singular matrix and its inverse matrix from files generated by algorithm (3.17) and use these two matrices to generate a McEliece public key.

Algorithm (3.5): Generate a random non-singular matrix and its inverse.

Input:

```
        k: an integer number represents dimensions values of the non-singular matrix
        and its inverse matrix //the value of k determines from algorithm (3.7)
```

Output:

```
        S: a non-singular matrix of k*k dimension
        Sinv: an inverse matrix of non-singular matrix of k*k dimension
```

Procedure

```
        1. Do
            1.1 For i ← 0 to k-1
                    For j ← 0 to k-1
                        Set S(i, j) ← randomize value 0 or 1
                    End loop (j)
                End loop (i)
            1.2 Apply gauss elimination with partial pivoting method to compute
                the inverse matrix of S matrix
          Until (S*Sinv= identity matrix)  //if (S*Sinv) = identity matrix that means S
          have an inverse matrix (Sinv) otherwise S doesn't have inverse.
End.
```

Example (3.1.c):

k = 4

The non singular matrix and its inverse matrix is:

$$S_{\mathbf{(4*4)}}= \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad S^{-1}{}_{\mathbf{(4*4)}} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

Some of matrices do not have an inverse so each matrix must be checked if it has an inverse or not by using gauss elimination with partial pivoting method to compute its inverse matrix. And then check if the multiplication result of these two matrices S and

S^{-1} leads to an identity matrix, then it means S has an inverse, otherwise S doesn't have.

iii. Generate randomly a permutation binary matrix of n*n dimension and generate its inverse. Permutation matrix means that matrix have only one in each row and column. The inverse matrix of permutation matrix is the transpose of permutation matrix. The algorithm of creating a permutation matrix is as follows:

Algorithm (3.6): Generate a random permutation matrix and its inverse.

Input:

n: an integer number represents dimensions values of the permutation matrix and its inverse // the value of n determines from algorithm (3.7)

Output:

P: a permutation matrix of n*n dimension

Pinv: an inverse permutation matrix of n*n dimension

Procedure

```
1. Set x ←0; Set count ←1
2. Set ind(count) ← randomize value between 0 and n-1
3. Do
    3.1 Set P(x, ind(count)) ← 1
    3.2 For j ← 0 to n-1
            If j ≠ ind(count) Then Set P(x, j) ← 0
        End loop (j)
    3.3 Increment x by 1;   Increment count by 1
    3.4 Set flag ← True
    3.5 Do While (flag = True) and (x ≤ n-1)
            Set new1 ← randomize value between 0 and n-1;   Set co ← 0
            For i ← 0 to count - 1
                If new1 ≠ ind(i) then increment co by 1
            End loop (i)
            If co = count then
                Set ind(count) ← new1;   Set flag ← False
                End If
              Loop
          Until x = n
      4. For i ← 0 to n-1
            For j ← 0 to n-1
                Set Pinv(j, i) ← P(i, j)
            End loop (j)
         End loop (i)
End.
```

Example (3.1.d):

n = 7

The permutation matrix and its inverse matrix are:

$$P_{(7*7)} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad P^{-1}_{(7*7)} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The public key of hamming code or extended hamming code can be generated by multiplying these three matrices (generator matrix of hamming code or extended hamming code G_0, non-singular matrix S and permutation matrix P). This showed as follows in figure (3.2) and in algorithm (3.7). This public key is a matrix of k*n dimension:

$$\mathbf{G_{(k,n)} = S_{(k,k)}\, G_{0(k,n)}\, P_{(n,n)}}$$

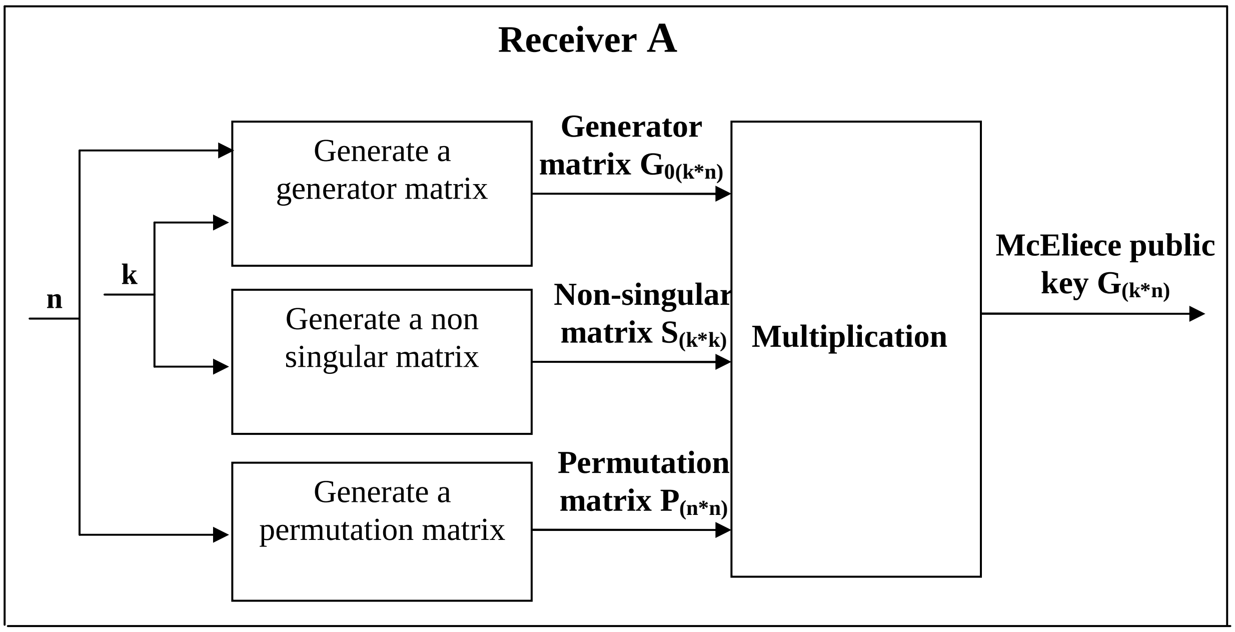
Receiver A
n
k
Generate a generator matrix
Generate a non singular matrix
Generate a permutation matrix
Generator matrix $G_{0(k*n)}$
Non-singular matrix $S_{(k*k)}$
Permutation matrix $P_{(n*n)}$
Multiplication
McEliece public key $G_{(k*n)}$

Figure (3.2): Key generation phase.

<u>Algorithm (3.7):</u> <u>key generation phase.</u>

Input:

```
m: is an integer number, 2≥ m≥ 10 // m=(n-k) for hamming code, m=(n-k)-1
    for the extended hamming code.
t: 0≤t≤2, number of error // (0≤t≤1) for hamming code, and (0≤t≤2) for the
    extended hamming code.
```

Output:

```
G: a public key of hamming code or a public key of extended hamming code
of k*n dimension
t: number of error
time: represents execution time of the key generation phase
```

Procedure

```
1. Set t1← current time
2. If generating a public key of hamming code then   Set n ← $2^m – 1$
   Else if generating a public key of extended hamming code then   Set n ← $2^m$
3. Set k ← $2^m$ - m – 1
4. If generating a public key of hamming code then       Call Generate a
      random generator matrix of hamming code that input (n,k) and output
     ($G_0$,HParity)  // algorithm (3.1)
   Else if generating a public key of extended hamming code then     Call
    Generate a random generator matrix of extended hamming code that input
    (n,k) and output ($G_0$, ExParity) //algorithm (3.4)
  5. Call Generate a random non-singular matrix and its inverse that input (k)
     and output (S,Sinv) // algorithm (3.5)
  6. Call Generate a random permutation matrix and its inverse that input (n)
     and output (P,Pinv) // algorithm (3.6)
  7. Multiply these three matrices to produce a public key (G):
     Set $G(k, n) \leftarrow S(k,k) * G_0(k,n) * P(n,n)$
  8. Set t2← current time
  9. Set time← t2-t1
End.
```

Example (3.1):

The McEliece public key of hamming code is:

$$G_{(4*7)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

The above public key $G_{(4*7)}$ can be obtained by multiplying these three matrices; generator matrix of hamming code $G_{0(4*7)}$ in example (3.1.a), non-singular matrix $S_{(4*4)}$ in example (3.1.c) and permutation matrix $P_{(7*7)}$ in example (3.1.d). That showed as follows:

$$G_{(4,7)} = S_{(4,4)} * G_{0(4,7)} * P_{(7,7)}$$

3.3 Encryption Phase

In this phase the original message will be converted to a cipher message by multiplying the original one (the original message or the clear message is a binary vector of 1*k dimension after converted from text to binary representation) with the public key that was generated with the previous phase. Also in this phase, the errors have been added (one error when using hamming code and one or two errors when using extended hamming code, these errors refer to the noise in the channel) to the encrypted message. This is shown in figure (3.3) and the implemented steps are presented in algorithm (3.8). The encrypted message is a binary vector of 1*n dimension:

Encrypted message $y1_{(1,n)}$ = original message $me_{(1,k)} * G_{(k,n)}$ + error vector $e_{(1,n)}$

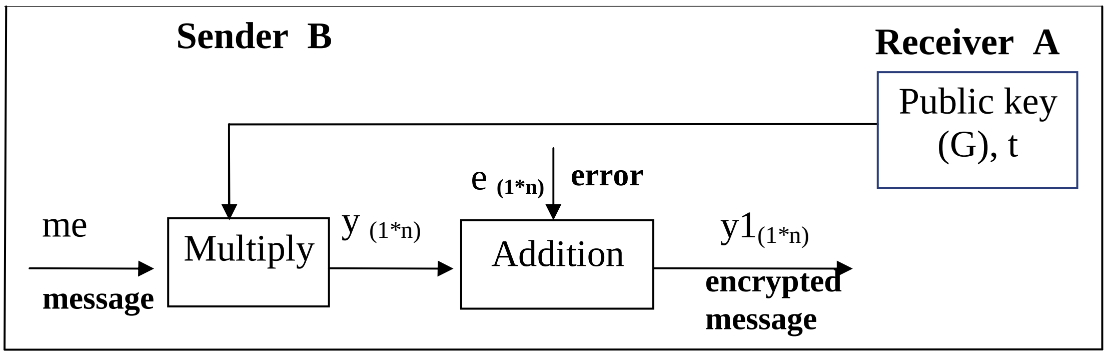

Figure (3.3): Encryption phase.

<u>Algorithm (3.8):</u> <u>Encryption phase.</u>

Input:

```
G: a public key of hamming code or a public key of extended hamming code of k*n dimension
t: number of error
me: represents original message
```

Output:

```
y1: an encrypted message with error
time: represents execution time of the encryption phase
```

Procedure

```
1. Sender (B) obtained A's public key G and t
2. Convert the original message (me) from text to binary representation
3. Set t1← current time
4. q ← length (me) / k
5. If q ≠ Int(q) Then
       q ← Int(q) + 1
   End If
6. Call Message encryption that input (G,me,q) and output (y)//algorithm (3.9)
7. If t ≠ 0 Then
       For x ← 0 to q - 1
           Call Generate a random error vector that input (t) and output (e) //
```

```
              algorithm (3.10)
             For j ← 0 to n - 1
                Set y1(x, j) ← (y(x, j) + e(j)) and 1
             End loop (j)
          End loop (x)
       End If
    8. Set t2← current time
    9. Set time← t2-t1
    10. Convert the encrypted message (y1) from binary to text representation
    11. Sender (B) Sends encrypted message y1 to receiver (A)
End.
```

Algorithm (3.9): Message encryption.

Input:

G: a public key of hamming code or a public key of extended hamming code of k*n dimension

me: represents original message

q: number of blocks of the original message // each block of 1*k dimension

Output:

y: encrypted message of q*n dimension

Procedure

```
    1. Set co ←0; Set co1 ←k
    2. For l ← 0 to q - 1
       2.1 For j ← 0 to n - 1
             Set x ← 0;     Set x1 ← 0
             For i ← co to k - 1
                Set r1 ← me(i) * G(x1, j)
                Increment x by r1;   Increment x1 by 1
             End loop (i)
             Set y(l, j) ←x and 1
          End loop (j)
       2.2 Increment co by co1; Increment k by co1
       End loop (l)
End.
```

Errors which have been added to the encrypted message are already arranged in a binary vector of 1*n dimension. These errors must be generated randomly. This random generation was as follows:

Algorithm (3.10): Generate a random error vector.

Input:

t: number of error

Output:

e: an error vector of 1*n dimension

Procedure

```
1. Set x←1; Set ind1←randomize value between 0 and n-1; Set index(x)←ind1
2. Do While x < t
     2.1 Set ind1 ← randomize value between 0 and n-1;     Set co ← 0
     2.2 For i ← 1 to x
             If ind1 = index(i) Then
               Set co ← 1
               Exit For (i)
             End If
         End loop (i)
     2.3 If co = 0 Then
             Increment x by 1
             Set index(x) ← ind1
         End If
   Loop
3. For i ← 0 to n - 1
      For j ← 1 to x
         If index(j) = i Then
            Set e(i) ← 1
            Exit For (j)
         End If
         Set e(i) ← 0
      End loop (j)
   End loop (i)
End.
```

Example (3.2):

Let original message me = 1 1 0 1

Let error vector e = 0 0 0 0 1 0 0

Encrypted message can be obtained by multiplying the original message $me_{(1*4)}$ with the public key $G_{(4*7)}$ that was generated in example (3.1), and then adding error vector $e_{(1*7)}$ to it. This will be as follows:

Encrypted message $y1_{(1*7)} = me_{(1*4)} * G_{(4*7)} + e_{(1*7)}$

$= 0\ 1\ 1\ 0\ 0\ 1\ 0 + 0\ 0\ 0\ 0\ 1\ 0\ 0$

$= 0\ 1\ 1\ 0\ 1\ 1\ 0$

3.4 Decryption Phase

In this phase the encrypted message is converted to an original message by using three secret keys (generator matrix, non-singular matrix and permutation matrix). Also, in this phase the message was decoded from the error added to it. The decryption process is done by multiplying the encrypted message by the inverse matrix of the permutation matrix (that was generated through the first phase (key generation phase)) to produce a binary vector of 1*n dimension. This binary vector was multiplied by the transpose of parity check matrix (H). The parity check matrix is a matrix of n*(n-k) dimension, which depends on the generator matrix (generator matrix G_0= [I:P], parity check matrix H = $[P^T:I_{n-k}]$). This matrix consists of all non-zero binary rows. The result of previous multiplication is a binary vector of 1*(n-k) dimension called a syndrome. The syndrome was used to discover if the binary vector of 1*n dimension has errors (one error or two) or not. If it has one error, then using the syndrome, the position of the error can be detected and corrected, but if the binary vector has two errors, then those two errors can be detected without knowing their positions.

- Syndrome decoding for hamming code is:
  1. If syndrome equal to zero that means no error exists.
  2. If syndrome not equal to zero, this means one error exists and can be corrected by matching the syndrome with rows of the $H^T_{(n*(n-k))}$. The row number in H^T that is identical to syndrome refers to the position of the error.
- Syndrome decoding for extended hamming code is:
  1. If syndrome equal to zero that means no error exists.
  2. If the first (n-k-1) bits of syndrome equal to zero and the last bit (parity check bit) of syndrome fails, this means one error exists and can be corrected by the same previous method of matching the syndrome with the rows of H^T.
  3. If the first (n-k-1) bits of syndrome not equal to zero and the last bit (parity check bit) of syndrome fails, this means one error exists and it can be corrected also by the same previous method.
  4. If the first (n-k-1) bits of syndrome not equal to zero and the last bit (parity check bit) of syndrome is passed, this means two errors exist and will seek for re-transmission, so using the syndrome two errors in the codeword can be detected but not corrected.

In case of no errors found, or after correcting the binary vector of 1*n dimension with the syndrome, the first k bits of this binary vector was multiplied by the inverse matrix of the non-singular matrix (that was generated in the first phase (key generation phase)) to obtain the original message. The decryption phase was represented in figure (3.4).

$$\mathbf{y2_{(1*n)} = y1_{(1*n)} * Pinv_{(n*n)}}$$

$$\mathbf{y1_{(1*n)} = me_{(1*k)} * G_{(k*n)} + e_{(1*n)}}$$

$$\mathbf{y2_{(1*n)} = (me_{(1*k)} * S_{(k*k)} * G_{0(k*n)} * P_{(n*n)} + e_{(1*n)}) * Pinv_{(n*n)}}$$

$$\mathbf{y2_{(1*n)} = me_{(1*k)} * S_{(k*k)} * G_{0(k*n)} + e_{(1*n)} * Pinv_{(n*n)}}$$

/ ... after using the syndrome procedure,

$$\mathbf{y2_{(1*k)} = me_{(1*k)} * S_{(k*k)}}$$

$$\mathbf{me_{(1*k)} = (me_{(1*k)} * S_{(k*k)}) * Sinv_{(k*k)}}$$

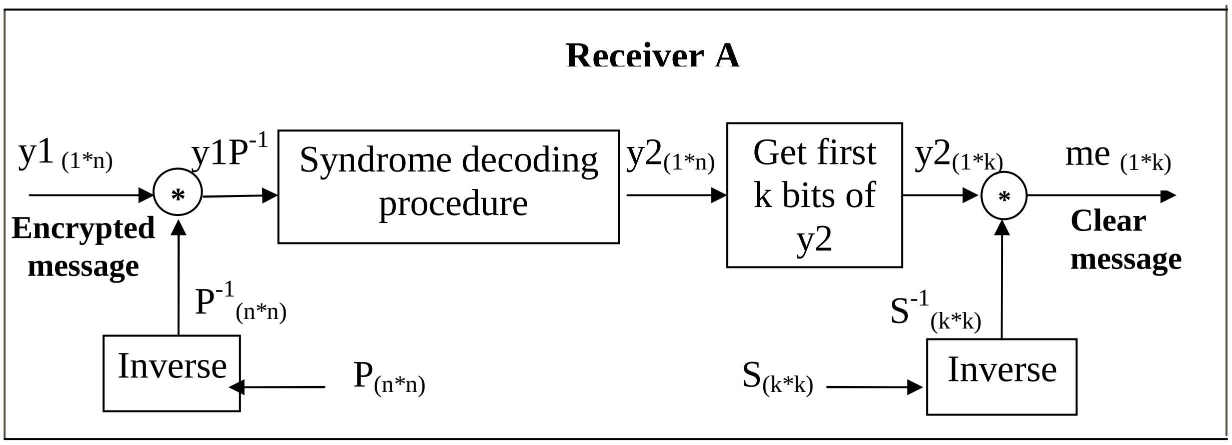

Figure (3.4): Decryption phase.

<u>Algorithm (3.11):</u> <u>Decryption phase (Hamming decryption).</u>

Input:

y1: an encrypted message with error

Pinv: an inverse permutation matrix of n*n dimension

HParity: a parity part of the generator matrix of k*n-k dimension

Sinv: an inverse matrix of non-singular matrix of k*k dimension

Output:

me: represents original message

time: represents execution time of the decryption phase

Procedure

```
1. Receiver (A) receives encrypted message y1 from sender (B)
2. Convert the encrypted message (y1) from text to binary representation
3. Set t1← current time
4. q ← length (y1) / n
5. If q ≠ Int(q) Then
       q ← Int(q) + 1
   End If
6. Call SynPar that input (y1,q,Pinv,HParity) and output (Sy,$H^T$,y2) //
    algorithm (3.13)
7. Set x1 ← 0
8. For j ← 0 to (n - k) - 1
```

```
          Set x1 ← x1 + Sy( 0,j)
        End loop (j)
      **9.** If (x1 = 0) Then   no error added to the message
      **10.** If (x1 ≠ 0) Then
           **10.1** For r ← 0 to q – 1 // error correction loop
                 For i ← 0 to n - 1
                   Set co ← 0
                   For j ← 0 to (n - k) - 1
                     If Sy(r,j) ≠ $H^T$(i, j) Then
                        Set co ← 1
                        Exit For (j)
                     End If
                   End loop (j)
                   If co = 0 Then
                      Set y2(r,i) ← (y2(r,i) + 1) and 1
                      Exit for (i)
                   End If
                 End loop (i)
               End loop (r)
         End if
      **11.** For r ← 0 to q – 1 // eliminates the non-singular matrix (it is a secret key)
           For i ← 0 to k - 1
             Set x1 ← 0
             For j ← 0 to k - 1
               Set r1 ← y2(r, j) * Sinv(j, i)
               Increment x1 by r1
             End loop (j)
             Set me(r, i) ← x1 and 1
           End loop (i)
         End loop (r)
      **12.** Set t2← current time
      **13.** Set time←t2-t1
      **14.** Convert the original message (me) from binary to text representation
**End.**
```

The previous algorithm (3.11) illustrated the decryption phase of hamming code while the next algorithm (3.12) illustrates decryption phase of the extended hamming code.

Algorithm (3.12): Decryption phase (Extended hamming decryption).

Input:

y1: an encrypted message with error

Pinv: an inverse permutation matrix of n*n dimension

ExParity: a parity part of the generator matrix of k*n-k dimension

Sinv: an inverse matrix of non-singular matrix of k*k dimension

Output:

me: represents original message

time: represents execution time of the decryption phase

Procedure

```
1. Receiver (A) receives encrypted message y1 from sender (B)
2. Convert the encrypted message (y1) from text to binary representation
3. Set t1← current time
4. q ← length (y1) / n
5. If q ≠ Int(q) Then
       q ← Int(q) + 1
   End If
6. Call SynPar that input (y1,q,Pinv,ExParity) and output (Sy,$H^T$,y2) //
    algorithm (3.13)
7. Set x1 ← 0
8. For j ← 0 to (n - k) - 2
       Set x1 ← x1 + Sy(0,j)
   End loop (j)
9. Set x ← x1 and 1
10. If ((x1 = 0) And (Sy(0,(n - k) - 1) = 0)) Then no error add to the message.
11. If ((x1 ≠ 0) And (x = Sy(0,(n - k) - 1))) Then
        Two errors has been detected
        Return  // end algoritm
    End If
12. If (((x1 = 0) And (Sy(0,(n - k) - 1) = 1)) Or ((x1 ≠ 0) And (x ≠ Sy(0,(n –
            k) – 1)))) Then
              Do step 11.1 in algorithm (3.11)
            End if
        13. Do step 12 in algorithm (3.11)
        14. Set t2← current time
        15. Set time←t2-t1
        16. Convert the original message (me) from binary to text representation
End.
```

Algorithm (3.13): SynPar.

Input:

y1: an encrypted message with error

q: number of blocks of the encrypted message // each block of 1*n dimension

Pinv: an inverse permutation matrix of n*n dimension

Parity: a parity part of the generator matrix of k*(n-k) dimension

Output:

Sy: syndrome of q*(n-k) dimension

H^T: a parity check matrix of n*n-k dimension

y2: the result of multiplying (y1) by (Pinv) of q*n dimension

Procedure

```
1. Set r ← 0;   Set co1 ← n
2. For l← 0 to q - 1
    2.1 For j ← 0 to co1 - 1
          Set x ← 0;   Set x1 ← 0
          For i ← r to n - 1
             Set r1 ← y1(i) * Pinv(x1, j)
             Increment x by r1;    Increment x1 by 1
          End loop (i)
          Set y2(l, j) ← x and 1
        End loop (j)
    2.2 Increment r by co1;    Increment n by co1
  End loop (l)
3. For i ← 0 to k - 1
    For j ← 0 to (n - k) - 1
```

```
            Set $H^T(i, j) \leftarrow Parity(i, j)$
          End loop (j)
        End loop (i)
      4. Call generate identity matrix that input (n-k) and output (id)//algorithm (3.2)
      5. Set $x \leftarrow 0$
      6. For $i \leftarrow k$ to $n - 1$
          6.1 For $j \leftarrow 0$ to $(n - k) - 1$
                 Set $H^T(i, j) \leftarrow id(x, j)$
               End loop (j)
          6.2 Increment x by 1
        End loop (i)
      7. For $x \leftarrow 0$ to $q - 1$
           For $j \leftarrow 0$ to $(n - k) - 1$
             Set $x1 \leftarrow 0$
             For $i \leftarrow 0$ to $n - 1$
               Set $r1 \leftarrow y2(x, i) * H^T(i, j)$
               $x1 \leftarrow x1 + r1$
             End loop (i)
             Set $Sy(x, j) \leftarrow x1$ and 1
           End loop (j)
         End loop (x)
End.
```

Example (3.3):

Let the encrypted message y1 = 0 1 1 0 1 1 0 (from example (3.2))

The original message can be obtained by multiplying the encrypted message $y1_{(1*7)}$ with the inverse matrix of permutation matrix $P^{-1}_{(7*7)}$ (from example (3.1.d)) to produce another binary vector $y2_{(1*7)}$. This binary vector must be corrected by multiplying it with the transpose of parity check matrix $H_{(7*3)}$ (that depends on the generator matrix in example (3.1.a)). This previous multiplication gives the syndrome $Sy_{(1*3)}$.

$$G_{0(4*7)} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right]$$

$$H^{T}_{(7*3)} = \left[\begin{array}{ccc} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]$$

$$y2_{(1*7)} = y1_{(1*7)} * P^{-1}_{(7*7)}$$

$y2 = 1\ 0\ 0\ 0\ 1\ 1\ 1$

$Sy_{(1*3)} = y2_{(1*7)} * H^{T}_{(7*3)}$

$Sy = 0\ 0\ 1$

From matching this syndrome with the rows of H^T finding that the syndrome equal to the seventh row in matrix, so the error occurs in position 7. The codeword after correction will be = (1 0 0 0 1 1 0), and then multiplying the first k bits of this corrected codeword with the inverse matrix of non-singular matrix to obtain the original message.

$me_{(1*4)} * S_{(4*4)} = (1\ 0\ 0\ 0)$, and obtain original message by:

$me_{(1*4)} = (me_{(1*4)} * S_{(4*4)}) * S^{-1}_{(4*4)}$

$me_{(1*4)} = (1\ 0\ 0\ 0) * S^{-1}$

Original message $me_{(1*4)} = (1\ 1\ 0\ 1)$

3.5 Brute Force Attack

In order to evaluate the security of McEliece cryptosystem, a brute force attack was used. Brute force attack is a method of defeating the system by generating all possible probabilities for each secret key that participates in the creation of the public key, and working through all these possible keys in order to decrypt a message. All possible keys for each secret key that participates in the generation of McEliece public key was generated as follows:

1. Generate all possible probable binary generator matrices (secret key). These probabilities depend on the properties of the binary generator matrix (The minimum distance between any two codewords equal to 3, and the minimum distance of linear block code is equal to the minimum weight of any nonzero word in the code). Here the operation of creating all generator matrix probabilities is based on the second part of this matrix (parity matrix) because the first part (identity matrix) has a fixed structure and its elements have fixed value. The minimum weight for each codeword was obtained from the two ones of the second part and the single one of the first part. As a result the minimum weight for each codeword in the generator matrix equal to 3. This parity matrix must be linearly independent. And the number of all possible probabilities depends on the size of the key (generator matrix length). The algorithm of generating all possible probabilities of the generator matrix illustrates in algorithm (3.14).

Algorithm (3.14): Generate all possible probabilities of generator matrix for brute force attack.

Input:

n: an integer number represents dimension value (a y-axes) of the generator matrix

k: an integer number represents dimension value (an x-axes) of the generator matrix

Output:

Gntimes: represents the number of all possible generator matrices

File: file contains all possible generator matrices

Procedure

```
1. Set r ← n-k;    Set Gntimes ← 0
2. Open File for write
3. For i ← 0 to k – 1
       Set $G_0(i)$ ← 3
   End loop (i)
4. Set flag ← False
5. Do
     5.1 Do
           Set flag1 ← False
           For i ← 0 to k - 2
              For j ← i + 1 to k - 1
                 If $G_0(i) = G_0(j)$ Then
                    Increment $G_0(i)$ by 1
                    Call Check number of ones with input ($G_0(i)$,r) and output
                    (no) // algorithm (3.15)
                    If Not no Then increment $G_0(i)$ by 1
                    Set flag1 ← True
                 End If
              End loop (j)
           End loop (i)
        Loop While flag1
     5.2 Set flag2 ← False
     5.3 For i ← 0 to k - 2
           If $G_0(i) > 2^r$ Then
              Set flag2 ← True;     Set $G_0(i)$ ← 3
              Increment $G_0(i + 1)$ by 1
             Call Check number of ones with input ($G_0(i+1)$,r) and output
             (no) // algorithm (3.15)
              If Not no Then increment $G_0(i + 1)$ by 1
           End If
         End loop (i)
     5.4 If Not flag2 Then
           Put File, $G_0$
           Increment Gntimes by 1;     Increment $G_0(0)$ by 1
           Call Check number of ones with input ($G_0(0)$,r) and output
```

```
                     (no)// algorithm (3.15)
                    If Not no Then increment $G_0(0)$ by 1
                    For i ← 0 to k - 2
                        If $G_0(i) > 2^r$ Then
                            Set $G_0(i)$ ← 3
                            Increment $G_0(i + 1)$  by 1
                            Call Check number of ones with input ($G_0(i+1)$,r) and
                              output (no)// algorithm (3.15)
                            If Not no Then increment $G_0(i + 1)$ by 1
                        End If
                    End loop (i)
                End If
          5.5 If $G_0(k - 1) > 2^r$ Then   Set flag ← True
        Until flag
    6. Close File
    7. Convert all probabilities in File from decimal to binary representation
End.
```

Algorithm (3.15): Check number of ones.

Input:

x: an integer number represents codeword value of G_0 matrix

r: an integer number represents the value of (n-k)

Output:

no: boolean value either true or false

Procedure

```
    1. Set no ← True
    2. For i ← 0 to r
           If $x = 2^i$ Then
               Set no ← False
               Exit For
           End if
       End loop
End.
```

The complexity assessment of the number of probable generator matrices was computed by conducting many experiments on generating the possible generator matrices with different matrix length. The algorithm of this complexity assessment is as follows:

Algorithm (3.16): Complexity assessment of the number of probable generator matrices

Input:

n: an integer number represents dimension value (a y-axes)of the generator matrix

k: an integer number represents dimension value (an x-axes) of the generator matrix

Output:

Gntimes1: represents the number of all possible generator matrices

Procedure

```
1. Set r ← n – k;   Set Gntimes1 ← 1;   Set pr ← $2^r$ - r - 1
2. For i ← 1 to k
      2.1 Set Gntimes1← Gntimes1 * pr
      2.2 Decrement pr by 1
   End loop
End.
```

2. Generate all possible probable non-singular matrix. The generation of all possible keys depends on its secret key properties. Non-singular matrix or (scrambling matrix) means that the matrix has an inverse matrix, and every matrix must be checked if it has an inverse matrix or not by using gauss elimination with partial pivoting method to compute its inverse matrix and then comparing the result of multiplication S and S^{-1} with identity matrix, if this multiplication result equal to it means that this matrix S have an inverse otherwise doesn’t have. The number of all possible probabilities depends on the McEliece cryptosystem parameter that determines dimensions of

this secret key (key size). The algorithm of creating all possible probabilities of the non-singular matrix shows in algorithm (3.17).

Algorithm (3.17): Generate All possible probabilities of non-singular matrix for brute force attack.

Input:

k: an integer number represents dimensions values of the non-singular matrix and its inverse matrix //the same k in algorithm (3.14)

Output:

Sntimes: represents the number of all possible non-singular matrices

File1: file contains all possible non-singular matrices

File2: file contains all possible inverse matrices of non-singular matrices

Procedure

```
1. Open File1, File2 for write
2. For i ← 0 to k-1
       For j ← 0 to k-1
           Set a(i,j) ← S(i,j) ← 0
       End loop (j)
   End loop (i)
3. For i ← 0 to k-1
       Set a(i,k-1-i)← S(i,k-1-i) ←1
   End loop (i)
4. Do While (ChangeState =1)
       4.1 For i ← 0 to k-1
               For j ← 0 to k-1
                   Set a(i,j) ← S(i,j)
                   If (i=j) then      Set id(i,j) ←1    else    Set id(i,j) ← 0
               End loop (j)
           End loop (i)
       4.2 Call Compute inverse matrix that input (a,k,id) and output (invS)
           //algorithm (3.18)
       4.3 For x ← 0 to k-1
               For i ← 0 to k-1
                   Set sum ← 0
                   For j ← 0 to k-1
```

```
                Set sum ← sum+S(x,j)*invS(j,i)
              End loop (j)
              Set R(x,i) ← int(sum) and 1
            End loop (i)
          End loop (x)
4.4 Set Identity ← 1
4.5 For i ← 0 to k-1
        For j ← 0 to k-1
          If ((i=j) And (R(i,j)≠1)) then
            Set Identity ← 0;      Set i ←k-1
            Exit for (j)
          End if
          If ((i≠j)And(R(i,j)≠0)) then
            Set Identity ← 0;     Set i ← k-1
            Exit for (j)
          End if
        End loop (j)
    End loop (i)
4.6 If (Identity=1) then
        Put File1,S
        Put File2,invS
        Increment Sntimes by 1
    End if
4.7 Set Carry←1
4.8 For i ←k-1 down to 0
        Set ChangeState ← 0
        For j ← k-1 down to 0
          If ((S(i,j)=0)And(Carry=1)) then
            Set S(i,j)←1; Set Carry ← 0; Set ChangeState← 1;  Set i← 0
            Exit for (j)
          End if
          If ((S(i,j)=1)And(Carry=1)) then
            Set S(i,j) ← 0;   Set Carry ←1
          End if
        End loop (j)
```

```
              End loop (i)
         Loop
      5. Close File1; Close File2
End.
```

Algorithm (3.18): Compute inverse matrix.

Input:

a: a non-singular matrix of k*k dimension

k: an integer number represents dimensions values of the non-singular matrix and its inverse matrix //the same k in algorithm (3.14)

id: an identity matrix of k*k dimension

Output:

invS: the inverse matrix of matrix S of k*k dimension

Procedure

```
1. For j ← 0 to k-1
       Set p(j) ← j
   End loop (j)
2. For co ← 0 to k-2
     2.1 Set l ←co;   Set max ←a (p(co),co)
     2.2 For x ← co+1 to k-1
             If (abs(a (p(x),co))>abs(max)) then
               Set max ← a(p(x),co);   Set l ← x
             End if
         End loop (x)
     2.3 Set temp ← p(l);      Set p(l) ← p(co);      Set p(co)← temp
     2.4 For i ← co+1 to k-1
             Set a(p(i),co) ← a(p(i),co)/a(p(co),co)
             For j ← co+1 to k-1
                Set a(p(i),j) ← int(a(p(i),j) – a(p(i),co)*a(p(co),j)) and 1
             End loop (j)
             For j ← 0 to k-1
                Set id(p(i),j) ← int(id(p(i),j) – a(p(i),co)*id(p(co),j)) and 1
             End loop (j)
         End loop (i)
```

```
    End loop (co)
3. For j ← 0 to k-1
    3.1 Set tt ← int(id(p(k-1),j)/a(p(k-1),k-1))
    3.2 If tt<0 then     Set tt ← tt*-1
    3.3 Set invS(k-1,j)← tt and 1
    End loop (j)
4. For x ← 0 to k-1
    For j ← k-2 down to 0
        Set sum ← id(p(j),x)
        For co ← j+1 to k-1
            Set sum←sum-a(p(j),co)*invS(co,x)
        End loop (co)
        Set tt ← int(sum/a(p(j),j))
        If tt < 0 then     Set tt ← tt *-1
        Set invS(j,x)← tt and 1
    End loop (j)
    End loop (x)
End.
```

In algorithm (3.19) the complexity assessment of the number of probable non-singular matrices was discovered and computed by generating the all possible non-singular matrices with many different matrix dimensions. The algorithm of the complexity assessment for the possible non-singular matrices is as follows:

Algorithm (3.19): Complexity assessment of the number of probable non-singular matrices

Input:

k: an integer number represents dimensions values of the non-singular matrix and its inverse matrix

Output:

Sntimes1: represents the number of all possible non-singular matrices

Procedure

```
1. Set Sntimes1 ← 1;    Set x ← 1;    Set j ← 2
```

```
    2. For i ← 1 to k - 1
        2.1 Set x ← 4 * x + j
        2.2 Set Sntimes1 ← Sntimes1 * x
        2.3 Set j ← j * 2
     End loop
End.
```

3. Generate all possible probabilities of the third secret key (permutation matrix P). The generation of permutation matrix probabilities depends on its secret key properties, and the number of these probabilities depends on the length of the matrix (key size). Permutation matrix means that the binary matrix has only one (1) in each row and column. Every matrix must be checked if it has this property or not. The algorithm of generation shows in algorithm (3.20). The complexity assessment of the number of possible probable permutation matrices is n! (i.e., Permutation matrix $P_{(n*n)}$ is a matrix of n*n dimension).

Algorithm (3.20): Generate all possible probabilities of permutation matrix for brute force attack.

Input:

n: an integer number represents dimensions values of the permutation matrix and its inverse // the same n in algorithm (3.14)

Output:

Pntimes: represents the number of all possible permutation matrices

File3: file contains all possible permutation matrices

Procedure

```
    1. Open File3 for write
    2. For i ← 0 to n-1
          For j ← 0 to n-1
             Set P(i,j) ← 0
          End loop (j)
       End loop (i)
```

```
3. For i ← 0 to n-1
      Set P(i,n-1-i)←1
   End loop (i)
4. Do while (ChangeState=1)
      4.1 For i ← 0 to n-1
             Set Numof1s ← 0
             For j ← 0 to n-1
                If (P(i,j)=1) then  Increment Numof1s by 1
             End loop (j)
             If ((Numof1s<1) or (Numof1s>1)) then exit for (i)
          End loop (i)
      4.2 If (Numof1s=1) then
             For j ← 0 to n-1
                Set Numof1s ← 0
                For i ← 0 to n-1
                   If (P(i,j)=1) then    Increment Numof1s by 1
                End loop (i)
                If ((Numof1s<1) or (Numof1s>1)) then exit for (j)
            End loop (j)
            If (Numof1s=1) then
                Put File3, P
                Increment Pntimes by 1
            End if
          End if
      4.3 Set Carry←1
      4.4 For i ← n-1 down to 0
             Set ChangeState ← 0
             For j ← n-1 down to 0
                If ((P(i,j)=0)And (Carry=1)) then
                   Set P(i,j)←1; Set Carry ← 0; Set ChangeState←1; Set i ← 0
                   Exit for (j)
                End if
                If ((P(i,j)=1) And (Carry=1)) then
                    Set P(i,j) ← 0;  Set Carry ←1
                End if
                End loop (j)
            End loop (i)
      Loop
   5. Close File3
End.
```

The output files (File, File1 and File3) contain all probabilities of the secret keys (G_0, S, P), so one of each secret key probabilities could be chosen to generate the McEliece public key.

After generating all possible probabilities for each secret key G_0SP (generator matrix G_0, non-singular matrix S and permutation matrix P) that was participated in the generation of McEliece public key, the encrypted message was decrypted through all these probabilities and found the original one. The algorithm of trying all these probable keys until finding the original message will be as follows:

Algorithm (3.21): Trying all probable secret keys to find the original message.

Input:

```
   Encrypted message: an encrypted message with error
    n: an integer number represents dimension value (a y-axes) of the public key
    matrix //the same n in algorithm (3.14)
    k: an integer number represents dimension value (an x-axes) of the public key
    matrix //the same k in algorithm (3.14)
    Pntimes: represents the number of all possible permutation matrices
    Sntimes: represents the number of all possible non-singular matrices
    Gntimes: represents the number of all possible generator matrices
```

Output:

```
   Decrypted message: represents original message
```

Procedure

```
    1. Open File, File1, File2, File3 for read
    2. Set countG ← 0; Set  countS ← 0; Set countP ← 0
    3. Do
```

```
        3.1 If countP = Pntimes Then
                Increment countS by 1
                Set countP ← 0
            End If
        3.2 If countS = Sntimes Then
                Increment countG by 1
                Set countS ← 0
            End If
        3.3 If countG = Gntimes Then Exit loop
        3.4 Read countG generator matrix $G_0$ from File // this file contains all
            possible generator matrices
        3.5 Read countS non-singular matrix S from File1 // this file contains all
            possible non-singular matrices
        3.6 Read countS the inverse of non-singular matrix Sinv form File2 //
             this file contains all possible inverse matrices of non-singular
             matrices
        3.7 Read countP permutation matrix P form File3 // this file contains all
            possible permutation matrices
        3.8 Computes the transpose of permutation matrix that is represent
             inverse permutation matrix
        3.9 Read Encrypted message
        3.10 Decrypt Encrypted message by Call Decryption phase //algorithm
              (3.11)
        3.11 If Decrypted message = original message then
                    The Encrypted message was decrypted correctly and the correct
                     message was founded
                     Print Decrypted message
                     Exit loop
              End if
        3.12 Increment countP by 1
    Until (countG ≥ Gtimes)
  4. Close File; Close File1; Close File2; Close File3
End.
```

3.6 A Mechanism to Enhance McEliece Cryptosystem

After applying a brute force attack to evaluate the security of McEliece public key cryptosystem, some weak points have been appeared and noticed (as shown in chapter four) especially when m = 3, that the number of probable secret keys are very small and its very easy to attack the system. So an adopted mechanism to enhance the performance of this cryptosystem was designed and implemented. This mechanism makes the cryptosystem very secure (because the number of probable secret keys becomes very large) and the size of the public keys is not large, also has less complexity.

This mechanism based on generating a number of public keys instead of one public key and kept these public keys and its secret keys in a file. This is shown in figure (3.5) and the implemented steps are presented in algorithm (3.22). For example generating three public keys of 4*7 dimensions, and saved these public keys and its private keys in a file.

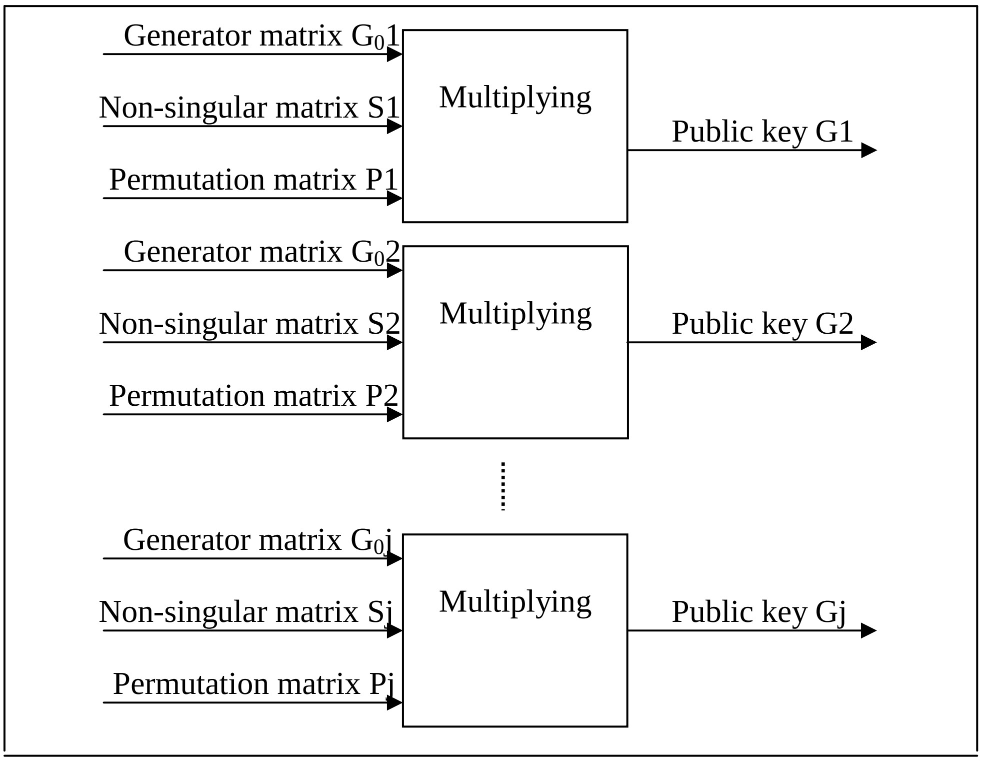

Figure (3.5): Key generation phase for the mechanism.

Algorithm (3.22): key generation phase for the mechanism.

Input:

m: an integer number, $2 \geq m \geq 10$, $m=(n-k)$ in hamming code, $m=(n-k)-1$ in extended hamming code.

t: $0 \leq t \leq 2$, number of error

counter: represents number of generation public keys

Output:

File4: file contains generated public keys of hamming code or extended hamming code G of k*n dimension

FileG$_0$: file contains read generator matrices

FileSinv: file contains the inverse matrices of read non-singular matrices

FilePinv: file contains the inverse matrices of read permutation matrices

t: number of error

Procedure

```
1. Open File, File1, File3 for read
2. Open File4, FileG$_0$, FileSinv, FilePinv for write
3. If generating a public key of hamming code then   Set $n \leftarrow 2^m - 1$
   Else if generating a public key of extended hamming code then   Set $n \leftarrow 2^m$
4. Set $k \leftarrow 2^m - m - 1$
5. For $j \leftarrow 1$ to counter
    5.1 Read randomly the generator matrix $G_0$ from File // this file contains
        all possible generator matrices
    5.2 Put FileG$_0$, $G_0$ // save matrix $G_0$ in file
    5.3 Read randomly the non-singular matrix S from File1 // this file
        contains all possible non-singular matrices
    5.4 Put FileSinv, Sinv // save the inverse of matrix S in file
    5.5 Read randomly the permutation matrix P from File3 // this file
        contains all possible permutation matrices
    5.6 Put FilePinv, Pinv // save the inverse of matrix P in file
    5.7 Multiply these three matrices ($G_0$,S,P)to produce the public key (G):
          $G(k, n) \leftarrow S(k,k) * G_0(k,n) * P(n,n)$
    5.8 Put Filenam4, G
   End loop (j)
```

```
        6. Close File; Close File1; Close File3
        7. Close File4; Close FileG$_0$; Close FileSinv; Colse FilePinv
End.
```

In encryption phase, the side B obtains A’s public keys then start encrypting the original message by multiplying it by the first public key to produce another message. An error will be added to this message. The process of multiplying the message by the public key and adding error to it will be repeated depend on the number of generated public keys. For example if the number of public keys is 3, so the message will be multiplied by the first public key and then add error to it to produce a message that will be multiplied by the second public key and again add error to it. Finally the produced message will be multiplied by the third public key and error will be added to it. The last message represented the encrypted message which will be sent to A. This is shown in figure (3.6) and in algorithm (3.23).

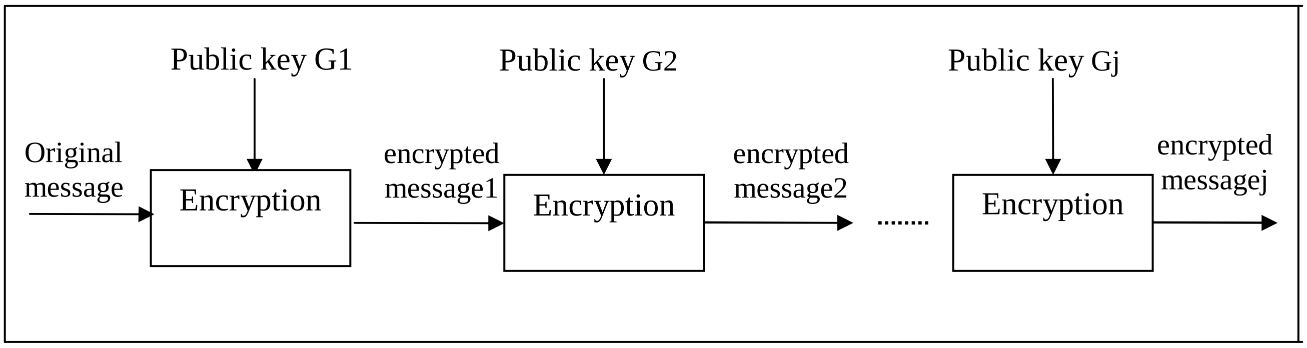

Figure (3.6): Encryption phase for the mechanism.

<u>Algorithm (3.23):</u> <u>Encryption phase for the mechanism.</u>

Input:

```
        counter: represents number of generation public keys
        me: represents the original message
        t: number of error
```

Output:

y1: an encrypted message with error

Procedure

```
1. Sender (B) obtained A's file of public key G and t
2. Convert the original message (me) from text to binary representation
3. Open File4 for read
4. For j ← 1 to counter
   4.1 Read the (j) public key G from File4 // this file contains generated
       public key
   4.2 q ← length (me) / k
   4.3 If q ≠ Int(q) Then
         q ← Int(q) + 1
       End If
   4.4 Call Message encryption that input (G,me,q) and output (y)
        // algorithm (3.9)
   4.5 If t ≠ 0 Then
         For x ← 0 to q - 1
           Call Generate a random error vector that input (t) and output (e)
             // algorithm (3.10)
           For xx ← 0 to n - 1
              Set y1(x, xx) ← (y(x, xx) + e(xx)) and 1
           End loop (xx)
         End loop (x)
       End If
   4.6 Set r ← 0
   4.7 For i ← 0 to q – 1
         For l ← 0 to n - 1
           Set me(r) ← y1(i,l)
           Increment r by 1
         End loop (l)
       End loop (i)
   End loop (j)
5. Close File4
```

```
    6. Convert last encrypted message (y1) from binary to text representation
    7. Sender (B) Sends encrypted message y1 to receiver (A)
End.
```

In the last phase, decryption phase, the side A receives an encrypted message from side B to decrypt it. Decryption process uses the secret keys of the generated public keys by A to decrypt the received message, starting from the secret keys of the last public key to the secret keys of the first public key. For example if the number of public keys is 3, so decryption process will start with the secret keys of the third public key to produce a message which will be decrypted using the secret keys of the second public key to produce another message. Finally this message will be decrypted using the secret keys of the first public key. The last message represents the original message that A encrypt. This is shown as follows in figure (3.7) and the implemented steps are presented in algorithm (3.24).

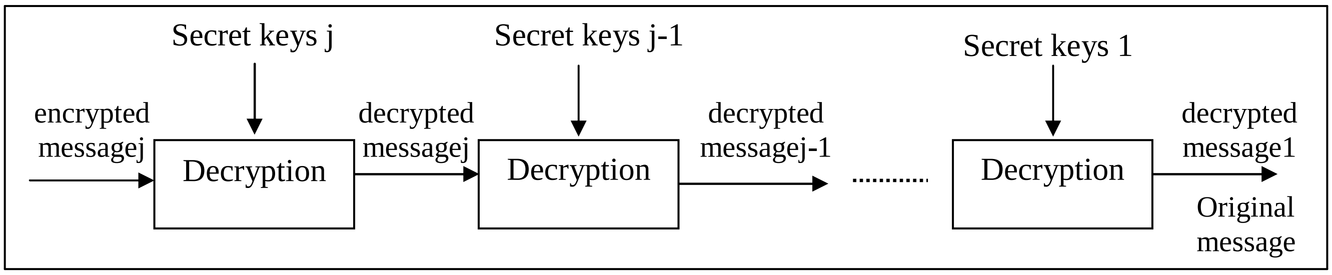

Figure (3.7): Decryption phase for the mechanism.

Algorithm (3.24): Decryption phase for the mechanism.

Input:

```
    counter: represents number of generation public keys
    y1: an encrypted message with error
```

Output:

```
    me: represents the original message
```

Procedure

```
    1. Receiver (A) receives encrypted message y1 from sender (B)
```

```
2. Convert the encrypted message (y1) from text to binary representation
3. Open FileG$_0$, FileSinv, FilePinv for read
4. For j ← counter down to 1
     4.1 q ← length (y1) / n
     4.2 If q ≠ Int(q) Then
               q ← Int(q) + 1
         End If
     4.3 Read (j) inverse permutation matrix Pinv from FilePinv
     4.4 Read (j) generator matrix G$_0$ and Parity part from FileG$_0$
     4.5 Call SynPar that input (y1,q,Pinv,Parity) and output (Sy,H$^T$,y2) //
           algorithm (3.13)
     4.6 Used the syndrome (Sy) to detect and correct the error
           If one error was existed then
               For r ← 0 to q – 1 // error correction loop
                   For i ← 0 to n - 1
                       Set co ← 0
                       For l ← 0 to (n - k) - 1
                           If Sy(r,l) ≠ H$^T$(i, l) Then
                               Set co ← 1
                               Exit For (l)
                           End If
                       End loop (l)
                       If co = 0 Then
                           Set y2(r,i) ← (y2(r,i) + 1) and 1
                           Exit for (i)
                       End If
                   End loop (i)
               End loop (r)
           End if
     4.7 Read (j) inverse non-singular matrix Sinv from FileSinv
     4.8 For r ← 0 to q – 1//eliminates non-singular matrix (it is a secret key)
           For i ← 0 to k - 1
               Set x1 ← 0
               For l ← 0 to k - 1
                   Set r1 ← y2(r, l) * Sinv(l, i)
```

```
                    Increment x1 by r1
                  End loop (l)
                  Set me(r, i) ← x1 and 1
                End loop (i)
              End loop (r)
          4.9 Set r ← 0
          4.10 For i ← 0 to q – 1
                 For l ← 0 to k - 1
                   Set y1(r) ←me (i,l)
                   Increment r by 1
                 End loop (l)
               End loop (i)
        End for (j)
      5. Close FileG0; Close FileSinv; Close FilePinv
      6. Convert last decrypted message (me) that represents original message from
         binary to text representation
End.
```

So the number of all possible probable secret keys becomes very large and the time needed to trying all these probable keys until decrypt the encrypted message is very long, therefore it is very difficult to break and attack the system. And the size of the generated public keys is still suitable.

3.7 Other Cryptosystem Parameters

The McEliece public key cryptosystem has many parameters, some of these parameters are:

1. Key size: represents size of the public key.
2. Message expansion: represents the ratio of the expansion of the encrypted message to the original message (original message is a binary vector of 1*k dimension where the encrypted message is a

binary vector of 1*n dimension, so the encrypted message was expanded n-k bits more than the original message).

3. Information rate: this is a measure of how much information on the average is being carried by a symbol.

Algorithm (3.25): Public key size, message expansion, information rate.

Input:

n: an integer number represents dimension value (a y-axes) of the public key matrix

k: an integer number represents dimension value (an x-axes) of the public key matrix

Output:

keysize: represents size of the public key in a byte

mesexp: represents message expansion

infra: represents information rate

```
Procedure
    1. Set keysize ← (k*n) / 8
    2. Set mesexp ← (n / k)
    3. Set infra ← (k / n)
End.
```

Chapter Four

Experimental Results

4.1 Introduction

In this chapter the results of the proposed system were presented. The security of the McEliece public key cryptosystem was evaluated by using brute force attack; different results of this attack are presented. An adopted mechanism was used to overcome the weakness of this cryptosystem. Also, this chapter presents the results of computing public key size, message expansion, information rate and execution time, and their influence on the security of the cryptosystem.

4.2 Brute Force Attack

A brute force attack was used to evaluate the security of this cryptosystem (McEliece cryptosystem). This attack is a method of defeating a cryptographic scheme by generating all possible probabilities for each secret key. These secret keys are; generator key, non-singular key and permutation key. And, exhaustively working through all possible keys in order to decrypt a message.

One can determine or estimate the time that is required to compute the secret keys, which is refer to the minimum amount of work to compute the key, then if this time is large enough the cryptosystem is a secure system. So the number of probabilities of the secret keys (i.e. the number of keys or the size of the key space) should be large enough to make this attack (brute force) computationally infeasible.

The results of probabilities number for each secret key are as follows:

a. The number of all possible probabilities of the first secret key (generator matrix $G_{0\ (k*n)}$) will be presented in table (4.1). This table presents all possible keys for the generator matrix with different cryptosystem parameter value (means that with different key size).

Table (4.1): The number of probable generator matrices.

m	Hamming code		Extended hamming code		The number of (G_0) probabilities	Time(second)
	n	k	n	k		
2	3	1	4	1	1	0.000067
3	7	4	8	4	24	0.000967
4	15	11	16	11	39916800	7145.656
5	31	26	32	26	4.033e+26	Long time
6	63	57	64	57	4.053e+76	=
7	127	120	128	120	6.689e+198	=
8	255	247	256	247	2.094e+485	=
9	511	502	512	502	3.069e+1139	=
10	1023	1013	1024	1013	4.405e+2606	=

Figure (4.1) presents the relationship between increasing the number of generator matrix probabilities and increasing the value of cryptosystem parameter (i.e. increasing key size). It is found that the number of these probable generator matrices is increased polynomially.

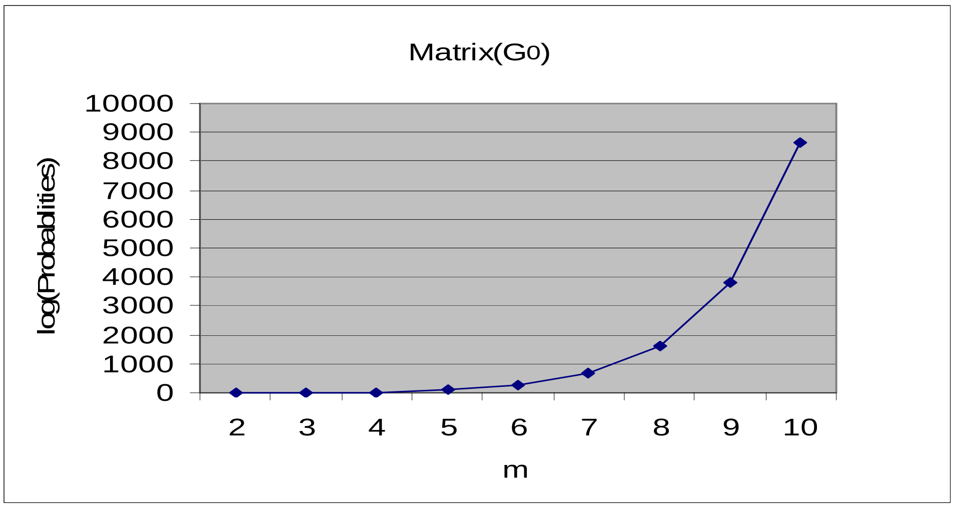

Figure (4.1): Relation between the number of G_0 probabilities and cryptosystem parameter (m).

b. The second secret key (non-singular matrix $S_{(k*k)}$) probabilities number is shown in table (4.2). The number of all possible probabilities for the non-singular matrix with different matrix length are presented in the next table.

Table (4.2): The number of probable non-singular matrices.

m	Hamming code		Extended hamming code		The number of (S) probabilities	Time(second)
	n	k	n	k		
2	3	1	4	1	1	0.0000098
3	7	4	8	4	20160	0.79375
4	15	11	16	11	7.681e+35	Long time
5	31	26	32	26	9.054e+202	=

Figure (4.2) shows the relation between the number of non-singular matrices with the cryptosystem parameter value (m), which represents the matrix size. The number of probable non-singular matrices is increased polynomially.

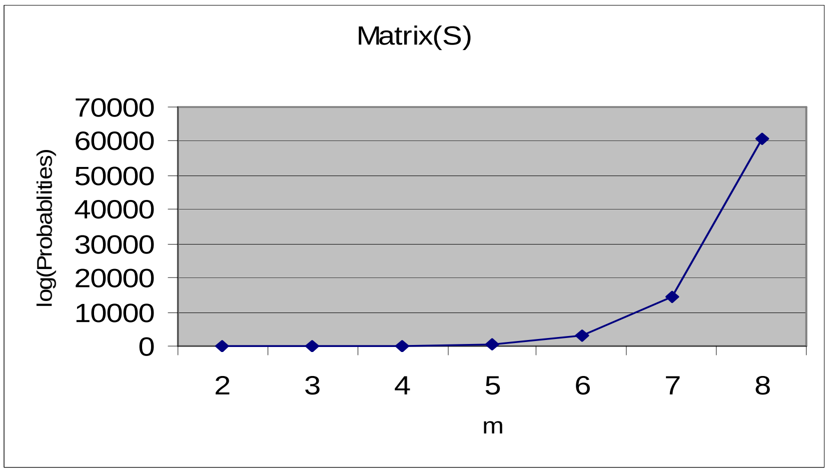

Figure (4.2): Relation between the number of S probabilities and cryptosystem parameter (m).

c. Permutation matrix ($P_{(n*n)}$) is the third secret key. The numbers of possible permutation matrices are listed in tables (4.3), and (4.4). These two tables offer the number of all possible probabilities for the permutation matrix when using hamming and extended hamming code with variable matrix size and different length.

Table (4.3): The number of probable permutation matrices when using hamming code.

m	n	k	The number of (P) probabilities	Time(second)
2	3	1	6	0.000093
3	7	4	5040	Long time
4	15	11	1307674368000	Long time
5	31	26	8.223e+33	Long time
6	63	57	1.983e+87	=
7	127	120	3.013e+213	=
8	255	247	3.351e+504	=
9	511	502	6.792e+1163	=
10	1023	1013	5.292e+2636	=

Figures (4.3) and (4.4) offer the relationship between the numbers of permutation matrices with the value of cryptosystem parameter when using hamming code and extended hamming code (i.e. with increasing matrix length). Also it is found the number of probable permutation matrices is increased polynomially.

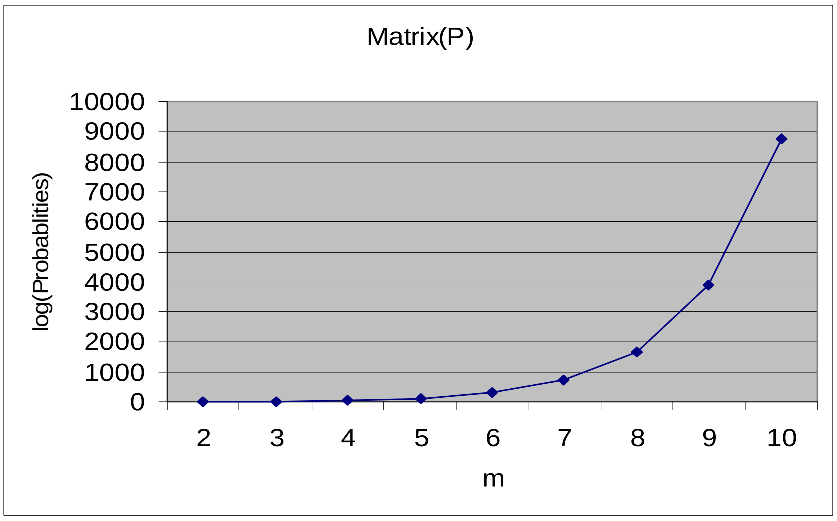

Figure (4.3): Relation between the number of P probabilities and cryptosystem parameter (m) when using hamming code.

Table (4.4): The number of probable permutation matrices when using extended hamming code.

m	n	k	The number of (P) probabilities	Time(in second)
2	4	1	24	0.00856
3	8	4	40320	Long time
4	16	11	20922789888000	Long time
5	32	26	2.631e+35	Long time
6	64	57	1.269e+89	=
7	128	120	3.856e+215	=
8	256	247	8.578e+506	=
9	512	502	3.477e+1166	=
10	1024	1013	5.419e+2639	=

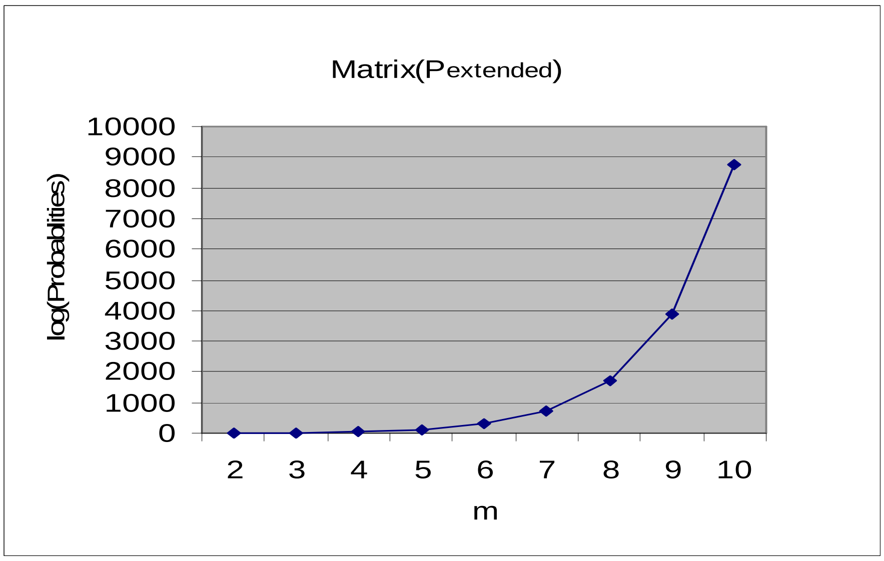

Figure (4.4): Relation between the number of P probabilities and cryptosystem parameter (m) when using extended hamming code.

Complexity assessment of the number of probable each secret key was discovered after conducting many experiments of generating each the possible secret keys with different key length.

Table (4.5): The number of probable secret keys and needed time to try these probabilities.

m	The number of secret keys probabilities (hamming)	Time	The number of secret keys probabilities (extended hamming)	Time
2	6	0.0000000108s	24	0.0000000768s
3	2438553600	2.275$_{min}$	19508428800	22.889$_{min}$
4	4.009e+55	1.049e+42years	6.415e+56	1.861e+43years
5	3.002e+263	8.595e+250years	9.607e+264	2.889e+252years

From results listed in these tables, it was found that when the value of cryptosystem parameter m is small (when m = 2, n = 3 and k = 1 using hamming code, m = 2, n = 4 and k = 1 using extended hamming code), the results of all possible secret key are:

1- For the first secret key (generator matrix), the number of all possible keys = 1.
2- For the second secret key (non-singular matrix), the number of all possible keys = 1.
3- For the third secret key (permutation matrix), the number of all possible keys when using hamming code = 6 and when using extended hamming code = 24.

The number of probable keys is very small and the time needed to break the cryptosystem and find the original message is very small.

When the value of the cryptosystem parameter m = 3, (n = 7 and k = 4) using hamming code, and (n = 8, k = 4) using extended hamming code, the possible secret key are:

1- For G_0 equal to 24.
2- For S equal to 20160.
3- For P using hamming code equal to 5040 and by using extended hamming code equal to 40320.

These values mean that the number of all probable keys is 2438553600 when using hamming code and 19508428800 when using extended hamming code which is greater than the previous case. If a high-speed computer performs 10^{10} operation per second and if the operation is supposed to be multiplication and addition operations then it needs 2.275 or 22.889 minutes to try all the probabilities (each probable has 308 multiplications and 252 additions when using hamming code, 384 multiplications and 320 additions when using extended hamming code) which is considered as a little time and thus not secure.

When m = 4, (n = 15 and k = 11) using hamming code and (n = 16 and k = 11) when using extended hamming code, all possible keys for each secret key are as follows:

1- For G_0 = 39916800.
2- For S = 7.681e+35.
3- When using hamming code P probabilities number = 1307674368000 and when using extended one P probabilities number = 20922789888000.

Here the probabilities number have been increased more than the previous case (when m = 3) and this increase makes such cryptosystem secure. As before a high-speed computer performs 10^{10} operation per second and if the operation is supposed to be multiplication and addition operations then it needs approximately 1.049e+42 or 1.861e+43 years to try all these probabilities (each probable has 8250 multiplications and additions when using hamming code, 9152 when using extended hamming code) which is regarded as a long time and hence the system is secure. This long time resulted from the large number of S probabilities’.

When m = 5 to 10, the number of possible probabilities for all secret keys are very large (when m=5, number of probabilities equal to

3.002e+263 or 9.607e+264), and the time needed for trying all these probabilities is very long and needs very large number of years to break the cryptosystem (the previous case when m=4, the time needed to break the system is very long, so what about this case when m≥5 and the number of probabilities has been increased more and becomes very large). So in this case the McEliece cryptosystem is very secure against this type of attack and very difficult to break.

So, when the value of m is equal to 2; the number of all possible probable keys is very small and it is very easy to attack this cryptosystem. When m is equal to 3; the number of probabilities is also small but better than the number of probabilities when m is equal to 2. So to make this cryptosystem secure against this attack when the value of its parameter m = 3, generate number of public keys and encrypting the message by them. After that decrypt the message by the secret keys of these public keys. So the number of possible keys becomes very large and needs very long time for trying all these probabilities until decrypt the encrypted message, therefore it is very difficult to break the system.

When m = 3 the number of possible secret keys is very small (2438553600), after using the proposed mechanism the number of probable secret keys becomes equal to 1.450e+28 when generating three public keys i.e. the number of secret keys probabilities when generating one public key is much less that when generating three public keys using the proposed mechanism, the number of all probable secret keys when using the proposed mechanism can be illustrated in the following equation:

$$Spm = (Sp)^N \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.1)$$

Spm = Number of all possible probable secret keys

Sp = Number of probable secret keys when generating one public key

N = Number of generated public keys

And if generate ten public keys the number of probable keys equal to $(Sp)^{10}$ =$7.435e+93^{10}$, and so on for m=4 as shown in the next table.

Table (4.6): Number of probable keys after using the mechanism.

m	No. of public keys	All possible probable secret keys
3	3	$(2438553600)^3$ = 1.450e+28
3	5	$(2438553600)^5$ = 8.623e+46
3	10	$(2438553600)^{10}$ = 7.435e+93
4	3	$(4.009e+55)^3$ = 6.443e+166

When m is equal to 4 the number of probabilities is large and the system is secure, in order to make it more secure, the same previous mechanism can be used by generating two or three public keys and encrypting the message by them. This mechanism is not useful when m equal to 2 because the number of probabilities that can be generated from generating large numbers of public keys still very small and therefore, the system can be broken.

4.3 Evaluation According Other Cryptosystem Parameters

Some of McEliece cryptosystem parameters were studied in this project. The security, implementation and use of this system were influenced by these parameters. These parameters are:

1-Public Key Size

Table (4.7) shows the effect of different McEliece public key size (when using hamming code and extended hamming code). Also from this table, the public key size of extended hamming code was larger than the public key size of hamming code. When the value of n and k is small (when n = 3 or 4 and k = 1, and n = 7 or 8 and k = 4) then the public key size is small and needs very little space in the memory. But on the other hand, the number of the secret keys is very small and the cryptosystem is very easy to be attacked.

Table (4.7): McEliece public key size when using hamming and extended hamming code in byte.

m	Hamming code			Extended hamming code		
	n	K	Public key size	n	k	Public key size
2	3	1	1	4	1	1
3	7	4	4	8	4	4
4	15	11	21	16	11	22
5	31	26	101	32	26	104
6	63	57	449	64	57	456
7	127	120	1905	128	120	1920
8	255	247	7874	256	247	7940
9	511	502	32066	512	502	32128
10	1023	1013	129538	1024	1013	129664

When n = 15 or 16 and k = 11, the number of probable secret keys is large and the time needed to break the system is very long, roughly 1.271e+38 years. This case has suitable public key size.

When the value of n = 31 or 32 and k = 26, the McEliece public key cryptosystem is very secure and any try to break it needs lots of years. Even the public key size is increased but still not large in comparison with the large capacity of storage area and high transmission speed.

When m≥6, the McEliece public key cryptosystem is very secure and any try to break it needs very large number of years. But the public key size becomes large and needs large space in the memory, especially when m = 10.

From this previous study it was found that when m=2 and 3 the public key size is very small but the cryptosystem is not secure. When m = 4 and 5, in this case the cryptosystem is secure also the public key size is not large comparing with the large capacity of the storage media and with high transmission speed. When m≥6 the number of probable keys is very large and the system is very secure but the public key size was increased, and will cause implementation problems especially when m = 10. This large public key size makes this cryptosystem not widely used because it needs large space in the memory also needs very high transmission speed. But changes in technology and economies, for example the plummeting cost of storage, keep it on the list of candidates for some applications.

2- Message Expansion

Table (4.8) offers message expansion of the cryptosystem, this table presents that the encrypted message was longer than the original message by (n-k) bits. System message expansion of hamming code was less than of extended hamming code. So the encrypted message is much longer than the plaintext. This expansion of the message is considered as a drawback of this cryptosystem because it makes the system more prone to transmission error.

Table (4.8): Message expansion system when using hamming and extended one.

m	Hamming code			Extended hamming code		
	n	K	Message expansion	n	k	Message expansion
2	3	1	3	4	1	4
3	7	4	1.75	8	4	2
4	15	11	1.364	16	11	1.455
5	31	26	1.192	32	26	1.230
6	63	57	1.105	64	57	1.123
7	127	120	1.058	128	120	1.067
8	255	247	1.032	256	247	1.036
9	511	502	1.018	512	502	1.01992
10	1023	1013	1.009	1024	1013	1.010859

3- Information Rate

Table (4.9): Information rate system when using hamming and extended hamming code.

m	Hamming code			Extended hamming code		
	n	K	Information rate	n	k	Information rate
2	3	1	0.333	4	1	0.25
3	7	4	0.571	8	4	0.5
4	15	11	0.733	16	11	0.6875
5	31	26	0.839	32	26	0.8125
6	63	57	0.905	64	57	0.890625
7	127	120	0.945	128	120	0.9375
8	255	247	0.969	256	247	0.965
9	511	502	0.982	512	502	0.980
10	1023	1013	0.990	1024	1013	0.989

Table (4.9) presents information rate for different values of n and k of hamming code and extended hamming code. When m is small (m =2 and 3) the information rate is low but it would be increased when the value of m was increased as shown in table (4.9). And system information rate of extended hamming code was lower than of hamming code.

4- Execution Time

Table (4.10) and (4.11) present execution time for encryption and decryption stages of the cryptosystem. These tables present that the encryption and decryption execution time of extended hamming code needs more little time than the time was needed to execute with hamming code. However, encryption and decryption has relatively little time. So, this faster execution time is considered as an advantage of the cryptosystem.

Table (4.10): System encryption and decryption execution time of hamming code.

m	n	K	Encryption time$_{(second)}$	Decryption time$_{(second)}$
2	3	1	0.00192	0.00198
3	7	4	0.00103	0.00190
4	15	11	0.00109	0.00282
5	31	26	0.00167	0.00510
6	63	57	0.003	0.00971
7	127	120	0.00603	0.0204
8	255	247	0.0130	0.0510
9	511	502	0.0311	0.142
10	1023	1013	0.0834	0.322

Table (4.11): System encryption and decryption execution time of the extended hamming code.

m	N	k	Encryption time$_{(second)}$	Decryption time$_{(second)}$
2	4	1	0.00223	0.00296
3	8	4	0.00115	0.00234
4	16	11	0.00117	0.00312
5	32	26	0.00171	0.00531
6	64	57	0.00304	0.0101
7	128	120	0.00606	0.0206
8	256	247	0.0131	0.0514
9	512	502	0.0312	0.143
10	1024	1013	0.0835	0.323

The time of extended hamming code was larger than the hamming one, but the difference was small. As a normal suggestion, the extended one was preferred, since it can correct one error and can detect two. Whereas the cryptosystem of hamming code can correct one error and cannot detect two errors.

Chapter Five

Conclusions and Future Work

5.1 Conclusions

1. The McEliece public key cryptosystem is not secure when its parameter value (m=2 and 3), because the number of McEliece secret keys are very small, and an adopted mechanism was used to make this cryptosystem secure when m=3. And when the value of the cryptosystem parameter (m=4) the number of cryptosystem secret keys become large and the time needed to break the system is long, so the system is secure. When ($m \geq 5$) the number of keys becomes very large and needs very long time to be discovered in order to decrypt a message, so the McEliece cryptosystem is very secure in this case.
2. The public key size of the McEliece cryptosystem is small when the value of the cryptosystem parameter (m=2 and 3) and the system is not secure. When (m=4 and 5), the system become is secure at the same time the storage area that was needed for the public key is not large. While when ($m \geq 6$) the cryptosystem is very secure but the size of the public keys become large, and this will cause implementation problems.
3. An adopted mechanism was designed and implemented to overcome the weak points of this cryptosystem and to enhance the performance of it (especially when m = 3, when m = 4 the system is secure, to make it more secure this mechanism can be used).

4. McEliece cryptosystem has low information rate when the value of cryptosystem parameter (m) is small (m=2 and 3), but this information rate would be increased when m is increased.
5. Encryption and decryption has relatively little time in the McEliece system. Therefore, the on going study of this system is vital to the future of cryptography and this cryptosystem may provide an alternative to the current public key cryptosystem. So, this high-speed encryption and decryption is considered as an advantage of the cryptosystem.

5.2 Future Work

1. Using another type of linear block codes instead of hamming code and extended hamming code such as Golay code, Reed Muller code or Goppa code and evaluates the security of McEliece cryptosystem and trying to enhance it when using one of these codes. These codes can detect and corrects multiple errors depending on its propriety.
2. Another type of attacks could be used instead of brute force attack to evaluate the security of McEliece public key cryptosystem when using one of these codes (hamming code, extended hamming code, Golay code, Reed Muller code or Goppa code).
3. Trying to overcome the drawbacks of this cryptosystem (McEliece cryptosystem): large public key, message expansion and low information rate.

References

[Ber97] Berson A. Thomas, ”Failure of the McEliece Public-Key cryptosystem Under Message-Resend and Related-Message Attack”, B. Kaliski (Ed.), Advances in Cryptology- Proceedings of Crypto '97, Lecture Notes in Computer Science, Springer Verlag, IEEE, 1997, [www.math.utah.edu/ftp/pub/tex/bib/idx/ lncs1997a/1294/0/213-z.html].

[Buc01] Buchmann A. J., “Introduction To Cryptography”, Springer-Verlag New Work, Inc, USA, 2001.

[Car05] Carlsson A., “INFORMATION THEORY”, 2005, [http://www.it.isy.liu.se/research/indexeng.html].

[Cha98] Chakravarti I. M., “Design Theory and Coding Theory”, 1998, [www.stat.unc.edu/321F.html].

[Che00] Cherowi W., “The McEliece Cryptosystem”, 2000, [http://www-math.cudenver.edu/~wcherowi/courses/m5410/ ctcmcel.html].

[Cla97] Clayton R., ”Brute force attack”, 1997, [http://en.wikipedia.org/wiki/ Brute_force_attack].

[Das86] Das J., Mullick S.K., Chatterjee P.K., ”principles of digital communication”, department of electrical engineering India institute of technology, Kanpur, India, John Wiely & sons, copyright 1986, Wiely Eastern limited new Delhi, 1st edition.

[Fre00] Frey B. J., “Error-Correcting Codes”, IS250 Computer-Based Communications Systems and Networks, Spring 2000, [www.ischool.berkeley.edu/~rosario/projects/error_correcting_codes.html].

[Han06] Hankerson A., “Algebraic Codes”, Sections: Hamming codes, extended codes, 12-Jun-2006.

[Hay01] Haykin S., ”Communication Systems”, 4th edition, 2001.

[Heu98] Heumann S., “Redundancy, Surprise, and Information Rate”, 12/4/1998, [www.mdstud.chalmers.se/~md7sharo/coding/main/node16.html].

[Hhl00] Hankerson D.R., Hoffman D.G., Leonard D.A., Lindner C.C., Phelps K.T., Rodger C.A., Wall J.R, "CODING THEORY AND CRYPTOGRAPHY”, Second Edition, 2000.

[How00] Howe D., **“**Breaking ciphers with many workstations”, 2000, [http://www.distributed.net/projects.html.en].

[Jam02] James Gannon, “Cryptography”, 2002, [http://en2.wikipedia.org/wiki/cryptography].

[Jar98] Jargon B., “Brute-force attack”, 1998, Security definitions.com.

[Jer03] Jeroen M., “Some Applications of Coding Theory in Cryptography“, Technische Universiteit Eindhoven, 1st edition, 2003.

[Joh98] Johansson T., "On the Complexity of Some Cryptographic Problems Based on the General Decoding Problem", Dept. of Information Technology, Lund University, Box 118 S-221 00 Lund, Sweden, 1998, [www.it.lth.se/projects/cryptology/e-papers/paper054.pdf].

[Jon06] Jones J. Antonia, ”DATA SECURITY”, 25 February 2006, [http://www.cs.cf.ac.uk/user/Antonia.J.Jones/lectures].

[Kor05] Korner T. W., ” Coding and Cryptography”, February 8, 2005, [http://www.dpmms.cam.ac.uk].

[Lee98] Lee P. J. and Brickell E. F., ”On observation on the security of McEliece’s public key cryptosystem”, Bell Communication Research, U.S.A., 1998.

[Leh93] Lehre S., “An introduction to Error-Correcting codes “, Georg-August-Universe, 1993, [http://www.num.math.uni-goettingen.de/Lehre/Lehrmaterial /texte/abthesis.html].

[Lou00] Loureiro S., ”Function Hiding Based on Error Correcting Codes”, Institut Eurecom, Sophia Antipolis, France, 2000, [www.eurecom.fr/~nsteam/Papers/cryptec99.pdf].

[Men96] Menezes A., P. Van and S. Vaston, “Handbook of Applied Cryptography”, CRC Press, 1996.

[Mey98] Meyer P., ” An Introduction to the Use of Encryption”, 1998, [www.hermetic.ch/crypto/intro.htm].

[Mic85] Michelson Arnold M.,” Error Control Techniques for Digital Communication”, published in Canada, 1st edition, 1985.

[Mos01] Moschoyiannis S., ”Group Theory and Error Detecting/ Correcting Codes”, M.Sc Thesis, University of Surrey, Department of Computing, School of Electronics Computing and Mathematics, September 2001, [www.cs.bham. ac.uk/~sxm/

GroupTheory&ErrorDetectingCorrectingCodes.pdf].

[Pet72] Peterson W., " ERROR-CORRECTING CODES", second edition, 1972.

[Pfl89] Pfleeger C., "Security in Computing", Prentice-Hall, Inc., USA, 1989.

[Pre92] Preneel B., Bosselaers A., Govaerts R., Vandewalle J., "A software implementation of the McEliec public key cryptosystem", 1992, [www.cosic.esat.kuleuven.be/publications /article- 267.pdf].

[Ric96] Richard Cam, Ph.D., "CONSTRUCTION OF SIMPLE AND PROVABLY ROBUST 4LZS CONTROL CODES", 31 Oct. 1996, [grouper.ieee.org/ groups/public/presentations/ nov1996/ BHgutp_e.pdf].

[Sch97] Schneier B., "Applied Cryptography", John Wiley and Sons Inc, second edition, 1997.

[Sha85] SHANMUGAM K. SAM, "digital and analog communication system", University of Kansas, John Wiley & Sons, copyright 1979, 1985 by John Wiley & sons New York, 1st edition, Published simultaneously in Canada.

[Smi02] Smith B., "Algebraic Coding Theory", April 22, 2002.

[Spe00] Spence S. A., " Introduction to Algebraic Coding Theory", Supplementary material for Math 336 Cornell University, 2000. [www.math.cornell.edu/~kbrown/336/coding.pdf].

[Sta03] Stallings W., “Cryptography and network security principle and practice”, Pearson educational international, 2003.

[Ste99] Steve Matuszek, “Encryption”, 1999, [http://www.cs.umbc.edu/ ~wyvern/ta/ encryption .html].

[Sua03] Suanne Au, Christina E. and Everson J.,” The McEliece Cryptosystem”, September 17, 2003, [www.math.unl.edu/~s-jeverso2/McElieceProject.pdf].

[Sun00] SUN HUNG-MIN, “Enhancing the Security of the McEliece Public-Key Cryptosystem”, Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 16, 799-812 (2000), [www.math.utah.edu/pub/tex/bib/cryptography2000.bib].

[You02] Yousif A. M., “Key Diskette Software Copy Protection System”, Ph.D. thesis, the University of Technology, September 2002.

الخلاصة

يعتبر موضوع الترميز وفك ترميز البيانات المرسلة عبر قنوات الاتصال من المواضيع التي لاقت اهتماماً كبيراً. من المعروف ان هناك العديد من الانواع لترميز تصحيح الخطأ: من هذه الانواع ترميز الـ hamming وترميز الـ extended hamming. ترميز الـ hamming له قابيلة اكتشاف وتصحيح خطأ واحد في حين ترميز الـ extended hamming له قابيلة اكتشاف خطأين بالاضافة الى اكتشاف وتصحيح خطأ واحد.

هذا العمل يهتم بتنفيذ نظام التشفير(McEliece) عندما يستخدم ترميز الـ hamming وترميز الـ extended hamming، تقييم الامنية لهذا النظام و تحسين اداء هذا النظام. نظام التشفير (McEliece) وهو نظام تشفير المفتاح المعلن الذي يعتمد على نظرية الترميز الجبري.

تصميم و تنفيذ النظام يتألف من اربعة مراحل. المرحلة الاولى هي مرحلة توليد المفتاح لنظام التشفير(McEliece) باستخدام ترميز الـ hamming وترميز الـ hamming extended. هناك ثلاثة مفاتيح سرية تشترك في توليد هذا المفتاح (المصفوفة المولدة، المصفوفة التي لها معكوس ومصفوفة الابدال). المرحلة الثانية هي مرحلة التشفير، في هذه المرحلة الرسالة الاصلية تحول الى رسالة مشفرة. الرسالة الاصلية تشفر باستخدام المفتاح المعلن الذي تم توليده في المرحلة الاولى وبعد ذلك يتم اضافة خطأ الى هذه الرسالة. المرحلة الثالثة هي مرحلة فك الشفرة. المفاتيح السرية سوف تستخدم في هذه المرحلة لفك تشفير الرسالة المشفرة والحصول على الرسالة الاصلية. وكذلك في هذه المرحلة فان الرسالة المشفرة سوف تصحح من الخطأ الذي اضيف لها. المرحلة الاخيرة تتعلق بتقييم الامنية لنظام التشفير(McEliece). عملية التقييم تعتمد على استخدام نوع من انواع الهجوم المسمى (brute force)، و امنية هذا النظام يمكن ان تقاس من خلال مقاومته لهذا النوع من الهجوم. بعد استعمال هذا الهجوم بعض نقاط الضعف قد ظهرت و لوحظت. للتغلب على هذه النقاط و لجعل هذا النظام اكثر امناً قد اقترحت ونفذت تقنية مطورة لهذا الغرض. كذلك في هذه المرحلة انواع اخرى من متغيرات هذا النظام قد تم تقييمها وهي حجم المفتاح، توسع الرسالة و نسبة المعلومات. امنية، تنفيذ و استخدام هذا النظام يتأثربهذه المتغيرات.

كل البرامج المطلوبة لانجاز هذا المشروع او العمل قد نفذت باستعمال اللغة البرمجية (visual basic) الاصدار السادس التي تعمل في بيئة نظام التشغيل (Windows XP).

جمهورية العراق

وزارة التعليم العالي و البحث العلمي

جامعة النهرين

كلية العلوم

تقنية لتحسين أداء نظام التشفير (McEliece)

رسالة

مقدمة إلى كلية العلوم، جامعة النهرين كجزء من متطلبات نيل شهادة الماجستير في علوم الحاسوب

من قبل

نور رضا عبد الرزاق القزاز

(بكالوريوس ٢٠٠٣)

المشرفون

د. ستار بدر سدخان	د. جمال محمد كاظم

٢٠٠٦	١٤٢٧