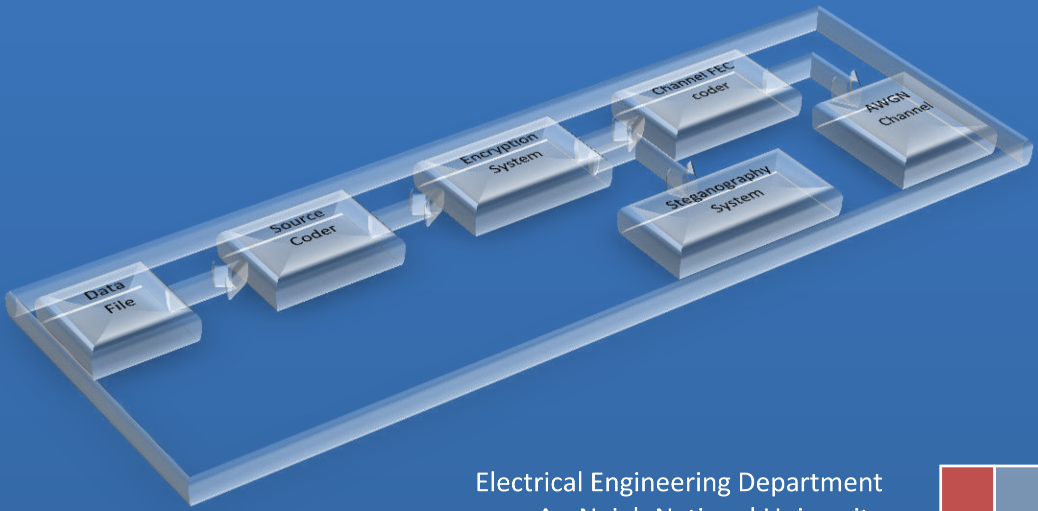
IMPLEMENTATION OF A DIGITAL COMMUNUCATION SYSTEM

MOHAMMAD TAHA HAMID **MURAD SODQI H. AYYASH**

Supervised by

Dr. ALLAM MOUSA



Electrical Engineering Department
An-Najah National University
2011

IMPLEMENTATION OF A DIGITAL COMMUNUCATION SYSTEM

MOHAMMAD TAHA HAMID

M.T.HAMID@HOTMAIL.COM

MURAD SODQI AYYASH

MURAD_AYYASH@HOTMAIL.COM

Supervised by

Dr. ALLAM MOUSA

A Thesis presented to the Electrical Engineering Department of An-Najah National University in the fulfillments of partial requirements of the B.Sc. degree in Electrical Engineering.

Department of Electrical Engineering

Faculty of Engineering

An-Najah National University

2011

ABSTRACT

The implementation of the digital communication system is important for the study, analysis, test and development of the performance of the system, in this project we introduce two parts to carry out the implementation, the theoretical part and it is concerned with the study and analysis of the algorithms in source coding, channel coding and data security represented in cryptography and steganography. the second part is the MATLAB implementations, simulations and analysis, the adopted algorithms for the implementation are Huffman, BWT compression, Run-Length coding and decoding, for lossless source coding, BCH coding and decoding, for forward error correction, advanced encryption standard AES with 128-bit cipher key with second protection layer of steganography for data security, the performance was simulated and tested under the AWGN channel with M-PSK digital modulation.

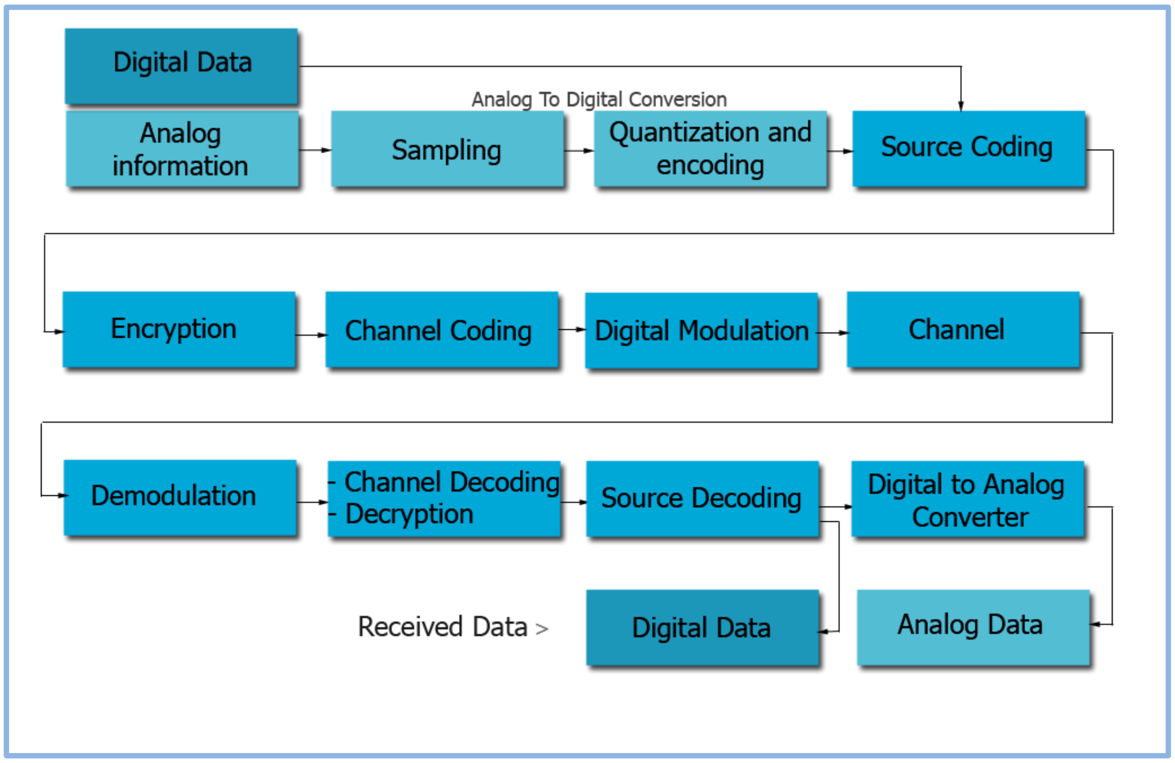
ACKNOWLEDGMENT

We wish to express our appreciation of the assistance given us by the Dr.Allam Mousa, the supervisor of our project, Department of Electrical Engineering, An-Najah National University.

Table of Contents

Introduction

The project aims at the implementation of the following digital communication system starting from the data processing with digital input data. The figure below shows a block diagram of a digital communication system with two possibilities for the input data that may be an analog or digital data**.**



Block diagram of Digital communication system

The brief overview for the system parts and its parts is described as follows:

Source coding

The data compression or source coding is the process of encoding information using fewer bits (or other information-bearing units) than an unencoded representation would use, by removing the redundancy and we can remove it until limit defined as entropy through use of specific encoding schemes. For example Huffman BWT. The Function of source coding is to reduce the size of information that will be transmitted to maintain the resources. BW , power and space data storage.
The first chapter introduces the BWT compassion which employs Huffman coding, MTF transforms ,BW transforms.

Channel coding

It is the addition of redundancy on the original input bit steam taking the channel capacity into consideration it is functions to ensure that the signal transmitted is recovered with very low probability of error at the destination and usually designed to make error-correction possible .
The BCH forward error correction algorithm is adopted and it is discussed in the second chapter

Encryption:

It is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. It is used to protect the information (privacy), authentication(verifying the message origin) and Integrity(establishing that a received message has not altered).Another layer maybe added to conceal the encrypted data which is the steganography .
The third chapter is a description of AES in order to implement Rijndael encryption algorithm .

The Channel

The channel is the media that the signal will propagate into to the receiving part of the system this channel introduces an error to the signal. it is limited by a specific bandwidth and consequently a specific capacity. The AWGN channel is adopted with digital M-PSK modulated data.

The last chapter is the MATLAB implementation of the system with simulations and analysis that are needed to measure the performance of the system.

1

Chapter One

SOURCE CODING

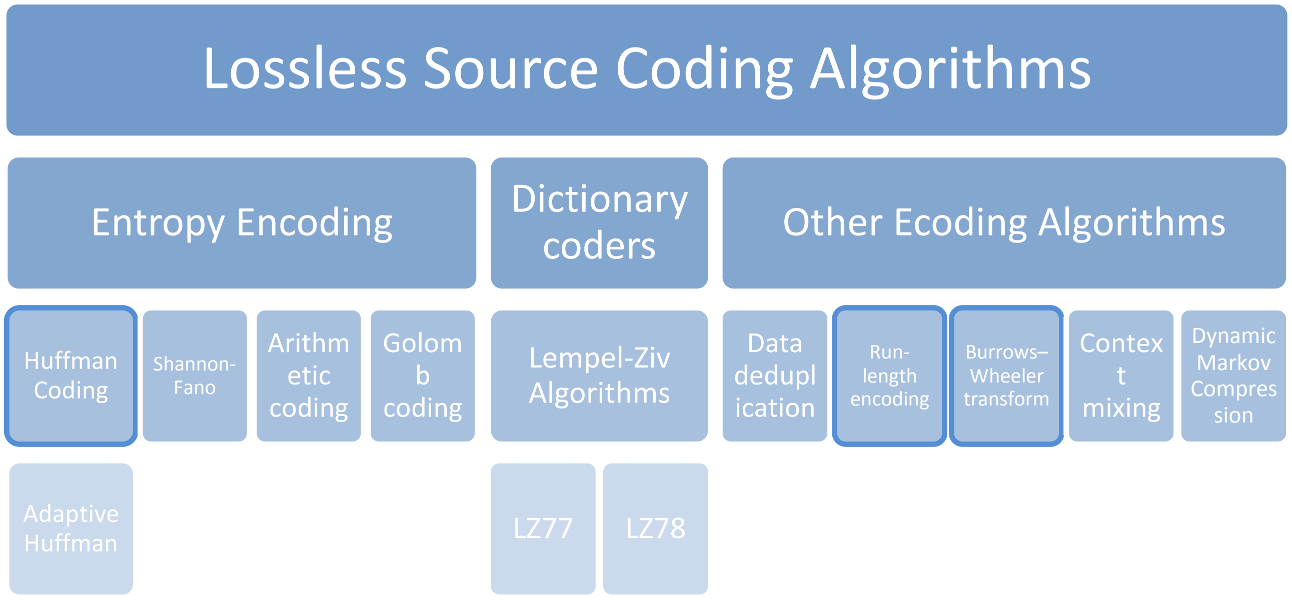
Implementation of Source Coding and Decoding

In this chapter analysis and implementation of Entropy coding such as Huffman coding , and other coding Techniques like Run Length Coding , MTF transform and BWT as an efficient text-coding technique

Source Coding

In this chapter the lossless source coding techniques are to be studied and analyzed in order to be implemented.

The lossless coding techniques is summarized in this chart



This chapter is a study, analysis and implementation of the lossless source coding techniques, the first algorithm is the Huffman Coding which is chosen since it has an acceptable code efficiency and performance for all types of files. The second is the Burrows–Wheeler transform which is more efficient compression coding for text files. The third is the run length coding.

Introduction

A code is a mapping of discrete of set of symbols $\{0, \dots\dots, M-1\}$ to finite binary sequences, *Data compression* or *Source coding* is the process of encoding information using fewer bits than an unecoded representation, by removing the redundancy and we can remove it until limit defined as entropy.

We have two kinds of source coding ,lossless source coding by which the data can be decoded to form exactly same bits can be achieved by moderate compression (e.g.: 2:1) this type is used in "ZIP" and important applications in medical images that's we can't accept any error in it. On the other hand we have lossy coding which is Method for representing discrete-space signals with minimum Distortion. Decompressed image is visually similar but has high compression ratio (ex: 20:1),this type is used in JPEG and MPEG .

According of the output of the coder we have Variable length coding that is compressed and decompressed with zero error (lossless data compression) and the each symbol representation of bits depends on the probability (ex: Huffman ,Shannon-Fano, …) and the second one is fixed length code which all symbol have the same codeword length independent of the probability(Lempel-Ziv).

A *Prefix Code* is a specific type of uniquely decodable code in which no code is a prefix of another code, as shown with a source $A = \{S0, S1, S3, S3\}$

Symbol	Codeword
S_0	0
S_1	10
S_2	110
S_3	111

Run length coding is very simple code that is orders the same data value occurs in many repeated data elements are stored as a single data value and count, rather than as the original run

For example if we have data like this (0000000111111111000000) more efficient to represent the data like this (07 19 06).

Efficiency parameter (η) for a code is very important to know which code is more efficient by this equation below

$$\mu = \frac{H}{L} * 100\%$$

Techniques we will cover in details in this section, Huffman coding, Shannon-Fano and BWT transform & compression.

Entropy and conditional Entropy

The basic definition of the Entropy is a measure of the source information depends on the probabilities of its data symbols .

The information of a single event or message ***' I '*** that any data contains is defined as the base 2 logarithm of its probability **'p'** as follows

$$I = \log_2(P)$$

And the entropy 'H' is the negative of the information

$$H = -I$$

For a data of many symbols its entropy is defined as the average entropy of all elements

$$H = -\sum_i P_i \,.\, \log_2(P_i) = \sum_i P_i \,.\, \log_2\left(\frac{1}{P_i}\right)$$

For a two random variables that take two probabilities of P and 1-P the entropy is as the following :

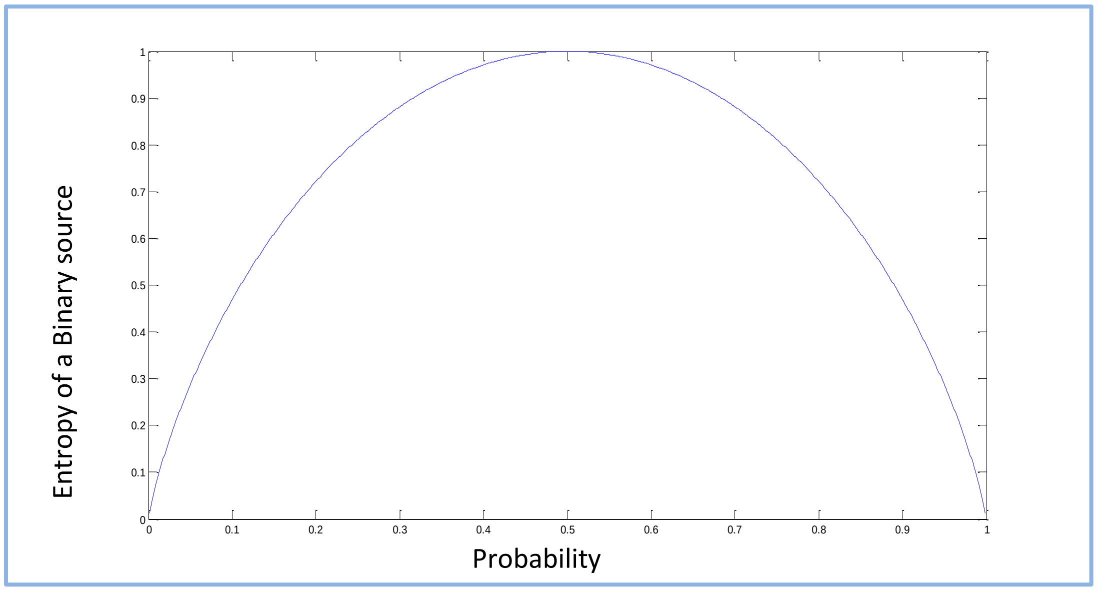


Figure 1: P vs. H for Two Symbols –Data (Binary Source Of Information)

Conditional Entropy and Mutual Information:

If X and Y are two variables, and if x and y are correlated, their *Mutual Information* is The average information that Y gives about X .

And their entropy is a conditional entropy and it is given as the entropy of X given Y :

$$H(x|y) = -\sum_{i} . \sum_{j} P(\mathrm{i},\mathrm{j})\ .\log_2 P(i|j)$$

The Mutual information between X and Y is

$$I(X, Y) = H(X) - H(X|Y)$$

This equation describes that the mutual information is the reduction in uncertainty of X given Y .

In order to implement source coding techniques, the following files are to be used as an input of the encoders , the files are analyzed by measuring its size before the compression and its entropy in bits per symbol and then the output file size , the results are for samples of :

Text Files

#	size before bytes(txt)	size after bytes	entropy
1	1.14 k	588	4.1486
2	30.5k	13.14k	4.179
3	56.9k	24.28k	4.155
4	79.6k	34.13k	4.157
5	183k	78.8k	4.1626

Images TIF(Tagged Image File)

#	Size Before Bytes	Size After Bytes	Entropy
1	2.25M	1.96M	6.555
2	2.26M	2.187M	7.74
3	28.8M	27.515M	7.85
4	264K	230.2K	7.57
5	510K	413.66K	6.63
6	916K	876.8K	7.77
7	1.38M	1.285M	7.55

Speech Files

#	Speech bytes	Speech Entropy Bit/symbol
1	286K	13.0752
2	160.6K	13.1137
3	59K	13.24
4	78.7K	14.11
5	52K	13.17
6	61K	12.8
7	23K	12.56

Images JPG

#	Image size Mb	Speech Entropy Bit/symbol
1	4.82	7.9177
2	3.41	7.7087
3	0.091	7.6722
4	0.838	7.311

It appears that the file size and its entropy are independent since the entropy depends on the probability of the source symbols , but it is different between the different types of files .

The entropy will affect the size at the output of the encoder and it will be different for each coding technique.

Lossless Coding Techniques

The Lossless Data Compression technique recommended preserves the source data accuracy by removing redundancy from the application source data. In the decompression processes the original source data is reconstructed from the compressed data by restoring the removed redundancy ,The reconstructed data is an exact of the original source data . The motivation for using compression is to Save storage space or bandwidth.

The performance of the data compression algorithm is independent of where it is applied , it may be necessary to rearrange the data into appropriate sequence before applying the data compression algorithm. The purpose of rearranging data is to improve the compression ratio.

After compression has been performed, the resulting variable-length data structure is then Packetized(compressed file). The packets containing compressed data should be transported through a channel communication link from the source to the receiver . The contents of the packets are then extracted and data about algorithm are provided to the receiver in order to be decompressed.

One of the Parameters that are needed to check for any algorithm , **compression ratio** is the ratio between the size of the compressed file and the size of the source file.

$$compression\ Ratio = \frac{\text{size after compression}}{\text{size before compression}}$$

Compression factor is the inverse of the compression ratio. That is the ratio between the size of the source file and the size of the compressed file.

$$compression\ factor = \frac{\text{size before compression}}{\text{size after compression}}$$

the most important one is **code efficiency**

$$\mu = \frac{H}{L} * 100\%$$

There are a few well known Lossless compression techniques, including Huffman coding, arithmetic coding and Lempel-Ziv coding.

Entropy Coding

Shannon-Fano Coding:

Overview

Shannon-Fano coding is prefix codes which produces variable size codes for the symbols occurring with different probabilities. The coding depends on the probability of occurrence of the symbol and the general idea is to assign shorter codes for symbols that occur more frequently and long codes for the symbols occurring less frequently.so the probabilities must be known This makes the algorithm inefficient

Shannon-Fano algorithm:

The algorithm used for generating Shannon-Fano codes is as follows:

1) For a given list of symbols, develop a corresponding list of probabilities so that each symbol's relative probability is known.
2) List the symbols in the order of decreasing probability.
3) Divide the symbols into two groups so that each group has equal probability.
4) Assign a value 0 to first group and a value 1 to second group.
5) Repeat steps 3 and 4, each time partitioning the sets with nearly equal probabilities as possible until further partitioning is not possible.

Huffman Coding

Overview

Huffman code is the coding technique that takes a fixed length code and convert it to a variable length code this coding method assigns shorter codeword to the high probability symbols and loner codes for lower probability symbols . This method builds a tree to represent the optimal unique prefix code of each symbol, The actual compression is then performed by simply applying the translation given by the prefix code tree , every message encoded by a prefix free code (Huffman code) is uniquely decodable , this coding method is used in JPEG and MP3 files with other lossy compression techniques .

In this section we will implement this code by MATLAB and apply this simulation on different types of files (text , speech and image) and find the relationship between this code and entropy, and measure the code efficiency to compare it with another codes.

Huffman Coding Algorithm

This algorithm is divided into two main steps the first one is the preparation of the probability vector and the second is the binary code assignment for each symbol.

First Step :

Initialization of a list of probabilities with the probability of each symbol .

finding the list of probabilities for two smallest probabilities $P[i]$ and $P[i-1]$.

Addition the two smallest probabilities to form a new probability $P = P[i] + P[i-1]$.

Remove $P[i]$ and $P[i]$ from the list.

Adding P to the list.

The steps will be repeated until the list only contains 1 entry of probability equals one.

After that , the following Tree is the result for a source of information has the following probability of each symbols as shown in the figure :

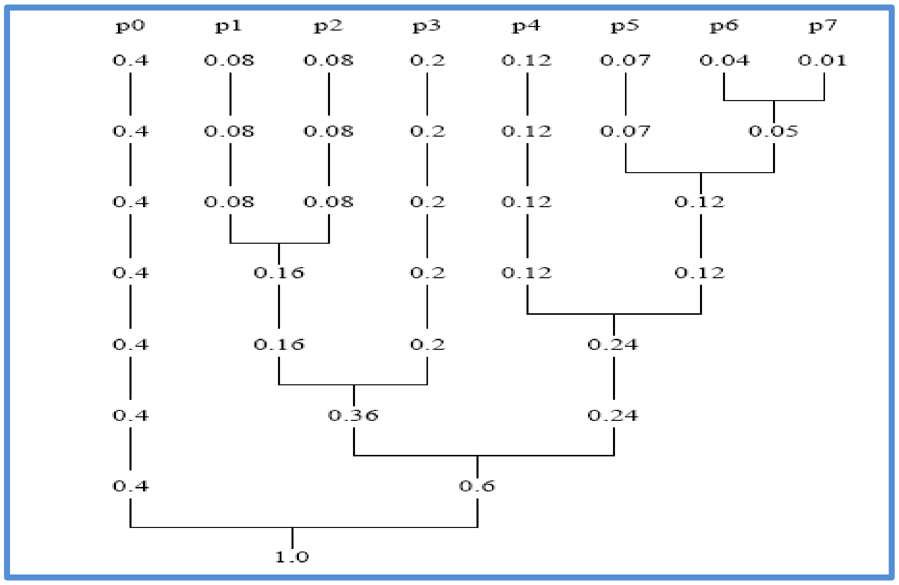


Figure 2 The Tree of the first step for 8 symbol-source

As shown all data words have to be inserted in the first row, starting with the highest one and then inserted into a tree . The tree will be computed in the following manner : The lowest two probabilities are added to one value Then the new value and all remaining values are copied into a new ***row*** The new row represents a new step and it is taken according to the its probability i.e the after taking the lowest two probabilities, the next two higher probabilities thant the first is taken and this will be continued until only one value (1) is obtained (the root of the tree)

The next step is the assignment of a binary code for each symbol by giving the binary value for the lower branch of each node of the tree and for the higher , it will be as in the figure 3

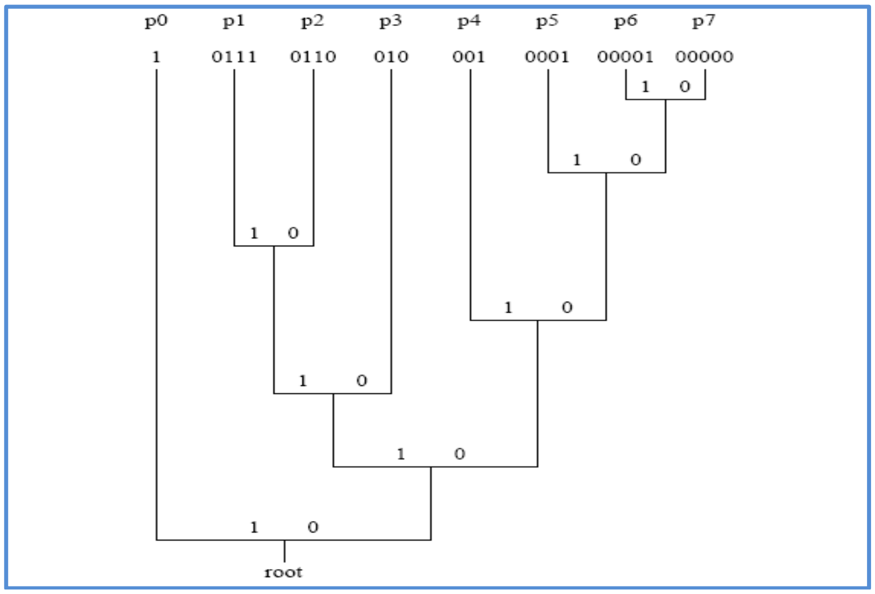


Figure 3 Binary code assignment for the symbols

As it shown , for every time the value is added, the higher value is assigned with a 0 and the lower with a 1, although taking *previous assignments* with it. As a result, the binary code is written in such way that appears in the figure 3 , each symbol with corresponding Huffman code.

Then the resulting code will be the minimum for higher probability and longer code for the lower probabilities according to the nodes of the tree.

Upper Bound for the Huffman Code Bit Rate

And Coding in Blocks

The bit rate of the code is related to the average codewords length L of the output code of Huffman coding , the following inequality is a theorem that shows that the average codeword length is restricted between the entropy of the source $H(X)$ and $H(X)+1$ as following .

assume that the source $A = \{X_0, \ldots., X_n\}$

$$H\left(X_n\right) \leq L < H\left(X_n\right) + 1$$

Coding can be done as a blocks of symbols to achieve a bit rate that approaches the entropy of the source symbols as the following

$$\underbrace{X_0, X_1, \ldots\ldots, X_{m-1}}_{Y_0} \quad \underbrace{X_m, \ldots\ldots\ldots X_{2m-1}}_{Y_1}$$

And Yn refers to the total blocks of symbols

It is obvious that

$$H\left(Y_n\right) = mH\left(X_n\right)$$

According to the firstly described inequality we have

$$H\left(Y_n\right) \leq L_y < H\left(Y_n\right) + 1$$

Dividing by the number of symbols m in the block

$$\frac{H(Y_n)}{m} \le \frac{L_y}{m} < \frac{H(Y_n)}{m} + \frac{1}{m}$$

$$H(X_n) \le L_x < H(X_n) + \frac{1}{m}$$

taking the limit as the block length approaches infinity i.e the size of the block is increasing with more symbols and codeword length

$$\lim_{m\to\infty} H(X_n) \le \lim_{m\to\infty} L_x < \lim_{m\to\infty} H(X_n) + \lim_{m\to\infty} \frac{1}{m}$$

The final equation results in

$$\lim_{m\to\infty} L_x = H(X_n)$$

Which means that as the block length is increasing the average length of the code (the bit rate) approaches the entropy of the source .
The result of increasing the block size is the efficiency reaches to 100% of the code , and for Huffman coding this performance could be achieved with large size of blokes.

Huffman encoder is implemented using MATLAB and it is applied for different digital data of different file types.

In order to obtain the relations between the different encoder parameters the simulation results are shown in the tables below.

Text Files using Huffman coding[*]

#	size before bytes	size after bytes	Compression factor	compression ratio	code efficiency	entropy
1	1.14 k	588	1.93	43.8%	0.99213	4.1486
2	30.5k	13.14k	2.32	43.1%	0.9913	4.179
3	56.9k	24.28k	2.34	42.6%	0.99128	4.155
4	79.6k	34.13k	2.33	42.86%	0.99123	4.157
5	183k	78.8k	2.32	43%	0.99205	4.1626

[*] **All compression results in this chapter are not practical results due to excluding the Header Files, the results taking from encoding and decoding at the same block without separation, practical results are shown in chapter4.**

The text files have an entropy around 4 bits per symbol so the predicted compression factor is to be around the 2.

Justification: ASCII characters is represented by 8 bits per symbol in the uncompressed file, after the compression it is represented in average by 4.16 bits per symbol (character)

Huffman Coding for a text data shows an acceptable compression performance since files sizes are approximately reduced to the half of the original ones.

The highest compression level is marked in red

Images TIF

#	Size Before Bytes	Size After Bytes	Compression Factor	Average Code Length	Code Efficiency	Entropy
1	2.25M	1.96M	1.147	6.64	0.986	6.555
2	2.26M	2.187M	1.0333	7.77	0.9961	7.74
3	28.8M	27.515M	1.0467	7.886	0.9959	7.85
4	264K	230.2K	1.1468	7.612	0.994	7.57
5	510K	413.66K	1.232	6.664	0.9957	6.63
6	916K	876.8K	1.0447	7.79	0.9965	7.77
7	1.38M	1.285M	1.073	7.56	0.997	7.55

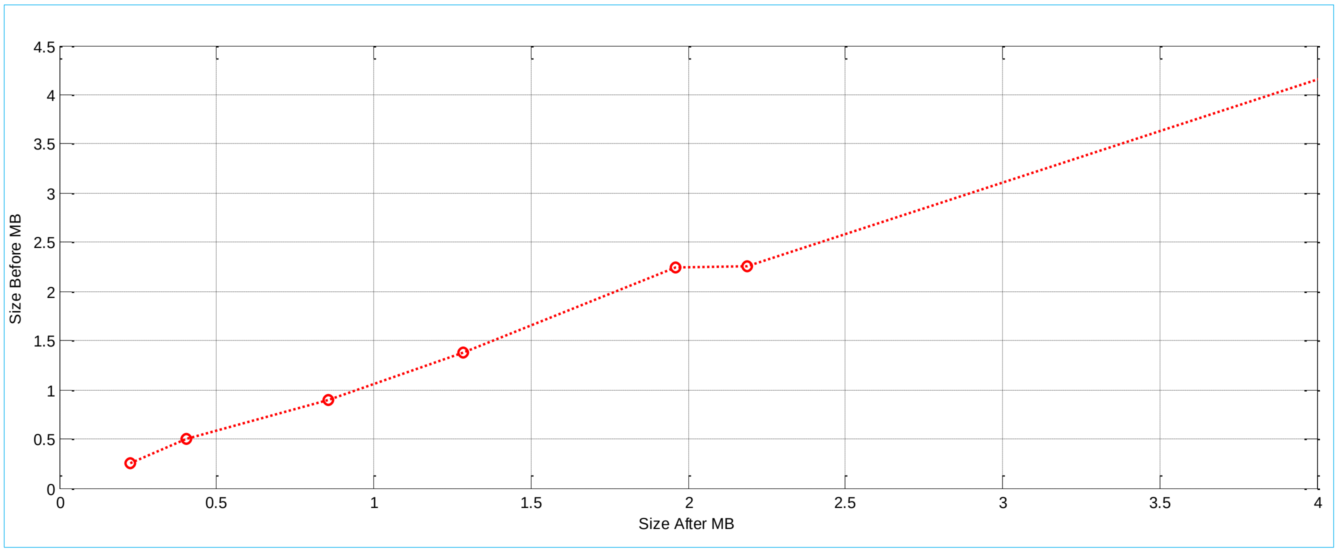


Figure 4 size before vs. size after Huffman Coding for TIF images

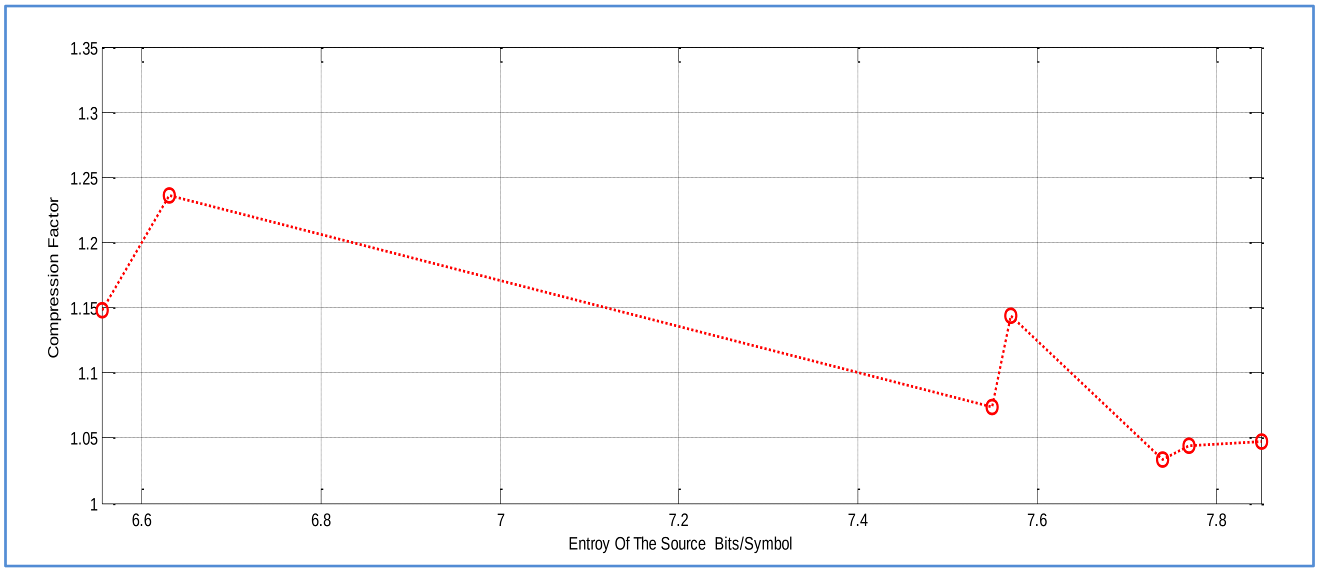


Figure 5 compression factor vs. entropy shows the effect of entropy on CR for ***different-size files***

It is obvious from the figure that the low entropy TIF files around 6.6 has the highest CR [**file 1** and **file 5** *'refer to the table above'*] but for file 5 of 6.63 entropy the code efficiency is greater than file 1 so it has a better CR than file 1

Huffman encoding shows adequate levels of compression for TIF image files .

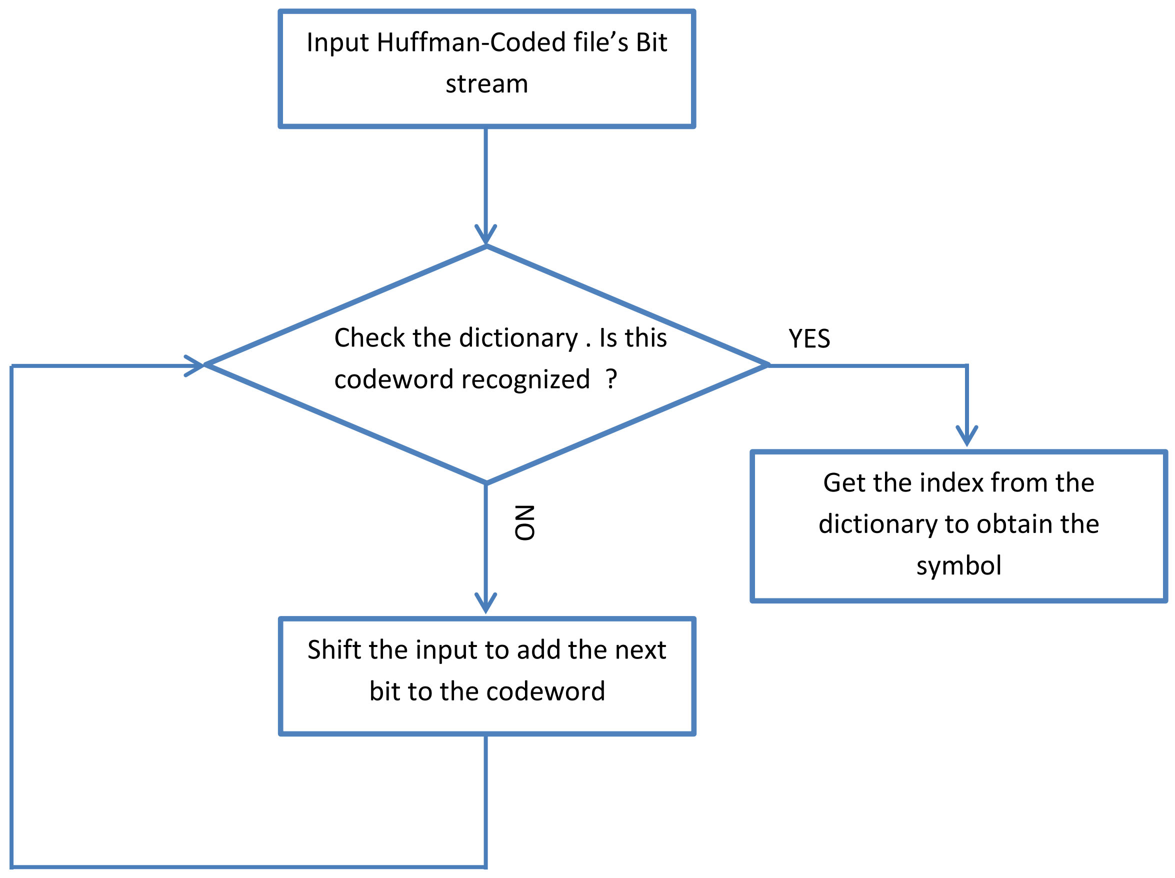
TIF Images have an entropy around 7 bits per symbol so the predicted compression factor is to be around the 1 .

Justification: TIF Image Symbols (pixels) is represented by 8 bits per symbol in the uncompressed file, after the compression it is represented in average by 7.38 bits per symbol ,as a result . Huffman encoder has a little levels of compression for this type of files .

In **figure 5** the relation between the entropy and the level of compression is inversely proportional , according to the justification above the first image has the highest level of compression due to the lowest entropy that it contains.

Huffman Decoding

For all files that are being decoded with Huffman algorithm there is decoding algorithm for this code and for all prefix codes in general it depends on the Huffman dictionary that is the output of the Huffman encode which contains the symbol index and corresponding Huffman code , this decoding algorithm is straight forward algorithm described in the chart below :


Input Huffman-Coded file's Bit stream
Check the dictionary . Is this codeword recognized ?
YES
Get the index from the dictionary to obtain the symbol
NO
Shift the input to add the next bit to the codeword

The process continuous until the end of the stream.

The Burrows-Wheeler Transform "BWT"

An Efficient Text- Compression Algorithm

The Burrows-Wheeler transform, also called "block-sorting" does not process the input sequentially, but instead it processes *a block of text* as a **single unit**.

BWT after rotating the block of text it sorts the characters in a block of text according to a ***lexical ordering*** of their following context. This process can be understood in terms of sorting a matrix containing all cyclic rotations of the text.

BWT is the a general name is assigned to the presses of sequential transformations that are BWT ,MTF 'move-to-front transform' , and entropy encoder , the first two transforms is aiming at ***decreasing the entropy of the source*** .

BWT Algorithm Description

The compression algorithm takes the input text, which is treated as a string ***S*** of ***N*** characters which are selected from an ordered alphabet of ***X*** characters in general. In this chapter it is the English with all set of symbols which are included in the ASCII code.

The ***first*** *step* is to create an ***N*** *x* ***N*** matrix **M** by using the input string **S** as the first row and rotating (cyclic shifting) the string ***N****-1* times and each time adding the new string as a new row to the existing ones. See figure 6 a

The ***second*** *step* is to sort the Matrix ***M*** lexicographically by rows. **At least one** of the rows of the newly created ***M'*** contains the original string **S**. and its denoted or named - the index of the first such row- ***I***. which if necessary for the decoding process.

The third and ***last*** *step* is to take the last characters of each row (from top to bottom) and write it in a separate string **L**. **L** and the Index **I** are the outputs of this transformation.

The Figure below shows such a matrix, constructed for the input string "mohammad,murad ". Each row is one of the fourteen rotations of the input, and the rows have been sorted lexically. The first column (**F**) of this matrix contains the first characters of the contexts, and the last column (**L**) contains the *permuted characters* that form the output of the BWT. The index ***I*** in this example is 10 as shown in the next figure.

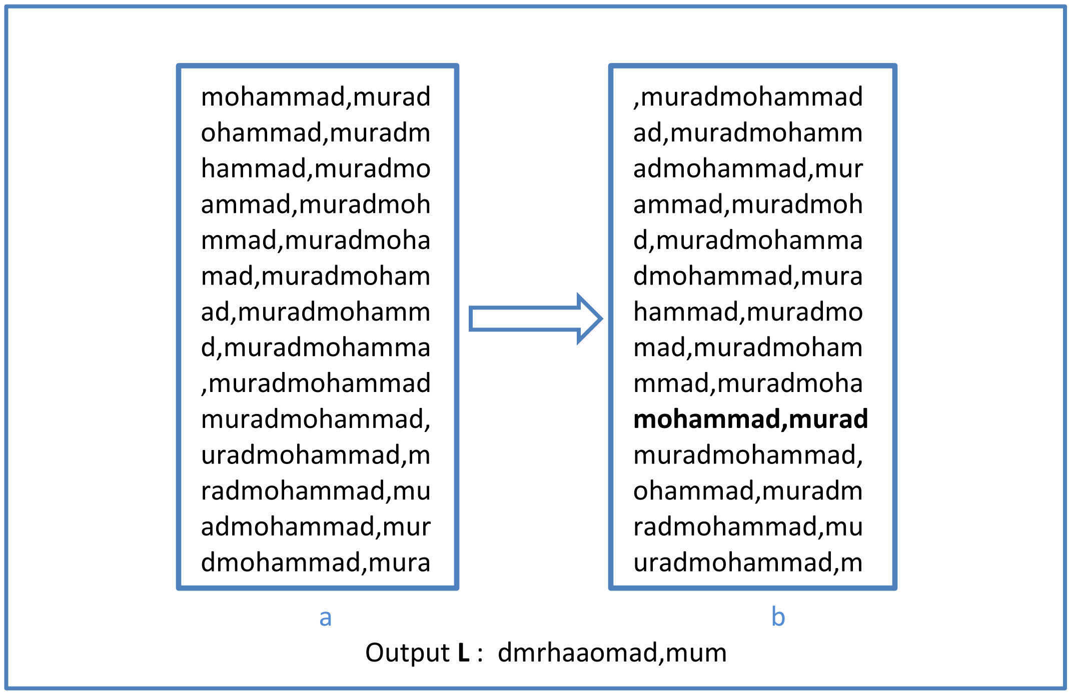


Figure 6 BWT of the input text mohammad,murad N=14 ,I=10 (a) not sorted cycles (b) lexical ordered rows

After that the output text is suitable to be encoded using ***Move-To-Front*** algorithm and then RLE and the entropy encoding as a final stage.
Figure 7 shows the standard BWT steps***, RLE is not Essential, and*** it is not included in this implementation.

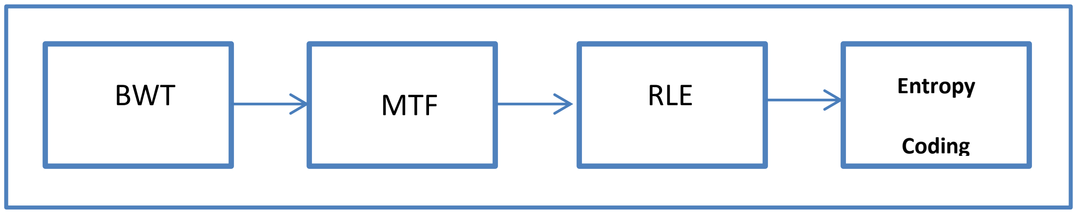


Figure 7 Typical scheme for the Burrows-Wheeler compression algorithm

Move-To-Front Transform

Move-to-Front encoding, is a scheme, as the name suggest, that uses a list of possible symbols and modifies it at every cycle (moving one symbol, the last one used). It uses the fact, that in many cases, the appearance of data words are clustered in short intervals. It uses self-organizing sequential search and variable-length encoding of integers. The advantages are that *allows fast encoding and decoding, and requires only* *one pass over the data to be compressed. It is used here as a part of BWT that is being discussed.*

Word	Position in dictionary	Dynamic dictionary of English alphabet
bananaaa	1	(abcdefghijklmnopqrstuvwxyz)
b**a**nanaaa	1,1	(bacdefghijklmnopqrstuvwxyz)
ba**n**anaaa	1,1,13	(abcdefghijklmnopqrstuvwxyz)
ban**a**naaa	1,1,13,1	(nabcdefghijklmopqrstuvwxyz)
bana**n**aaa	1,1,13,1,1	(anbcdefghijklmopqrstuvwxyz)
banan**a**aa	1,1,13,1,1,1	(nabcdefghijklmopqrstuvwxyz)
banana**a**a	1,1,13,1,1,1,0	(anbcdefghijklmopqrstuvwxyz)
bananaa**a**	1,1,13,1,1,1,0,0	(anbcdefghijklmopqrstuvwxyz)
Final code	**1,1,13,1,1,1,0,0**	(anbcdefghijklmopqrstuvwxyz)

Figure 8 MTF example for a word of repeated characters , bananaaa shown in steps

For the word mohammad,murad its MTF transformation is 4,5,5,4,3,0,1,5,5,3,7,7,5,5 for a dictionary of **its symbols.** Or the dictionary can be used as all printable ASCII characters [from decimal 32 to 126] , so no need to include it within compressed file .

Now , the BWT algorithm is implemented and applied to five text files , the results is compared with the Huffman coded files.

Text Files using BWT & Huffman Compression

BWT

#	size before bytes(.txt)	size after bytes	Entropy of the source text	Compression factor
1	1004	284	4.02	3.53
2	1.14k	474	4.18	2.4
3	2.29k	999	5.07	2.34
4	2.76k	0.989k	4.73	2.8
5	3.57k	1.33k	5.25	2.68

Huffman

#	size before bytes(.txt)	size after bytes	Entropy of the source text	Compression factor
1	1004	401	4.02	2.5
2	1.14k	513	4.18	2.27
3	2.29k	1.18k	5.07	1.94
4	2.76k	1.33k	4.73	2.07
5	3.57k	1.92k	5.25	1.85

It appears that BWT has **a greater** compression factors than that Huffman has, this can be explained due to the two transformations for the input text files BWT and MTF which changes the permutation (rearrangement) and shape of the source symbols, consequently these two transformation results in reduction of the source entropy before the last stage entropy coding as it shown in the table below

File No.	Entropy of the source text File input	Entropy of the source after BW&MTF transforms
1	4.02	3.412
2	4.18	3.881
3	5.07	4.15
4	4.73	3.477
5	5.25	3.614

Source files has average **4.63** bits per symbol while it is after BWT and MTF **3.706** bits per symbol in average , this causes the increase of CR's for these files by this technique. Now in this sample of text files BWT &MTF transform results in reducing files entropy in average with respect to original by **19.95%.**

The following figure shows the difference between Huffman and BWT compression for the tested text files :

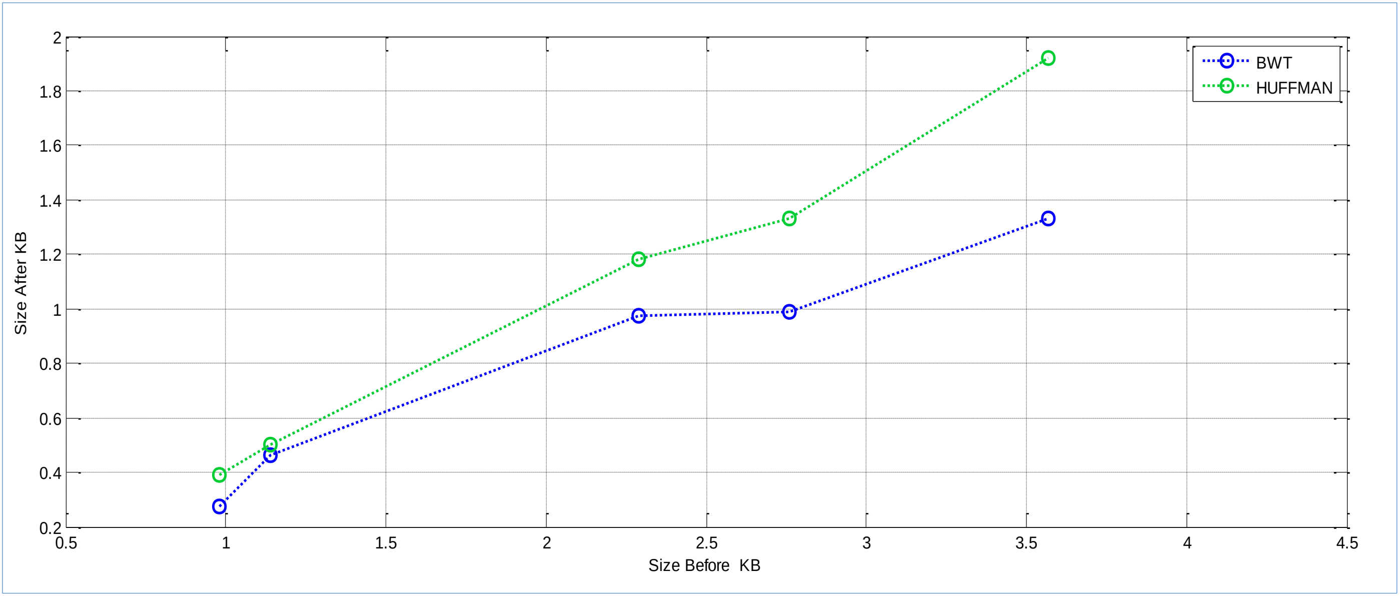


Figure 9 Comparison between ***BWT*** and ***Huffman*** coding for text files shows the size before against size after in K Bytes

In the last chapter more analysis is carried out with practical results and simulations.

The Decoding of Burrows-Wheeler Transform

The Burrows-Wheeler Transform has two methods of decoding, the first one is depending on the adding and sorting of the output text to have a matrix M of N x N size in such a way to reverse the coding process this method is easy and straight forward but it need more computer time to be done, the original string **S** is not obtained directly .

The second method is depending on the permutations it is more difficult than the first but it needs less commuting time and storage and it will be described.

This method ,unlike the first one , its output is the original string **S ,** at the beginning, we have the transformed string **L** and the index **I**.

L : d m r h a a o m a d , m u m

In the first step, the string ***L*** is sorted lexicographically and called ***F*** **:**

F : , a a a d d h m m m m o r u

It is similar to the first row of ***M'*** matrix .

Define the ***correspondence vector T*** of the length **N**, whose elements are numbers between 1 and N. The correspondence is defined as $F\ (\ T(\ k\ )\ ) =\ L\ (\ k\ )$ given that each ***k*** is unique integer and cannot be appear more than once starting from the first appearance .

in this example the correspondence vector ***T*** is like this ***:***

T = (5 ,8 ,13 ,7 ,2 ,3 ,12 ,9 , 4 ,6 ,1 ,10 ,14 ,11)

T(1)=5 states that L(1) has the index 5 at vector ***F ,*** or in other words "d" in the fifth position in F.

It is given that **in general** L(I) is the last character of **S** , in this example **I**=10 , L(10)="d" which is the last character of **S** = mohammad,murad

Now the text could be reconstructed from ***L*** and ***T*** with the following relations and steps ***"it will be appear that each step depends on the previous step while the first one depends on the index I ":***

Starting from the last symbol of ***S***

$$S\ (N)\ =\ L\ (\ X_0\ )$$
$$X_0\ =\ I$$

N: the number of elements of **S**

From i = 1 ... to N-1

$$S(N-i) = L(X_i)$$

And

$$X_i = T^i(X_i - 1)$$

To illustrate that, the previous example is taken;

First step

$$S(14) = L(10) = \text{“}d\text{”}$$
$$X_0 = 10$$

From **i = 1** to **14** ;

N=14

i=1

$$S(N-1) = S(13) = L(X_1)$$

$$X_i = T^i(X_i - 1)$$

$$X_1 = T^1(X_0) = T^1(10) = 6$$

So ,

$$S(13) = L(6) = \text{“a”}$$

i=2

$$S(N-2) = S(12) = L(X_2)$$

$$X_2 = T^2(X_1) = T^2(6) = 3$$

$$S(12) = L(3) = \text{“r”}$$

i=3

$$S(N-3) = S(11) = L(X_3)$$
$$X_3 = T^3(X_2) = T^3(3) = 13$$
$$S(11) = L(13) = \text{“u”}$$

i=4

$$S(N-4)=S(10)=L(T^{4}(13))=L(14)=\text{“ m ”}$$

.

.

And continue

.

.

.

i=13

$$S(N-13)=S(1)=L(X_{13})$$
$$X_{14}=T^{14}(X_{12})=T^{14}(5)=2$$
$$S(11)=L(2)=\text{“ m ”}$$

Then all ***S*** elements are obtained .

To summarize the relations that are used to decoding so far :

$$X_0 = I \qquad \text{.. (1)}$$

$$X_i = T_i\,(X_i - 1\,) \qquad \text{.. (2)}$$

Where: $i = 0, \dots, N-1$, $\boldsymbol{T}^{i}$ i*: represents index of T in the process of decoding*

$$S\,(N) = L\,(X_0\,) \qquad \text{.. (3)}$$

$$S(N-i) = L\,(X_i\,) \qquad \text{.. (4)}$$

Decoding of Move-To-Front Transform

In order to encode by MTF it is needed to construct a dictionary that is arranged according to the input symbols ,the arrangement is done by moving the symbol in the dictionary that is corresponding to the message symbol to the front of the dictionary .
Although the previous way is the encoding process it is also the same for decoding . to show that the example in figure is taken again

MTF code	Dictionary	Output
1,1,13,1,1,1,0,0	(abcdefghijklmnopqrstuvwxyz)	b
1,13,1,1,1,0,0	(bacdefghijklmnopqrstuvwxyz)	a
13,1,1,1,0,0	(abcdefghijklmnopqrstuvwxyz)	n
1,1,1,0,0	(nabcdefghijklmopqrstuvwxyz)	a
1,1,0,0	(anbcdefghijklmopqrstuvwxyz)	n
1,0,0	(nabcdefghijklmopqrstuvwxyz)	a
0,0	(anbcdefghijklmopqrstuvwxyz)	a
0	(anbcdefghijklmopqrstuvwxyz)	a

The output is bananaaa

Figure 9 MTF decoding for example of figure 8

Then the original message is obtained by one pass on the MTF code and arranging the dictionary which allows faster decoding than other techniques.

For the word mohammad,murad its MTF transformation is 4,5,5,4,3,0,1,5,5,3,7,7,5,5 and the decoding needs a dictionary of all possible ASCII symbols, this is done in the simulation part of source coding.

2

Chapter Two

CHANNEL CODING

Implementation of Binary BCH Linear Block Codes Encoder & Decoder

This chapter is a study and implementation of a BCH channel codes with important concepts such as the Joint Entropy and Conditional Entropy and its relationship with channel capacity and the capacity of two channel models BSC and AWGN channel.

Channel Coding Overview

Channel coding or error control coding is a part of a Digital communication system used to detect the errors in the transmitted data and correct it using redundant bits added to the original data according to a specific algorithm

This technique is divided into two parts the first is the forward error correction code FEC that used to detect the errors and correcting without the need of retransmission that is in the conditions of good BER and the second is Automatic repeat request ARQ which is used under huge errors to request a retransmission of data .

This figure shows the channel coding main parts :

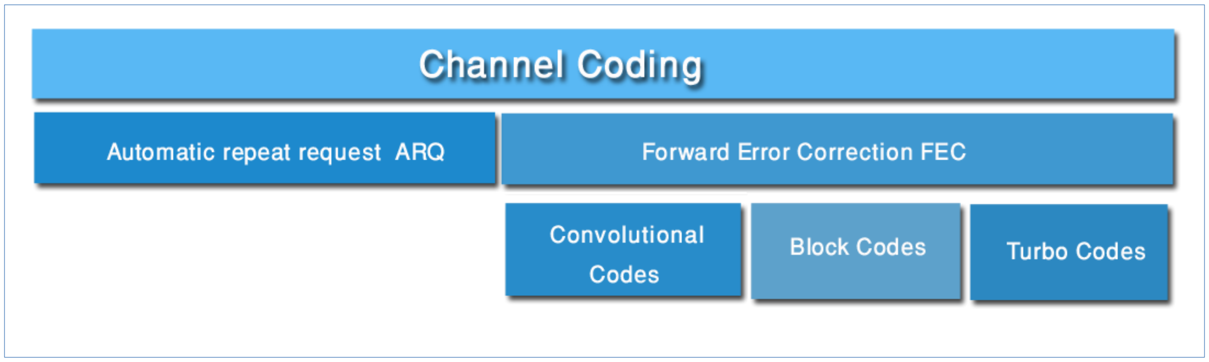


Figure 0.1 Channel Coding Classifications

Hamming Distance and Codeweight

- Two main measures for the coding performance are the hamming distance and code weight which are defined as: Hamming distance is number of places, bits in which they differ in two different codes but code weight is the number of one's in the code.

*The forward error correction coding is divided into main coding types which are the **Block Codes** , **convolutional codes** and **Turbo Codes***

Block Codes

The block codes is a channel coding technique that is divide the data which being transmitted into a fixed length blocks called Codewords *C* each of these Codewords is being coded according to an algorithm, if the output is containing the original data with parity bits the block code is called ***systematic*** but if the output is a new code without containing the original input data codeword explicitly then it is called ***unsystematic***.

For the block codes –which is the chapter topic, The error correction performance of a block code is described by the minimum Hamming distance d between each pair of code words, and is called the distance of the code.

The binary information sequence at the encoder input has a rate of $bits/sec$. Mainly there are two types of channel encoding techniques. The first is the block coding, by which a blocks of k information bits are encoded into corresponding bits blocks. Each n bits is called a code word with a total number of 2^k possible code words. The code rate, defined as the ratio $/n$, is a measure of the amount of the ***redundancy*** introduced by the specific block coding technique.

A block code C is constructed by breaking up the message data stream into blocks of length k has the form $\{m_0 , m_1 , \ldots , m_{k-1}\}$, and mapping these blocks into code words in C . The resulting code consists of a set of M code words $\{C_0 , C_1 , \ldots. , C_{M-1}\}$. Each code word has a fixed length denoted by n and has a form $(c_0 , c_1 , \ldots , c_{n-1})$.

The elements of the code word are selected from an alphabet field of q elements. In the ***binary code case***, which is our implementation, the field consists of two elements, 0 and 1. On the other hand, when the elements of the code word are selected from a field that has q alphabet elements, the code is **non-binary code**. As a special case when q is a power of 2 (i.e. $= 2^m$) where m is a positive integer, each element in the field can be represented as a set of distinct m bits. As indicated above, codes are constructed from fields with a finite number of q elements called ***Galois field*** and denoted by $GF\ (\ q\ )$.

In general, finite field $GF\ (\ q\ )$ can be constructed if q is a prime or a power of prime number. When q is a prime, the $GF(\ q\ )$ consist of the elements $\{0,1,2, \ldots. , q\ - 1\}$ with *addition* and *multiplication* operations are defined as a **modulo- q** . If q is a power of prime (i.e. $q\ =\ p^m$ where m is any positive integer), it is possible to extend the field $GF\ (p)$ to the field $GF\ (q\ =\ p^m\ )$. This is called the extension field of $GF\ (\ p\ )$ and in this case *multiplication* and *addition* operations are based on modulo- p arithmetic.

To construct the elements of the extension $GF\ (\ q\ =\ 2^m\ )$ from the binary $GF(2)$ with elements 0 and 1, a new symbol α is defined with multiplication operation properties as: $0. \alpha^i\ =\ \alpha^i .0\ = 0,\ 1. \alpha^i\ = \alpha^i .1\ =\ \alpha^i$ and $\alpha^i . \alpha^j\ =\ \alpha^j . \alpha^i\ =\ a^{i+j}$. The elements of the $GF\ (\ q\ =\ 2^m\ )$ that satisfy the above properties are $\{0,1, \alpha , \alpha^2 , \ldots , \alpha^j , \ldots\}$. As the field should has 2^m elements and be closed under multiplication α should satisfies the condition $\alpha^{q-1}\ =\ 1$. Hence; the elements of the extension $GF\ (\ q\ =\ 2^m\ )$ are $\{0,1, \alpha , \alpha^2 , \ldots , \alpha^{q-2}\}$ which is a commutative group under an addition and Multiplication (excluding the zero element) operations. α is called a ***primitive element*** since it can generate all other field elements and it is a root of a ***primitive polynomial*** $p(x)$. So each element in the field can be represented as a set of m-tuple bits. To make the picture clear, Table at page 42 shows the three representation for the elements of $GF(2^4)$ with a primitive polynomial $p(x) = x^4 + x + 1$ as our study case and implementation.

Channel Capacity

Entropy, *Information* and *Capacity*

According to the information theory and mathematical models for representing it, from the previous chapter, the entropy is defined as average information that is associated with the source of information, mathematically it represented by

$$H(X) = -\sum p(x)\, log\, p(x)$$

Now suppose the X is a binary source of information with zero probability is P the entropy is discussed in the previous chapter and it is

$$H(X) = \, -p\, log\, p \, - \, (1 - \, p)\, log(1 - \, p)$$

For discrete binary random variable the entropy could be denoted as *H(P)*

The Joint Entropy and Conditional Entropy

If the two variables X and Yare jointly distributed as P(X,Y) then the joint entropy is denoted as H(X,Y) and has mathematical equation

$$H(X,Y\,) = -\sum_{x} . \sum_{y} p(x,y)\, log\, p(x,y)$$

If one of these variables has the probability given another variable P(Y|X) then the conditional entropy is

$$H(Y|X\,) = -\sum_{x} . \sum_{y} p(x,y)\, log\, p(y|x)$$

$$= -\sum_{x} P(x) \sum_{y} p(x|\, y)\, log\, p(y|x)$$

Form the last equation the joint entropy could be rewritten according to a chain rule as

$$H(X,Y\,) = H(X) + H(Y|X)$$

This means that the entropy of X and Y is the entropy of X plus what Y has (entropy or uncertainty) given the knowing of X.

As for entropy the information is defined for a source X as

$$I(X) = -\sum \log p(x)$$

The mutual information is defined in previous chapter as

$$I(X,Y) = H(X) - H(X|Y)$$

$$I(X,Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

The channel capacity

The channel is defined as a probabilistic device that is fully described by the conditional probability function *P(Y|X)* and represented as

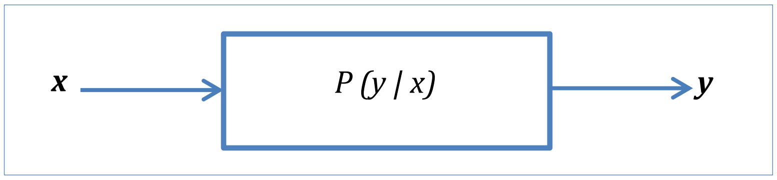


Figure 0.2 Mathematical Model Of The Channel

Now, the channel capacity " ***C*** " is defined as the maximum average information that can be transmitted over the channel per each channel use, mathematically :

$$C = max\{I(X,Y)\} = max\{H(X) - H(X|Y)\}$$

If the chancel carries binary data as input and output with probability of receive one or zero is P and the other is 1-P if the rows of the channel transition matrix that contains p(y|x) permutations of each other, and the columns are permutations of each other then the channel is a binary symmetric channel BSC this channel also as discrete memory less DMC . The previous channel characteristics could be represented as

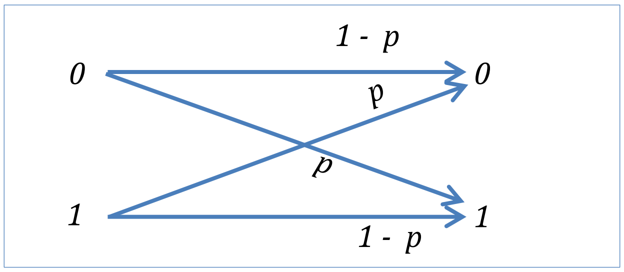


Figure 0.3 BSC with inputs, outputs and reception probabilities

The transition matrix for BSC channel is represented as $\begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$

For this channel the input X is {0,1} and the output Y is {0,1} .

Then the capacity for this channel could be obtained according to the equation like this knowing that *P(0|0)=1-P* , *P(0|1)=P* , *P(1|1)=1-P* and *P(1|0)=P* For a source X with probability symbols of P(0)=P(1)=0.5 for maximum information

$$C = max(I(X;Y)) = \Bigg[P(0)(1-p)log\frac{1-p}{(1-p)P(0)+pP(1)} + P(0)plog\frac{p}{pP(0)+(1-p)P(1)}$$

$$+ P(1)plog\frac{p}{(1-p)P(0)+pP(1)} + P(1)(1-p)log\frac{1-p}{pP(0)+(1-p)P(1)}\Bigg]\Bigg|_{P(1)=P(0)=\frac{1}{2}}$$

$$= (1-p)log2(1-p) + plog2p$$

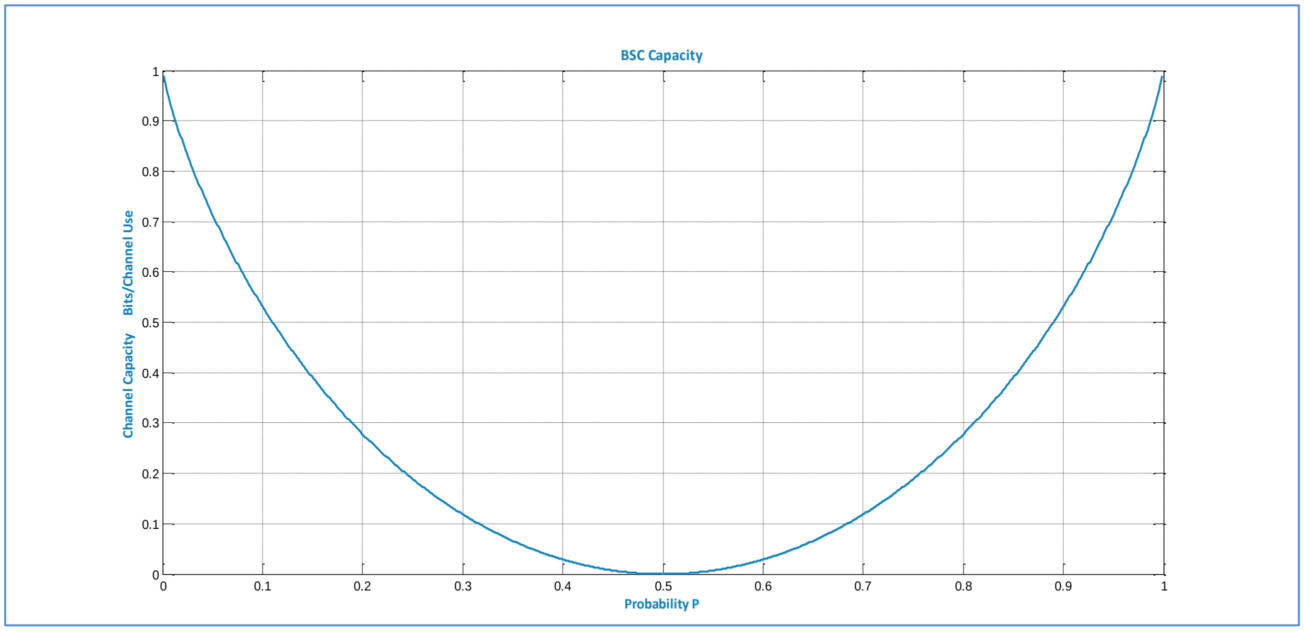


Figure 2.4 BCS Channel Capacity

This figure is deeply related with figure 1 of previous chapter for the entropy of the binary source , so the capacity of a binary symmetric channel BSC is

$$C = 1 - H(p)$$

Where p is the crossover probability and it is equal to *P(0|1) = P(1|0)* , it could be varied as $0 \leq p \leq 1/2$

Capacity for Additive White Gaussian Noise Channel

This channel model is more important since most of the channel coding techniques are designed and tested according to this model , the AWGN channel has a noise power density $\boldsymbol{N_0}$ that is normally distributed according to Gaussian distribution function , it is characterized as

$$\mathrm{P_N}(f) = \frac{N_0}{2}$$

$$p_N(f) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{n^2}{2\sigma^2}}$$

$$\sigma^2 = \frac{N_0}{2}$$

The capacity for continuous-time input and output signals of this channel is given by

$$C = \max\{H(Y)\} - H(N)$$

Where H(Y) the entropy of the continues time output signal and it is defined as :

$$H(Y) = -\int_{-\infty}^{\infty} p_Y\,(y)\log_2 p_Y(y)dy$$

And H(N) the entropy of the Gaussian Noise

$$H(N) = \frac{1}{2}\log_2(\pi e N_0)$$

Now if the signal is band limited with a bandwidth B then the capacity according to Shannon-Hartley theorem is

$$C = Blog_2(1 + \tfrac{S}{N})$$

Where S is the power of input signal to the channel and $N = N_0B$ is the noise power

Finally the capacity of an AWGN channel is deeply related to the input signal to the channel signal to noise ratio and the allocated bandwidth and it is dependent on the digital modulation scheme .

Finally the AWGN channel is chosen since this chapter will be an analysis implementation of a source coding techniques that aims at adding redundant bits to its input codeword to enable the error detection and correction and to reduce the probability of error over AWGN channel.

Block Channel Coding

BCH codes:

The Bose , **C**haudhuri and **H**ocquenghem codes(n, k) is a powerful random multiple error correcting cyclic codes.this class of codes is a generalization of Hamming codes for multiple error correction. There are two types of BCH code , Binary BCH codes were first discovered by A. Hocquenghem in 1959 and independently by R.C. Bose and D.K. Ray-Chaudhuri in 1960 in our case we take this deeply. On the other hand we have non-binary BCH codes, the most important subclass is the class of Reed-Solomon (RS) codes.

The decoding algorithms for BCH codes are syndrome decoding , Berlekamp's iterative algorithm, and Chien's search algorithm.

Binary primitive BCH codes

For any positive integers $m(m \geq 3)$ and $(t < 2^{m-1})$, there exist a binary BCH code with the following parameter :

Block length : $n = 2^m - 1$

Number of parity-check digits: $n - k \leq mt$

Minimum distance : $dmin \geq 2t + 1$

Where:

n: output codeword length
k:input bits
m:the order of primitive polynomial
t: number of errors that can be correct
$\boldsymbol{d}_{min}$: the minimum distance

clearly , this code capable to correcting t or fewer errors in the block of $n = 2^m - 1$ digits. To determine the generator polynomial $g(x)$ we need to know how many errors that the code can correct and the order of primitive polynomial $P(x)$ from this we can know the minimal polynomial , and the generator polynomial $g(x)$ of this code as specified in terms of its root of Galois Field $GF(2^m)$ that we illustrate in this chapter.

Mathematical Related Concepts

Minimal polynomials:

Let α be an element in $GF(2^m)$. We call the monic polynomial of smallest degree which has coefficients in $GF(2^m)$ and α and its conjugates as a root, the minimal polynomial of α.or in other word The minimal polynomial is irreducible over $GF(2^m)$, and any other non-zero polynomial GF with $P(\alpha) = 0$ is a (polynomial) multiple of P.

All the field elements of the form $a^j = (\alpha^i)^{2^l}$, $l \geq 1$ and i is odd, are called ***conjugates*** of α^i and all of them over the defined field have the ***same minimal polynomial*** (i.e. $m_j(x) = m_i(x)$)Hence, every even power of α in has the same minimal polynomial as the preceding odd power of α .

The primitive polynomial is used to obtain the generator polynomial $g(x)$,and m is the order of the primitive polynomial we will talk about this in the next part.

There is a lot of theorems for the minimal polynomial $m(x)$ but the most important theorems are chosen to be considered:

The first theorem which describes the first minimal polynomial over the finite filed states that :

If $m(x)$ be the minimal polynomial of an element a in $GF(2^m)$ and $m(x)$ is irreducible.

And by the definition the minimal polynomial

$$m_0(x) = x + 1$$

That is needed to determine the next minimal polynomial.

The second theorem describes how to construct the minimal polynomial and states that:

If $P(x)$ be a polynomial over $GF(2^m)$, and α is a root of $P(x)$ of order n in the multiplicative group of some field F of characteristic p. and Let r be the smallest integer so that $p^{r+1} \equiv 1$ mod n. then $\alpha, \alpha^p, \alpha^{p^\wedge 2}, \ldots, \alpha^{p^\wedge r}$ are all distinct roots of P(x).

To find the minimal polynomial :

$$m(x) = (x + \alpha)(x + \alpha^p) \ldots \ldots \ldots (x + \alpha^{p^r})$$

Where :
p: the order of α that is needed to find the minimal polynomial
r: integer number
To illustrate the previous mathematical theory we consider the field $GF(2^4)$.this means the order of $P(x)$ is 4 and the primitive polynomial may be $p(x) = x^4 + x^3 + 1$, and we need to determine what $m_2(x)$ will be

This have a root α , α^2 , α^4 , α^8 [*] only because after of this we have[$\alpha^{16} = \alpha$] so we ignore this case . or *in other* ***words we can say that α^2 , α^4 , α^8 are the conjugates of α so we can obtain the minimal polynomial*** m_1 ***and*** m_2***,***m_4***, and*** m_8 ***are the same***

$$m_1(x) = (x + \alpha)(x + \alpha^2)(x + \alpha^4)(x + \alpha^8)$$

After do this we find

$$m_1(x) = x^4 + x^3 + 1$$

And we can find any $m(x)$ by this way.

Generator Polynomial

The generator polynomial $g(x)$ of the t error correcting BCH code of block length $n = 2^m - 1$ of the lowest degree polynomial over $GF(2)$ that has

$$a\ , a^2, a^3, \dots \dots \dots .. \ , a^{2t}$$

Note that the generator polynomial of the binary BCH code is originally found to be the ***least common multiple of the minimum polynomials***

$$g(x) = LCM\{m_1(x).m_2(x).m_3(x) \dots \dots \dots .m_{2t}(x)\}$$

And

$$degree\ g(x) \leq mt$$

Hence the number of parity-check bits; $n - k$, of the code is at most mt.

However, generally, every even power of α in $GF(2^m)$ has the same minimal polynomial as some preceding odd power of a in $GF(2^{2m})$. As a consequence, the generator polynomial of the t-error-correcting binary BCH code can be reduced to

$$g(x) = lcm\{m_1(x).m_3(x) \dots .m_{2t-1}(x)\}$$

Where:

lcm : Least common multiple that means if we have any two or more minimal polynomial we take only one of them.

[*] α^6 , α^9 , α^{12} are also considered as α^3 conjugates and have the same minimal polynomials since the equation $a^j = (\alpha^i)^{2^l}, l \geq 1$ is satisfied.

Encoding of BCH code

After the preparation of the generator polynomial we go with steps to encode the data.

Step 1:

Generator matrix G is the most important and the complex step to encode the data that will be send and the dimensions of the G matrix is $k * n$.

Generator matrix G can be construct by put the coffecient of the generator polynomial $g(x)$ and followed by zeros with length of the parity bits $n - k$ and in the next row we shift one column and so on , and after this we need to make from column number 1 to k as identity because we deal with systematic code . And we obtain the most general generator matrix G as

$$G = [I_k|P]$$

Where:

$\boldsymbol{I_k}$: is the identity matrix of dimensions $k * k$
$\boldsymbol{P}$:parity matrix of dimensions $n - k * k$

Step 2 :

After we get the generator matrix G actually we need to obtain the codeword C as equation below

$$C = D.G$$

Where:
D: the information.
G: the generation matrix

Now the codeword is formed and data is ready for transmission.

Step 3:

The check matrix for a systematic code can be found directly from the generator matrix.

$$H = [P^T|I_{N-K}]$$

BCH(15,5,7)

Study Case & implementation

If the primitive polynomial P is being Considered $P(x) = x^4 + x + 1$, and the error correction $t = 3$ and the root in Galois field $GF(2^m)$ then we have the following specifications:

$$n = 2^4 - 1 = 15$$
$$dmin \geq 2 * 3 + 1 , dmin = 7$$
$$\text{The } degree\ g(x) \leq 4 * 3$$
$$n - k \leq mt , k = 5$$

Based on $(x) = x^3 + x + 1$:

Given 10011 , $P(x) = x^4 + x + 1$ if a is a root :

$$a^4 + a + 1 = 0$$

so

$$a^4 = a + 1$$

Form this it is needed to determine the minimal polynomial for all a,this is illustrated by this table

Filed elements					Minimal polynomial
Power representation	Polynomial representation	Binary representation			
$\boldsymbol{a^0}$	1	0001	1	$a^{15} = a^{30}$..	$x + 1$
$\boldsymbol{a^1}$	a^1	0010	2	$a^{16} = a^{31}$..	$x^4 + x + 1$
$\boldsymbol{a^2}$	a^2	0100	4	$a^{17} = a^{32}$..	$x^4 + x + 1$
$\boldsymbol{a^3}$	a^3	1000	8	$a^{18} = a^{33}$..	$x^4 + x^3 + x^2 + x + 1$
$\boldsymbol{a^4}$	$a^1 + 1$	0011	3	$a^{19} = a^{34}$..	$x^4 + x + 1$
$\boldsymbol{a^5}$	$a^2 + a$	0110	6	$a^{20} = a^{35}$..	$x^2 + x + 1$
$\boldsymbol{a^6}$	$a^3 + a^2$	1100	12	$a^{21} = a^{36}$..	$x^4 + x^3 + x^2 + x + 1$
$\boldsymbol{a^7}$	$a^3 + a^1 + 1$	1011	11	$a^{22} = a^{37}$..	$x^4 + x^3 + 1$
$\boldsymbol{a^8}$	$a^2 + 1$	0101	5	$a^{23} = a^{38}$..	$x^4 + x + 1$
$\boldsymbol{a^9}$	$a^3 + a^1$	1010	10	$a^{24} = a^{39}$..	$x^4 + x^3 + x^2 + x + 1$
$\boldsymbol{a^{10}}$	$a^2 + a^1 + 1$	0111	7	$a^{25} = a^{40}$..	$x^2 + x + 1$
$\boldsymbol{a^{11}}$	$a^3 + a^2 + a^1$	1110	14	$a^{26} = a^{41}$..	$x^4 + x^3 + 1$
$\boldsymbol{a^{12}}$	$a^3 + a^2 + a^1 + 1$	1111	15	$a^{27} = a^{42}$..	$x^4 + x^3 + x^2 + x + 1$
$\boldsymbol{a^{13}}$	$a^3 + a^2 + 1$	1101	13	$a^{28} = a^{43}$..	$x^4 + x^3 + 1$
$\boldsymbol{a^{14}}$	$a^3 + 1$	1001	9	$a^{29} = a^{44}$..	$x^4 + x^3 + 1$

To obtain the $m_2(x)$ it could be written as

$$m_2(x) = (x + a)(x + a^2)(x + a^4)(x + a^8)$$

After the multiplication

$$m_2\,(x) = x^4 + (a^8 + a^4 + a + a^4)x^3 + (a^{12} + a^9 + a^5 + a^{10} + a^6 + a^3)x^2 + (a^{13} + a^{14} + a^{11} + a^7)x + a^{15}$$

So[*];

$$m_2(x) = x^4 + x + 1$$

And by the same way all minimal polynomials could be found.

Now the $g(x)$ is the multiply of $m_1(x) and\ m_3(x) and\ m_5(x)$ since

$$g(x) = LCM\{m_1(x).\,m_3(x).\,m_5(x)\}$$

$$g(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)$$

Therefor;

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$

The coefficients of the $g(x)$ is

[1 0 1 0 0 1 1 0 1 1 1]

[*] *If we have $a^x + a^x$ then it is equal to zero , for each a^i it could be replaced from the table above for example $a^7 = a$ and $a^4 = a^2 + 1$, And $(a^2 + 1) + (a^2 + 1) = 0$ since, we add without carry*

The generation matrix of ***k*** by ***n*** that is needed can be built by putting the generator poly in the first row and followed by zeros and in the next row is the one bit rotation of previous row

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

Now , first k columns of G must be identity matrix $\boldsymbol{I}_{\boldsymbol{k}}$ this matrix is the ***reduced row echelon form*** of the previous matrix and it has the form $G=[I_k|P]=[I_5|P]$

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

Now The predefined parity check matrix $H=[P^T|I_{N-K}]=[P^T|I_{10}]$ is :

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

All possible codewords can be obtained by multiplying a Data of length k bit by generator matrix

$$C=d.G$$

Decoding of BCH Codes

Decoding process of the BCH codes is the most challenging task. Mainly, we have three decoding algorithms for BCH codes: Peterson- Gorentien-Zierler algorithm and Berlekamp- Massey algorithm, and syndrome, Peterson- Gorentien-Zierler algorithm is described here and also the syndrome algorithm that is implemented .

Now , assume that the received code word $(r_0, r_1, \dots \dots \dots., r_{n-1})$ is differs from the sent code Word $(c_0, c_1, \dots \dots \dots., c_{n-1})$ in $x^{i_1}, x^{i_2}, \dots \dots \dots., x^{i_v}$ positions , then the error code word will have a nonzero elements at these positions and the error polynomial can be written as

$$e(x) = x^{i_1} + x^{i_2} + \dots \dots \dots. + x^{i_v}$$

Where

V: the maximum error allowed or v=t

For this algorithm its necessary to compute the syndromes of the received code word polynomial (x) . define the syndrome S_j to be

$$S_j = r(a^j) \quad , j = 1,2, \dots \dots., 2t$$

Or

$$S_1 = x_1 + x_2 + \cdots + x_v$$

$$S_2 = x^2 + x^2 + \cdots + x^2$$

.

.

$$S_{2t} = x^2 + x^2 + \cdots + x^2$$

Where

$x_j = a^{i_l}$ is the error locations. Defining what is called error locator polynomial as

$$\Lambda(x) = \Lambda_v x^v + \Lambda_{v-1} x^{v-1} + \cdots + \Lambda_1 x + 1$$

So ,

$$\Lambda(x) = (1 - xX_1)(1 - xX_2) \dots (1 - xX_v)$$

that has zeros at $x = X_I$. It can be shown that from the two equation before can be coupled together in matrix form and written as

$$\underbrace{\begin{bmatrix} S_1 & S_2 & . & S_t \\ S_2 & S_3 & . & S_{t+1} \\ . & . & . & . \\ . & . & . & . \\ S_t & S_{t+1} & . & S_{2t-1} \end{bmatrix}}_{A} \begin{bmatrix} \Lambda_t \\ \Lambda_{t-1} \\ . \\ . \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} S_{t+1} \\ S_{t+2} \\ . \\ . \\ S_{2t} \end{bmatrix}$$

Peterson's algorithm is based on solving these matrix for $\Lambda_i\,'s$. If A is found to be singular that means we have less than t errors in the received code word. In this case we have to reconstruct a new syndrome matrix by deleting the two right most columns and the two bottom rows from A and .

solve a gain for $\Lambda_i\,'s$ excluding t Λ and so on. After $\Lambda_i\,'s$ are found the error correct polynomial defined in previous equation is constructed. Finally, the roots of $\Lambda(x)$ are to be found using Chien's search algorithm and the error locations set to be the reciprocal of these
roots.

The benefit we get from the described decoding is the increasing in the system reliability and robustness against noise for a given type of modulation as shown in this figure for BPSK

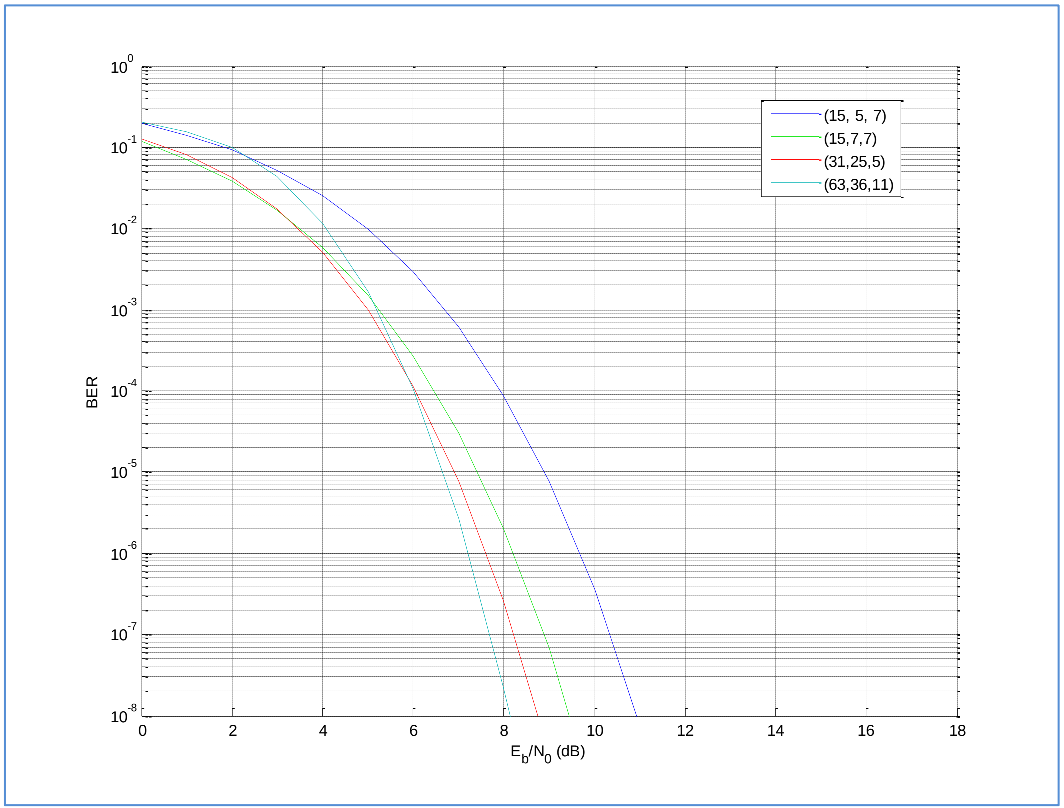
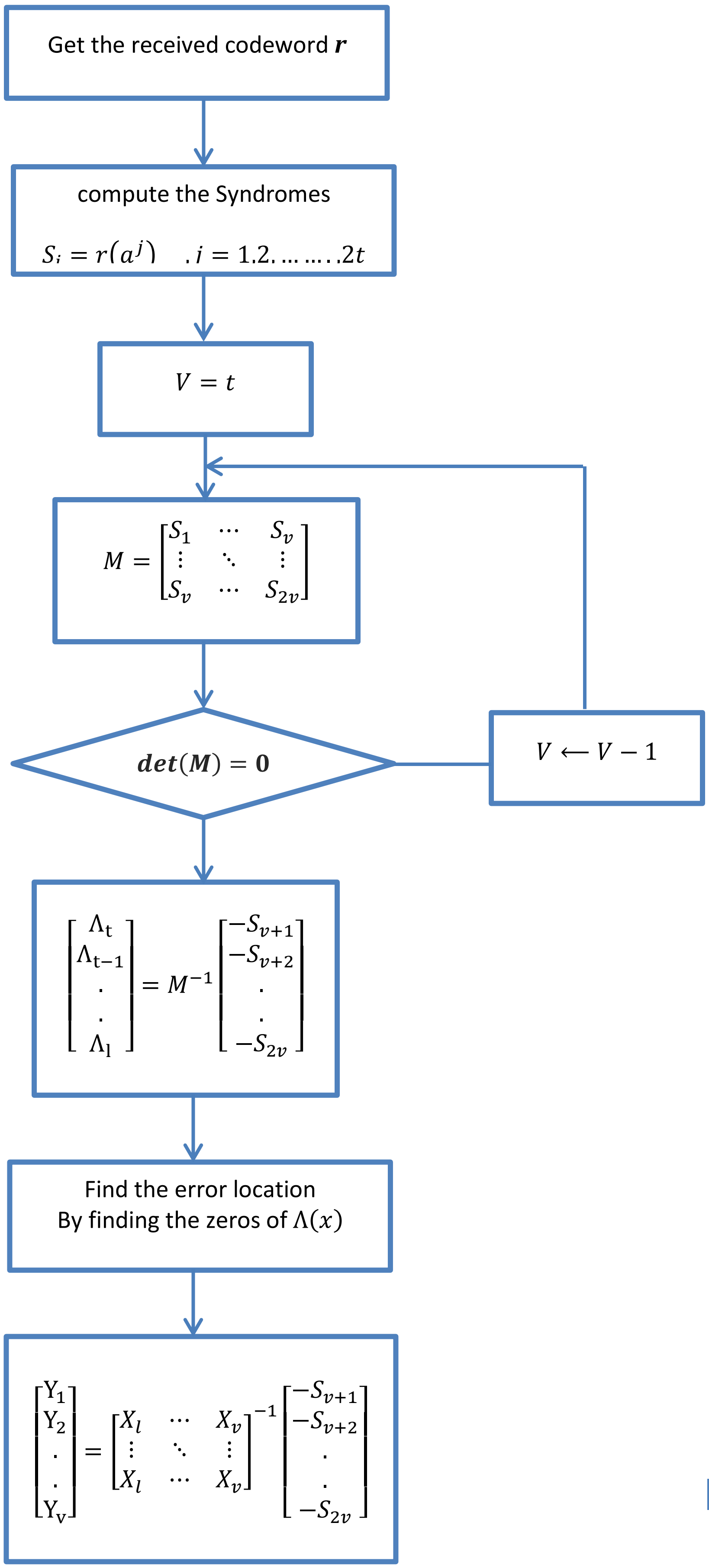


Figure 2.5 BCH codes performance comparison in a noisy channel - theoretically

Peterson- Gorentien-Zierler algorithm :


Get the received codeword $\boldsymbol{r}$
compute the Syndromes
$S_i = r(a^j) \quad . i = 1.2. \ldots \ldots . . 2t$
$V = t$
$M = \begin{bmatrix} S_1 & \cdots & S_v \\ \vdots & \ddots & \vdots \\ S_v & \cdots & S_{2v} \end{bmatrix}$
$\boldsymbol{det}(\boldsymbol{M}) = \boldsymbol{0}$
$V \leftarrow V - 1$
$\begin{bmatrix} \Lambda_t \\ \Lambda_{t-1} \\ . \\ . \\ \Lambda_l \end{bmatrix} = M^{-1} \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ . \\ . \\ -S_{2v} \end{bmatrix}$
Find the error location
By finding the zeros of $\Lambda(x)$
$\begin{bmatrix} Y_1 \\ Y_2 \\ . \\ . \\ Y_v \end{bmatrix} = \begin{bmatrix} X_l & \cdots & X_v \\ \vdots & \ddots & \vdots \\ X_l & \cdots & X_v \end{bmatrix}^{-1} \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ . \\ . \\ -S_{2v} \end{bmatrix}$

Syndrome Decoding of BCH Codes :

The decoding process of the BCH binary codes could be done by this algorithm , based on the parity check matrix

$$H = [P^T | I_{N-K}]$$

The multiplication of the parity check matric by a codword C is equal to zero

$$H.C^T = 0$$

If the input of the BCH decoder is the codeword $\boldsymbol{r}$ which contains errors so that

$$\boldsymbol{r} = C \oplus e$$

The definition states that the Syndrome $\boldsymbol{S}$ is the multiplication of the output codeword by the parity check matrix

$$S = H.r^T$$

The Syndrome $\boldsymbol{S}$ has a binary values of number 2^{n-k}

To obtain the error within the received codeword $\boldsymbol{r}$ from the following equation it could be obvious

$$S = H.r^T$$

$$S = H.(C \oplus e)^T$$

$$S = H.e^T \oplus H.C^T$$

$$S = H.e^T \oplus 0$$

So that the syndrome of an erroneous codeword $\boldsymbol{r}$ is the error in this codeword multiplied by the parity check matrix , and since the parity check matrix ***H is not invertible*** the error vector $\boldsymbol{e}$ could be found from a lookup table that contains all possible error patterns and each that corresponding to a Syndrome value .

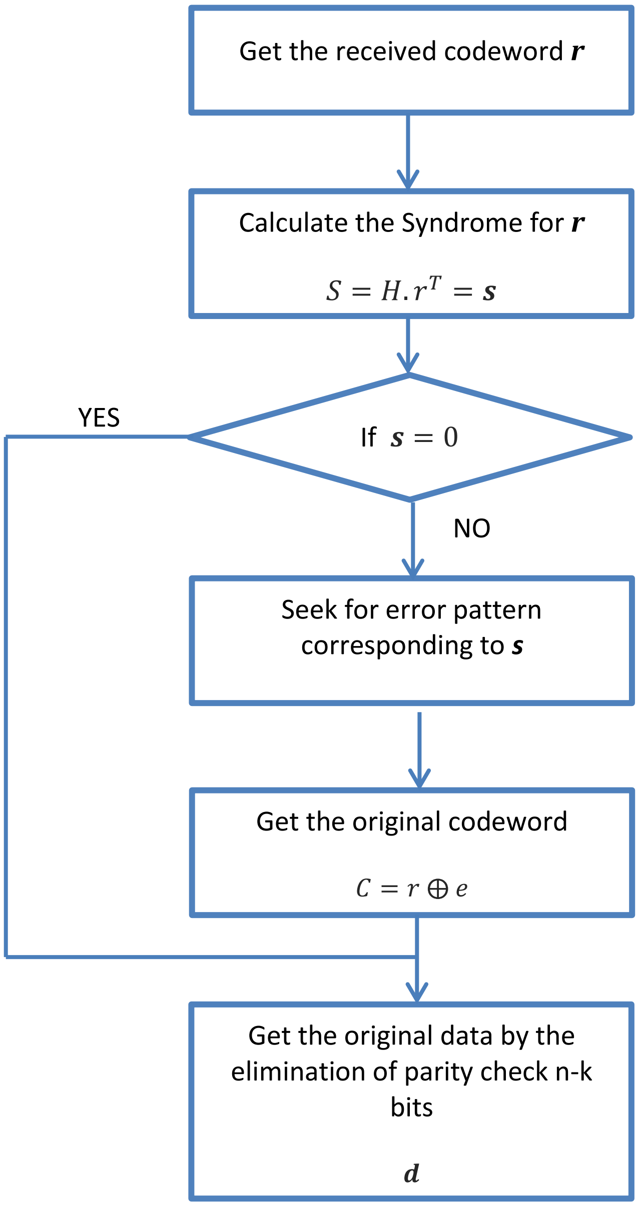
The lookup table which contains all syndromes and error patterns is called the Syndrome table and it should had a size of 2^{n-k} by n , the rows of this table could be sorted so that it will be corresponding to the syndrome decimal value in order to reduce the seeking time , the table for our case of BCH(15,5,7) is contains of 1024 rows by 15 columns .

The first value of the syndrome table is usually the correct codeword of syndrome zero and it is often be the all-zeros codeword

If the error pattern that is corresponding to the syndrome value is found then it is easy to obtain the correct original code word ***C.***

$$C = r \oplus e$$

This algorithm is implemented for BCH(n,k) and it could be summarized by the following flow chart



Get the received codeword $\boldsymbol{r}$
Calculate the Syndrome for $\boldsymbol{r}$
$S = H.r^T = \boldsymbol{s}$
If $\boldsymbol{s} = 0$
YES
NO
Seek for error pattern corresponding to $\boldsymbol{s}$
Get the original codeword
$C = r \oplus e$
Get the original data by the elimination of parity check n-k bits
$\boldsymbol{d}$

3

Chapter Three

ENCRYPTION

Data Security Implementation

This chapter is a description and implementation of an Advanced Encryption Standard Rijndael algorithm which is a symmetric encryption algorithm , another layer of security is added by implementing a steganography system

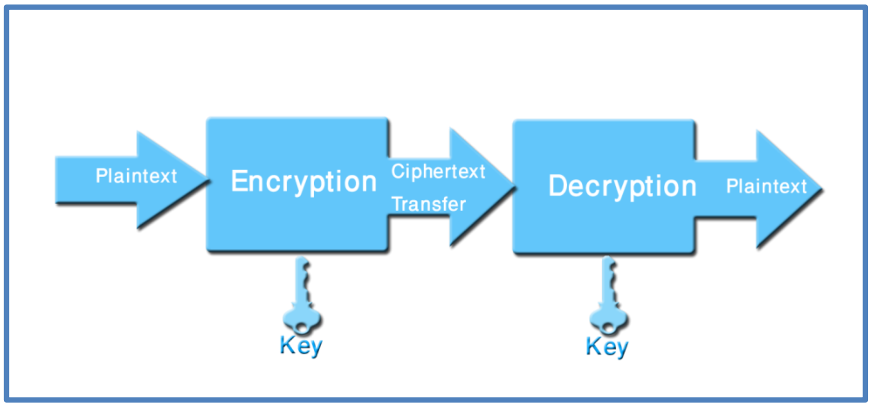
Encryption

Overview

Data that can be read and understood without any special measures is called plaintext or clear text. The method of disguising plaintext in such a way as to hide its substance is called encryption or in other words Encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key.

Symmetric encryption uses a single key to encrypt and decrypt the message. This means the person encrypting the message must give that key to the recipient before they can decrypt it.

Asymmetric encryption, also known as Public-Key encryption, uses two different keys - a public key to encrypt the message, and a private key to decrypt it. The public key can only be used to encrypt the message and the private key can only be used to decrypt it.



Symmetric Encryption Processes

In this chapter the symmetric encryption is conceded to be analyzed, described and implemented .The Advanced Encryption Standard is adopted in this chapter and it is represented by the a Rijndael algorithm with 128-bit cipher key.

AES

The National Institute of Standards and Technology, (NIST), provided proposals for the Advanced Encryption Standard, (AES). The AES is a Federal Information Processing Standard, (FIPS), which is a cryptographic algorithm that is used to protect electronic data [1]

The AES algorithm is a symmetric block cipher that can encrypt,(encipher), and decrypt, (decipher), information. Encryption converts data to an unintelligible form called cipher-text. Decryption of the cipher-text converts the data back into its original form, which is called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

Finalist candidate algorithms are five AES algorithms they are: MARS, RC6, Rijndael , Serpent, and Twofish. In this chapter we will take a Rijndael algorithm with key length 128 bit and implementing it by MATLAB.

Rijndael Algorithm:

The Rijndael algorithm was developed by Joan Daemen of Proton World International and Vincent Fijmen of Katholieke University at Leuven.

The main advantages of Rijndael algorithm ***flexibility*** , ***security*** Having the support of a rich algebraic structure enables Rijndael to be more secure than the average algorithm, Although it is flexible and defends against attacks, Rijndael ***does not require a lot of memory to operate***.

The input, the output and the cipher key for Rijndael are each bit sequences containing 128, 192 or 256 bits with the constraint that the input and output sequences have the same Length. A bit is a binary digit, 0 or 1, while the term 'length' refers to the number of bits in a sequence. In general the length of the input and output sequences can be any of the three allowed values but for the Advanced Encryption Standard (AES) .

Rijndael algorithm based on galois field $GF(2^8)$ generated by the primitive polynomial

$$p(x) = x^8 + x^6 + x^5 + x + 1$$

Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits. Assuming one byte [our case] equals 8 bits, the fixed block size of 128 bits is 128 ÷ 8 = 16 bytes.

The Basic Algorithm

For simplicity we choose a 128 bits length of data, this algorithm have a two branches encryption and key schedule
We will illustrate these deeply.

Specification of Rijndael Algorithm :

1.State

A byte in Rijndael is a group of 8 bits and is the basic data unit for all cipher operations. Such bytes are interpreted as finite field elements using polynomial representation, where a byte b with bits $b_0 b_1 \dots \dots b_7$ represents the finite field element:

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0 = \sum_{i=0}^{7} b_i x^i$$

All input , output and the cipher key are represented as a one dimensional array of bits for essentially programming the input is converted to two dimensional array of bytes and this process called state. In our case 128 bits, the state array is 4*4 of bytes however also the cipher key also need to convert it in the same style that is mean 4*4 array of bytes.
If there are a sequence of data $in_0 in_1 in_2 in_3 \dots \dots . in_{15}$ that want to convert into a state that can be done by

$$s[r, c] = data[r + 4c]$$

Where
r : row
c : column

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

2. Transformation :

2.1 S-Box

An S-Box takes some number of input bits, m, and transforms them into some number of output bits, n: an m×n S-Box can be implemented as a lookup table with 2m words of n bits each and in our implementation we are using lookup table as we shown below.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	01	DE	A5	63	6A	26	7E	C9	7F	67	A4	05	03	64	2E	32
1	AE	04	BA	B5	B2	50	3A	17	08	82	0F	94	ED	7C	F5	71
2	6C	24	8A	B9	D9	E2	CC	38	B0	6D	EC	8D	3D	CA	9D	A9
3	B1	6F	E3	80	35	3B	B6	4A	E7	21	55	B3	68	BD	6E	19
4	F0	16	6B	EB	59	28	1D	2C	D6	41	3F	D5	C7	3E	8F	89
5	36	88	45	8E	DD	8C	34	CD	2F	A2	22	F7	AF	29	9E	91
6	E9	86	C0	40	18	83	F6	25	C2	A1	54	AB	66	EF	A6	E8
7	B4	5A	84	C4	52	5F	E5	02	5D	EA	D4	DB	D2	85	5B	27
8	00	44	93	47	DF	46	1A	D7	37	51	49	A8	1C	B8	4F	F9
9	C5	43	60	20	0C	57	7B	A3	61	E1	2A	E4	33	C6	53	74
A	0B	9A	76	E6	65	FF	C3	3C	9F	75	56	F8	69	F3	9C	87
B	7D	F4	5E	FD	BF	23	0D	DA	AA	99	95	9B	0E	5C	96	39
C	D3	90	30	92	C1	2D	1B	E0	81	97	15	72	10	1F	98	62
D	78	4D	13	73	AC	CE	D0	1E	FE	8B	2B	0A	06	C8	4E	F2
E	CB	CF	58	7A	EE	A0	B7	DC	12	42	FB	FC	07	14	4B	AD
F	D8	48	77	11	D1	A7	BC	70	F1	FA	BB	79	09	BE	4C	31

Very simple to use its each byte in the state that represent in HEX the first one is the row in the s-box and the second number is the column number and the intersection between the two is the number transformed, it is done for all number in the state.

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline\end{array} \xrightarrow{\text{Apply s-box Tr}} \begin{array}{|c|c|c|c|}\hline s_{0,0}' & s_{0,1}' & s_{0,2}' & s_{0,3}' \\ \hline s_{1,0}' & s_{1,1}' & s_{1,2}' & s_{1,3}' \\ \hline s_{2,0}' & s_{2,1}' & s_{2,2}' & s_{2,3}' \\ \hline s_{3,0}' & s_{3,1}' & s_{3,2}' & s_{3,3}' \\ \hline\end{array}$$

But to know how this lookup table obtain the S-box is generated by determining the multiplicative inverse for a given number in $GF(2^8) = GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$, Rijndael's finite field (zero, which has no inverse, is set to zero). The multiplicative inverse is then transformed using the following affine transformation.

$$\begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\end{bmatrix}\oplus\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}$$

Where $x_0 \ldots \ldots x_7$is the multiplicative inverse as a vector.

The matrix multiplication can be calculated by the following algorithm:

- Store the multiplicative inverse of the input number in two 8-bit unsigned temporary variables: s and x.
- Rotate the value s one bit to the left; if the value of s had a high bit (eighth bit from the right) of one, make the low bit of s one; otherwise the low bit of s is zero.
- Exclusive or the value of x with the value of s, storing the value in x
- For three more iterations, repeat steps two and three; steps two and three are done a total of four times.
- The value of x will now have the result of the multiplication.

After the matrix multiplication is done, exclusive or the value by the decimal number 99 (the hexadecimal number 0x63, the binary number 1100011, and the bit string 11000110 representing the number in LSb first notation).

Or we can put this matrix into an equation as follow

$$b_i' = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

Where

$c_i = 63\ hex$ And b_i each bit of the byte that we need to transform.

2.2 Shift Rows

The Shift Rows transformation operates individually on each of the last three rows of the state by cyclically shifting the bytes depends on the number of the rows, we don't have shift in row(0) but row(1) we have a one cycle shift and so on.as illustrated below

$s_{0,0}{}'$	$s_{0,1}{}'$	$s_{0,2}{}'$	$s_{0,3}{}'$
$s_{1,0}{}'$	$s_{1,1}{}'$	$s_{1,2}{}'$	$s_{1,3}{}'$
$s_{2,0}{}'$	$s_{2,1}{}'$	$s_{2,2}{}'$	$s_{2,3}{}'$
$s_{3,0}{}'$	$s_{3,1}{}'$	$s_{3,2}{}'$	$s_{3,3}{}'$

$$\xrightarrow{\text{Apply shift rows Tr}}$$

$s_{0,0}{}'$	$s_{0,1}{}'$	$s_{0,2}{}'$	$s_{0,3}{}'$
$s_{1,1}{}'$	$s_{1,2}{}'$	$s_{1,3}{}'$	$s_{1,0}{}'$
$s_{2,2}{}'$	$s_{2,3}{}'$	$s_{2,0}{}'$	$s_{2,1}{}'$
$s_{3,3}{}'$	$s_{3,0}{}'$	$s_{3,1}{}'$	$s_{3,2}{}'$

Where

$s_{i,j}{}'$ is the state after transformed by s-box

2.3 Mix columns

The Mix Columns transformation acts independently on every column of the state take each column and multiply it by mix columns matrix as follow:

$$\begin{bmatrix} s_{0,j}{}' \\ s_{1,j}{}' \\ s_{2,j}{}' \\ s_{3,j}{}' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix}$$

This can also be seen as the following:

$$\begin{aligned} s'_{0,j} &= 02 * s_{0,j} \oplus \; 03 * s_{1,j} \; \oplus \; 01 * s_{2,j} \oplus 01 * s_{3,j} \\ s'_{1,j} &= 01 * s_{0,j} \oplus \; 02 * s_{1,j} \; \oplus \; 03 * s_{2,j} \oplus 01 * s_{3,j} \\ s'_{2,j} &= 01 * s_{0,j} \oplus \; 01 * s_{1,j} \; \oplus \; 02 * s_{2,j} \oplus 03 * s_{3,j} \\ s'_{3,j} &= 03 * s_{0,j} \oplus \; 01 * s_{1,j} \; \oplus \; 01 * s_{2,j} \oplus 02 * s_{3,j} \end{aligned}$$

This can be done only by multiplication in the field . as we illustrated in channel coding chapter .

2.4 Add round key

In the add Round Key transformation, from the key schedule (the round key described later) are each added (XOR' d) into the columns of the state so that:

$$\begin{bmatrix} s_{0,j}{}'' \\ s_{1,j}{}'' \\ s_{2,j}{}'' \\ s_{3,j}{}'' \end{bmatrix} = \begin{bmatrix} s_{0,j}{}' \\ s_{1,j}{}' \\ s_{2,j}{}' \\ s_{3,j}{}' \end{bmatrix} \oplus \begin{bmatrix} w_{0,i} \\ w_{1,i} \\ w_{2,i} \\ w_{3,i} \end{bmatrix}$$

This is done four times in each round to apply this transform.

3.Key schedule

In this algorithm the length of cipher key is 128 bits, which is generated into 4 * 4 bytes. Label the first four columns as $W(0)W(1)W(2)W(3)$ respectively . From this primary key we need to generate round keys for all 10 rounds with 128 bits .and we need in these steps S-box and Rcon that is illustrated as follows:

Rcon (Round Column)

Is something that should be used when we generate a first column in each round

01	02	04	08	10	20	40	80	1b	39
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

We can get the first columns of all round keys by

$$W(i) = W(i-1)\ xor\ W(i-4)\ xor\ Rcon(R)$$

Where:
i : the number of the column
$W(i)$:the round key at such column
R : the number of the round

This equation can be used at first column in each round *only.*

Now we need to generate all key rounds and we will illustrate it step by step :

Round (0):

It takes the primary key $W(0)W(1)W(2)W(3)$

Round (1):

The first column of this round key if we called (4) , that takes the last column of the primary key $W(3)$ and rotate it once then go to the S-box and replace all elements of this column or that can be expressed as

$$W(4) = W(3)\ xor\ W(0)\ xor\ Rcon(1)$$

And the remaining columns can be obtained by

$$W(i) = W(i-1)\ xor\ W(i-4)$$

From round 2 to round 10 keys which is the same idea in round 1 key

For the first columns of these keys are used

$$W(i) = W(i-1)\ xor\ W(i-4)\ xor\ Rcon(R)$$

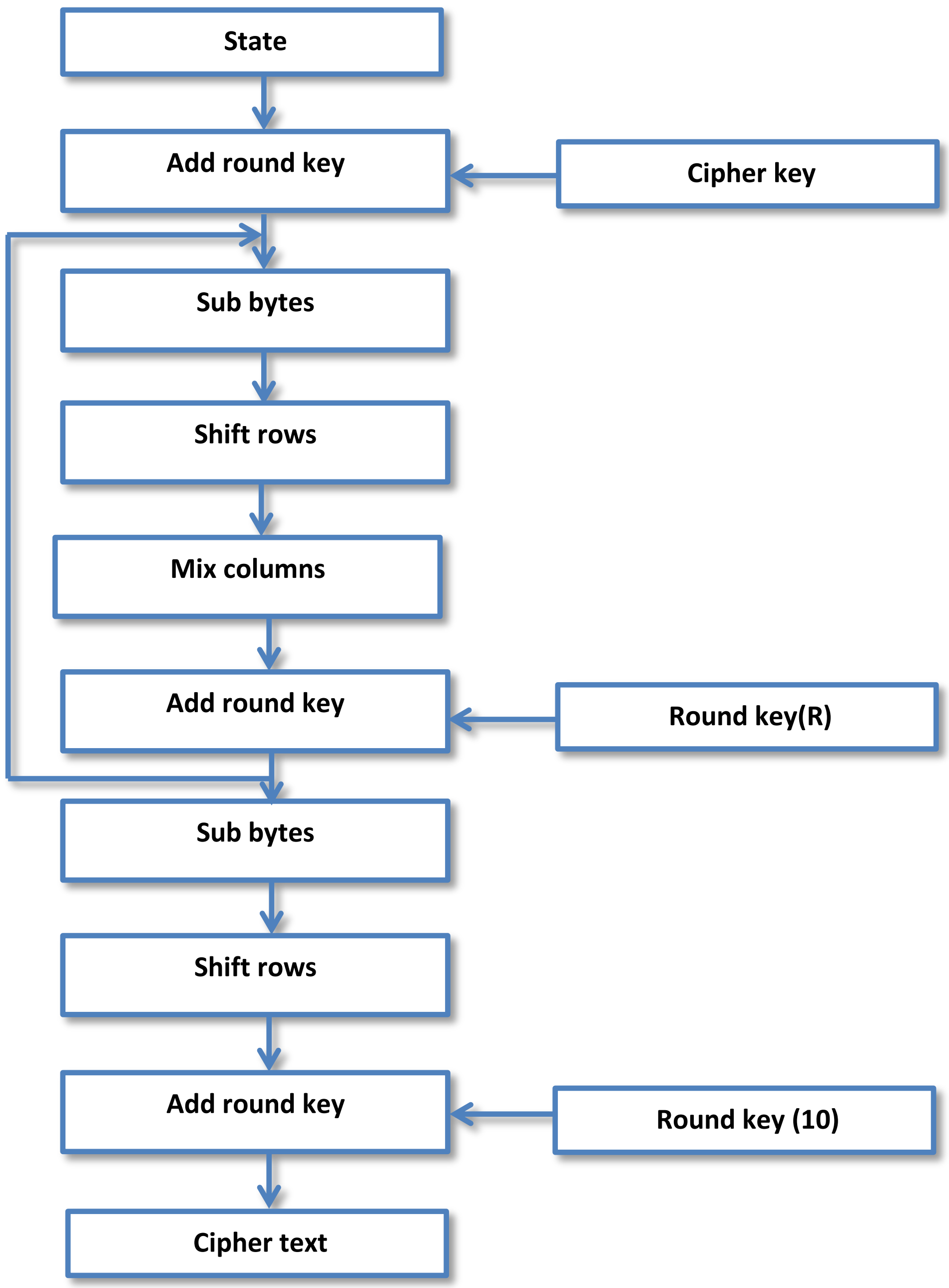
And for others are used

$$W(i) = W(i-1)\ xor\ W(i-4)$$

After perform this operation, we have finished the most important step before the second branch which is encryption.

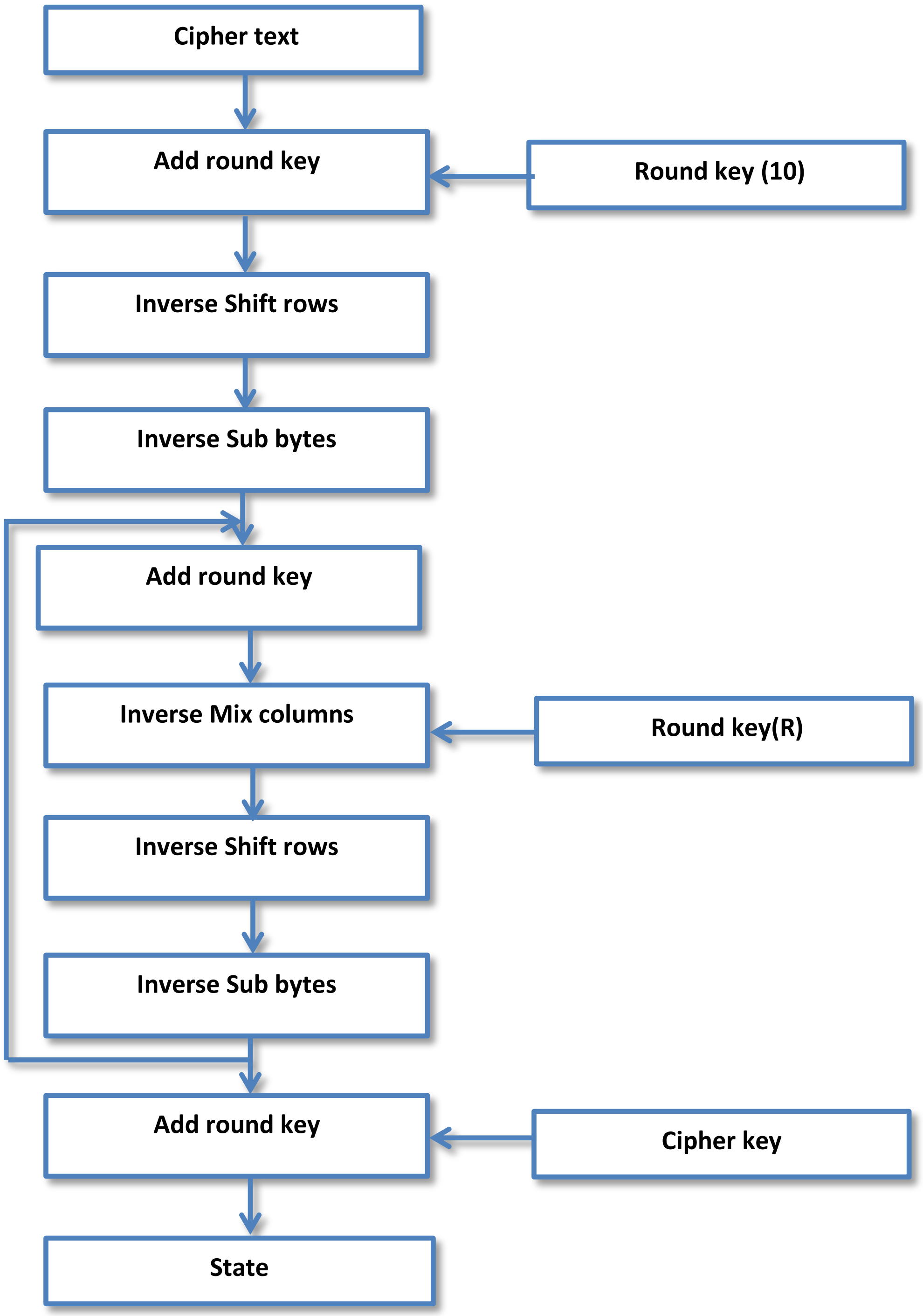
Encryption process

Encryption algorithm



Decryption

To decrypt, perform cipher in reverse order, using inverses of the transformation and the same key schedule.



Inverse transformation:

1.Inverse S-Box

As we see in the lookup table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	80	00	77	0C	11	0B	DC	EC	18	FC	DB	A0	94	B6	BC	1A
1	CC	F3	E8	D2	ED	CA	41	17	64	3F	86	C6	8C	46	D7	CD
2	93	39	5A	B	21	67	05	7F	45	5D	9A	DA	47	C5	0E	58
3	C2	FF	0F	9C	56	34	50	88	27	BF	16	35	A7	2C	4D	4A
4	63	49	E9	91	81	52	85	83	F1	8A	37	EE	FE	D1	DE	8E
5	15	89	74	9E	6A	3A	AA	95	E2	44	71	7E	BD	78	B2	75
6	92	98	CF	03	0D	A4	6C	09	3C	AC	04	42	20	29	3E	31
7	F7	1F	CB	D3	9F	A9	A2	F2	D0	FB	E3	96	1D	B0	06	08
8	33	C8	19	65	72	7D	61	AF	51	4F	22	D9	55	2B	53	4E
9	C1	5F	C3	82	1B	BA	BE	C9	CE	B9	A1	BB	AE	2E	5E	A8
A	E5	69	59	97	0A	02	6E	F5	8B	2F	B8	6B	D4	EF	10	5C
B	28	30	14	3B	70	13	36	E6	8D	23	12	FA	F6	3D	FD	B4
C	62	C4	68	A6	73	90	9D	4C	DD	07	2D	E0	26	57	D5	E1
D	D6	F4	7C	C0	7A	4B	48	87	F0	24	B7	7B	E7	54	01	84
E	C7	99	25	32	9B	76	A3	38	6F	60	79	43	2A	1C	E4	6D
F	40	F8	DF	AD	B1	1E	66	5B	AB	8F	F9	EA	EB	B3	D8	A5

Very simple to use its each byte in the cipher that represent in HEX the first one is the row in the Inverse S-Box and the second number is the column number and the intersection between the two is the number transformed, do it for all number in the cipher text.

2. Inverse shift rows:

Its completely inverse of the shift rows in the encryption side , assume we have this matrix and we need to transform it.

$$\begin{array}{|c|c|c|c|}\hline s_{0,0}' & s_{0,1}' & s_{0,2}' & s_{0,3}' \\ \hline s_{1,1}' & s_{1,2}' & s_{1,3}' & s_{1,0}' \\ \hline s_{2,2}' & s_{2,3}' & s_{2,0}' & s_{2,1}' \\ \hline s_{3,3}' & s_{3,0}' & s_{3,1}' & s_{3,2}' \\ \hline \end{array} \xrightarrow{\text{Apply inverse shift row Tr}} \begin{array}{|c|c|c|c|}\hline s_{0,0}' & s_{0,1}' & s_{0,2}' & s_{0,3}' \\ \hline s_{1,0}' & s_{1,1}' & s_{1,2}' & s_{1,3}' \\ \hline s_{2,0}' & s_{2,1}' & s_{2,2}' & s_{2,3}' \\ \hline s_{3,0}' & s_{3,1}' & s_{3,2}' & s_{3,3}' \\ \hline \end{array}$$

3. Inverse mix Columns

The inverse Mix Columns transformation acts independently on every column of the cipher take each column and multiply it by mix columns matrix as follow:

$$\begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} s_{0,j}' \\ s_{1,j}' \\ s_{2,j}' \\ s_{3,j}' \end{bmatrix}$$

This can also be seen as the following:

$$s_{0,j} = 0E * s_{0,j}' \oplus\ 0B * s_{1,j}' \ \oplus\ 0D * s_{2,j}' \oplus 09 * s_{3,j}'$$
$$s_{1,j} = 09 * s_{0,j}' \oplus\ 0E * s_{1,j}' \ \oplus\ 0B * s_{2,j}' \oplus 0D * s_{3,j}'$$
$$s_{2,j} = 0D * s_{0,j}' \oplus\ 09 * s_{1,j}' \ \oplus\ 0E * s_{2,j}' \oplus 0B * s_{3,j}'$$
$$s_{3,j} = 0B * s_{0,j}' \oplus\ 0D * s_{1,j}' \ \oplus\ 09 * s_{2,j}' \oplus 0E * s_{3,j}'$$

This can be done only by multiplication in the field .

4.inverse Add round key

In the inverse add Round Key transformation, from the key schedule (the round key described before) are each added (XOR' d) into the columns of the cipher so that:

$$\begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} = \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} \oplus \begin{bmatrix} w_{0,i} \\ w_{1,i} \\ w_{2,i} \\ w_{3,i} \end{bmatrix}$$

It is done four times in each round to apply this transform.
After carrying out these transformation like decryption flow chart and using the same cipher key that used in encryption we should recover the same file that encrypted.

Assume we have a text file which contains murad,mohammad

murad,mohammad $\xrightarrow{\text{ASCII representation}}$ 109 117 114 97 100 44 109 111 104 97 109 109 97 100

In this example we have 14 bytes only so we need to add two zeros as result we need 16 bytes.
Cipher key : [43 40 171 9 126 174 247 207 21 210 21 79 22 166 136 60]

Then the cipher text is $\hat{a}n\grave{o}j\ 1'b\acute{y}i$

And at the decryption side the recovered data is murad,mohammad

Steganography

As a second data security layer

Steganography can be defined as the science of writing hidden messages in such a way that no one, except the sender and intended recipient, can detect the existence of the message

Steganography has a difference with cryptography. While the Cryptography involves the encryption algorithms that change the message symbols as it done with the AES . An encrypted message is obvious. One may not know the intended meaning of the message, but it is obvious that it exists!

However, Steganography makes an effort to hide the fact that the encrypted data even exists, so not drawing attention to it. It replaces bits of unused data into the file- (i.e. graphics, sound, text, audio, or video) with some other bits that have been obtained secretly or unauthorized manner. Our implementation employs the embedding with image data.

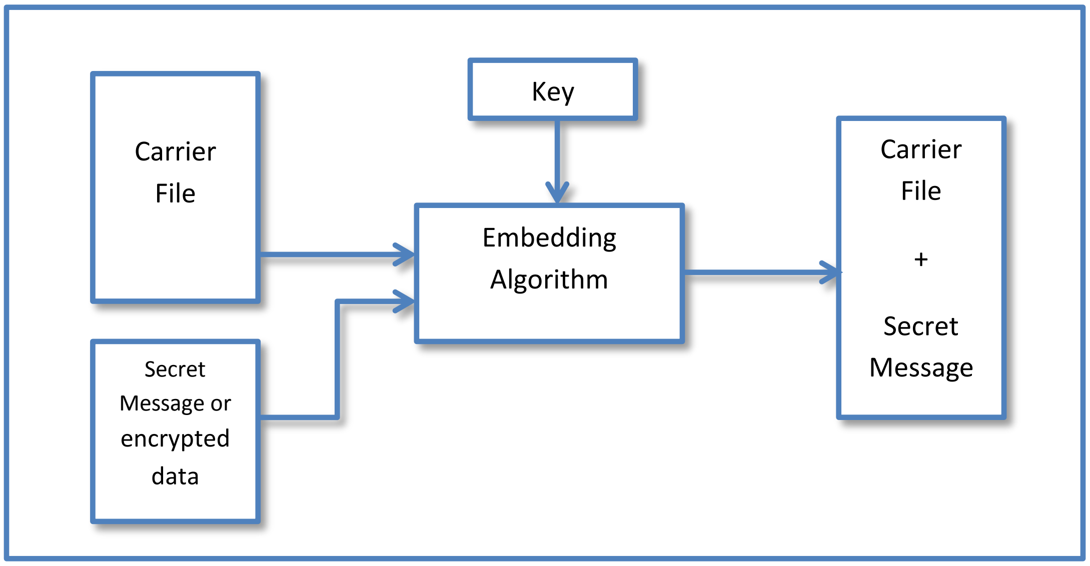

Key
Carrier File
Embedding Algorithm
Carrier File + Secret Message
Secret Message or encrypted data

Figure 3.1 General Steganography Algorithm Block Diagram

The embedding algorithm that is used the Least significant bit Embedding LSB and its depends on bits replacement of a source file's least significant bit according to the message or data information bits.

This method could be used apart from the encryption to provide less computation time and difficult data transformations, or could be used with it.

4

Chapter Four

SYSTEM IMPLEMENTATION

MATLAB Implementation of the System

This chapter introduces the programming codes and MATLAB functions that are created so that to carry out the operations and tasks that each block in the digital communication system performs.

System Performance Simulations

The Source Coder

In order to carry out this test, there is a set of different size and type files to be encoded, the compressed or encoded files (the output) are then compared with the input files according to some parameters. The coders that are used Huffman, Run-Length, and BWT.

File name	Size Before [Bytes]	BWT [Bytes]	Huffman [Bytes]	RunLength [Bytes]	RAR [Bytes]	Average code Length for Huffman [Bits/symbol]	Average code Length for BWT [Bits/symbol]	Entropy Of the source [Bits/symbol]
File 1	1k	0.532k	0.664 k	1.53 k	0.47	4.40	2.94	4.38
File 2	1.36 k	0.787k	0.874 k	2.67 k	0.77	4.40	3.59	4.38
File 3	3.66 k	1.96 k	2.40 k	7.16 k	1.88	4.89	3.74	4.85
File 4	4.12k	1.95 k	2.62 k	7.72 k	1.82	4.75	3.27	4.72
File 5	6.08 k	2.92 k	4.05 k	11.9 k	2.82	5.09	3.5	5.05
File 6	12.7 k	5.02 k	7.21 k	25 k	4.87	4.41	2.97	4.39
File 7	16.0 k	6.02 k	9.14 k	31.5 k	5.85	4.46	2.85	4.44
File 8	18.1 k	6.75 k	10.2 k	35.6 k	6.72	4.45	2.85	4.43

The next figure shows the difference in the size after and before for the chosen text files

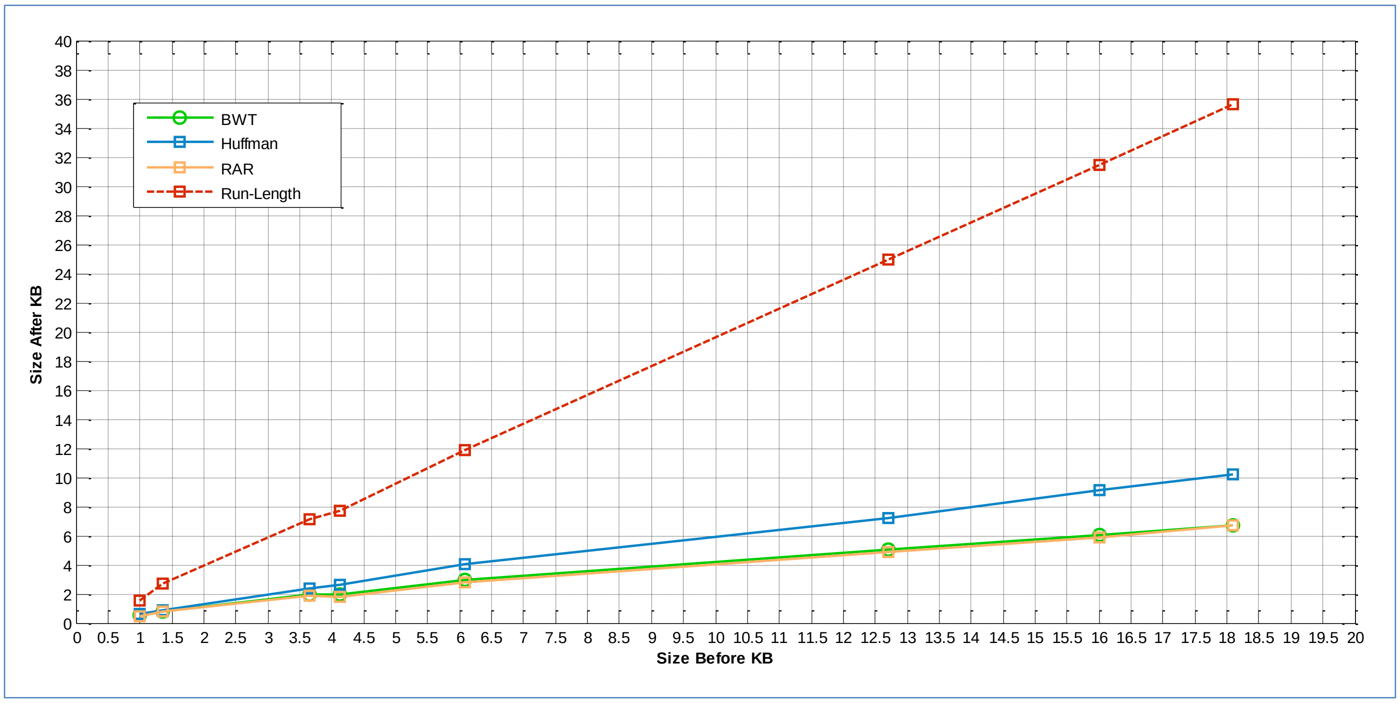


Figure 4.1 comparison plot for the three encoders compared with *RAR*

Run-length encoder seems not suitable for major types of files and it is may implemented within a specific algorithm which has a considerable number of repeated successive symbols which is not usually occurs in an ordinary text file. As a result the run length coding is excluded form the next comparisons.

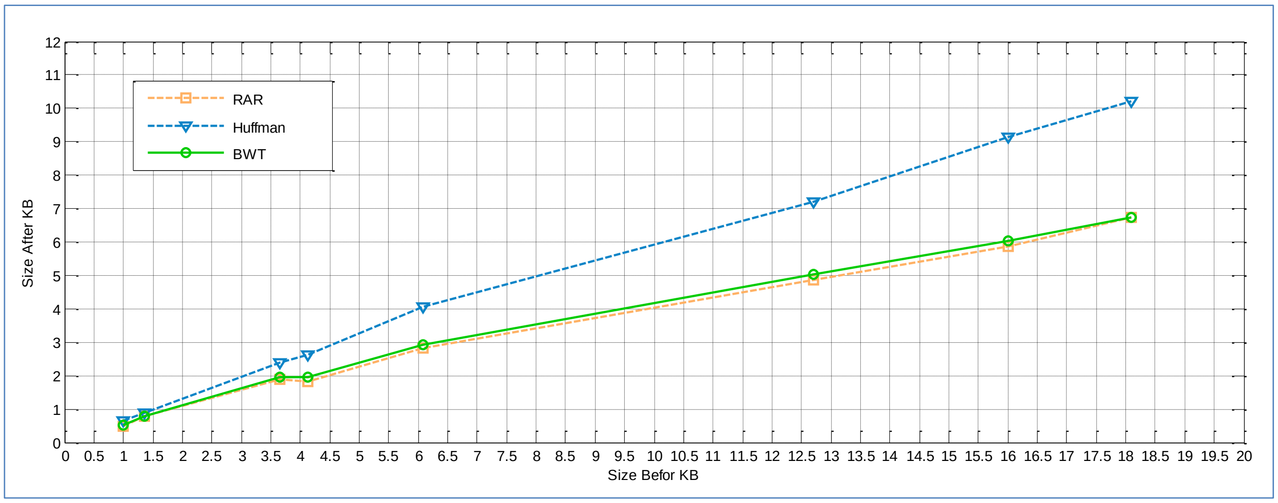


Figure 4.2 the performance of the adopted text codes compared with *RAR*

The output of source coder

In this implementation the output file should be include the decoding information represented in a header attached to a file.

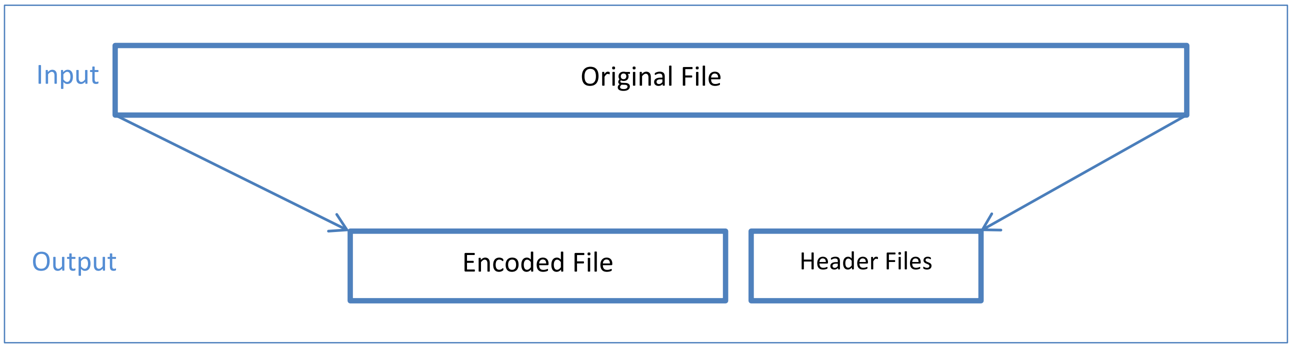


Figure 4.3 output file structure compared with input file

The header data contains important details for the decoding process to be carried out successfully, for each coder the content of the header is different, this is shown in the next section of this chapter.

The same test is done to voice wave files and the results for the Huffman encoder are shown in the table below

File name (.wav)	Size Before [K Bytes]	Size after [K Bytes]	Average code Length for [Bits/symbol]	Entropy [Bits/symbol]
hello	25.7	20.6	5.95	5.91
how	29.2	23.3	5.97	5.95
hi	30.1	21.4	5.357	5.317
Good	38.6	35.0	6.94	6.91
ok	55.7	49.1	6.768	6.74

The Channel Coder and Channel

While the main parameters in the previous section are the size of the file and average code length the channel forward error correction code testing parameter is the bit error rate, this parameter is affected by the modulation type and it is order and the FEC technique, the implementation of BCH (15,5,7) is used in this test, this type of channel FEC codes can correct up to 3 errors in the code-word of 15 bits.

To test the correction capability of this type, the following code-word is assumed to be the corresponding to the 5-bit message signal:

Input 5 bit word:	0 1 0 0 1
Output 15 bit code word:	0 1 0 0 1 **1 0 1 1 1 0 0 0 0 1**

A random three errors are introduced to the code word to check the output of the decoder:

Erroneous code word:	0 1 0 1 1 **1 0 1 0 1 0 1 0 0 1**
Recovered message:	0 1 0 0 1

Now a four random errors are introduced to the same code-word:

Erroneous code word:	0 0 0 0 1 **1 0 0 0 1 0 0 0 0 0**
Recovered message:	0 0 0 0 0

Now a binary file 6.4 K Bytes is tested using this coder for different modulation schemes and the following simulation results for bit error rate is obtained

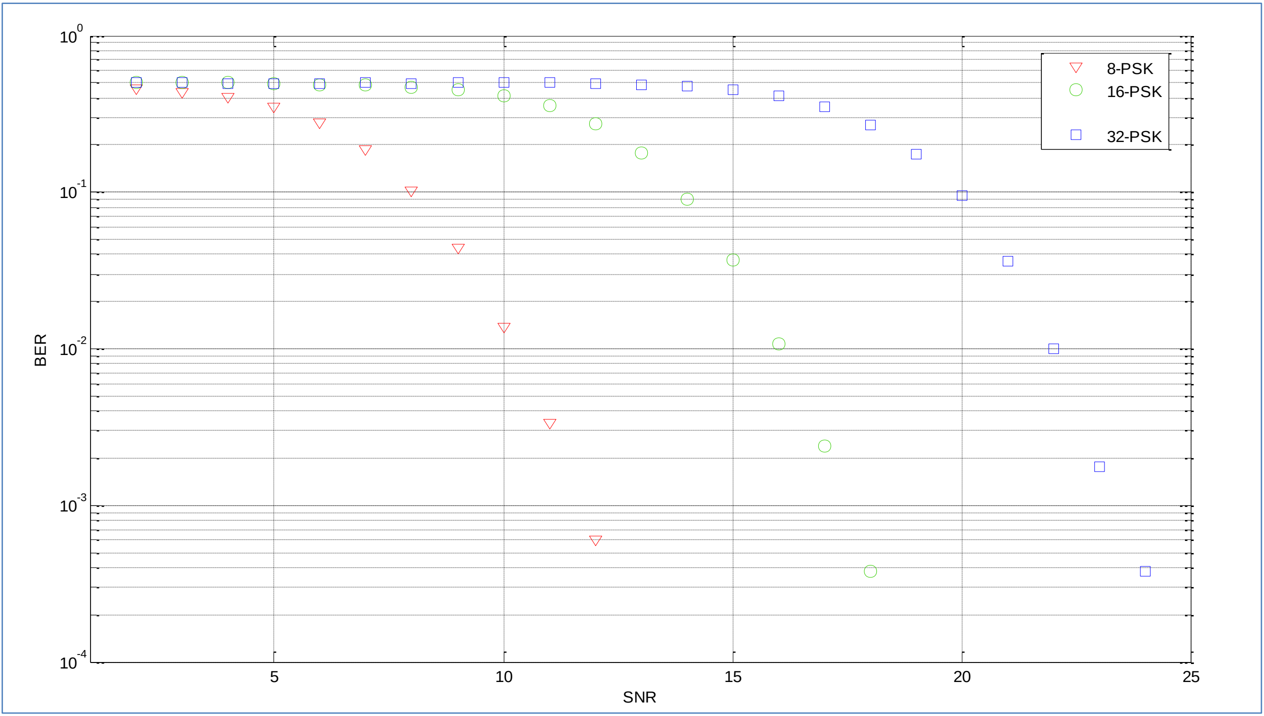


Figure 4.4 BER against SNR for different coded PSK schemes with BCH (15, 5, 7)

The above results is compared with the theoritical results and shows some differences due to finite length of input data

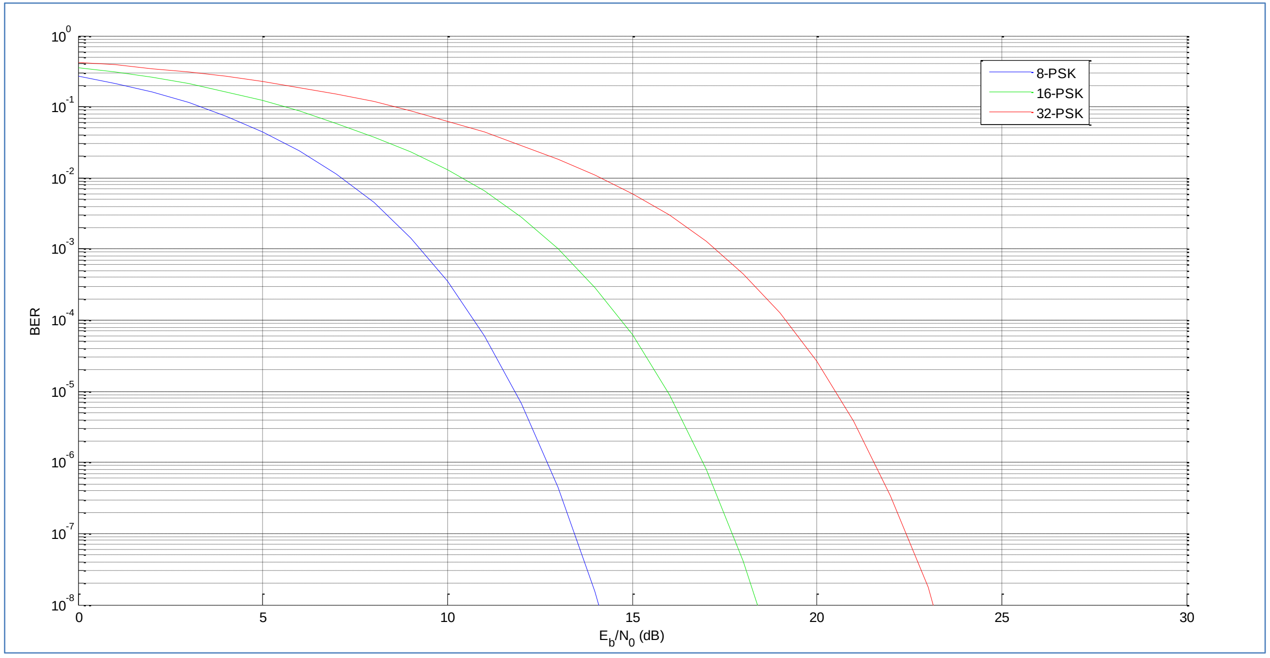


Figure 4.5 Theoretical results for the same test

The theoretical performance of the implemented BCH type shows that has a great BER than others but it has a good error corrections capability with shorter block size to be coded

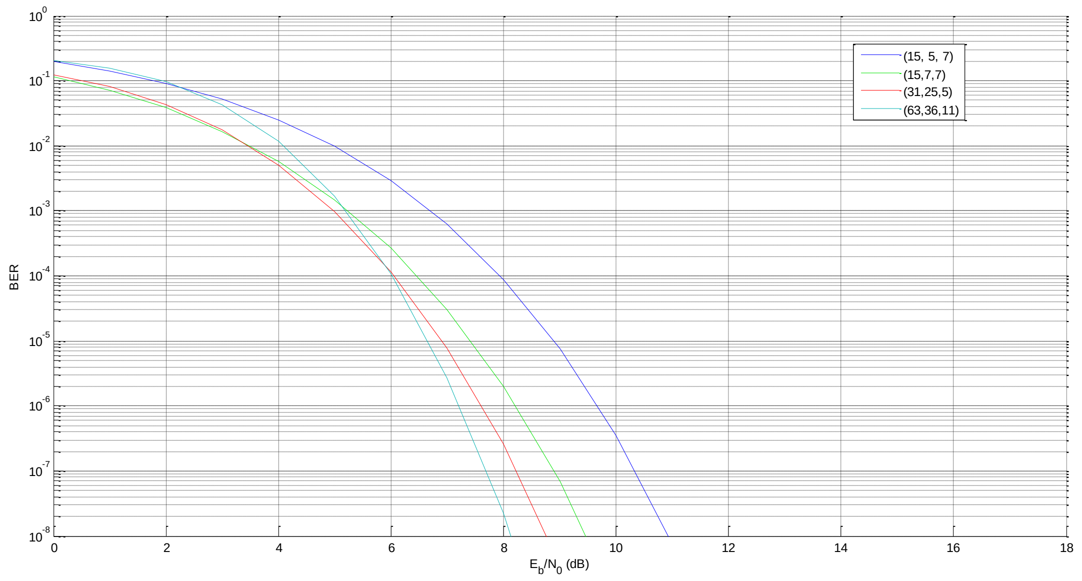


Figure 4.6 different BCH types performance

The Encryption-Decryption Process

To test the encryption process a text file is being encrypted and decrypt to ensure that the AES Rijndael Block cipher is working properly.

The implemented AES is a symmetric block cipher algorithm with 128-bit key and 128-bit block size.

The input data is a text data this data is shown as it is in the text editor

Data compression or Source coding is the process of encoding information using fewer bits than an unencoded representation would use by removing the redundancy and we can remove it until limit defined as entropy.

The 128-bit symmetric cipher key used is

(2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C)$_{HEX}$

The input for the AES RJ block cipher is broken into 4 by 4 blocks of 8-bit symbols as follows in decimal representation

68	32	112	115	32	83	99	111	103	32	32	99
97	99	114	105	111	111	101	100	32	116	112	101
116	111	101	111	114	117	32	105	105	104	114	115
97	109	115	110	32	114	99	110	115	101	111	115
32	101	100	32	111	116	32	110	101	32	115	104
111	110	105	105	114	105	117	103	119	98	32	97
102	99	110	110	109	111	115	32	101	105	32	110
32	111	103	102	97	110	105	102	114	116	116	32

97	110	111	32	114	110	105	119	100	101	32	111
110	101	100	114	101	116	111	111	32	32	114	118
32	110	101	101	115	97	110	117	117	98	101	105
117	99	100	112	101	116	32	108	115	121	109	110
103	101	100	97	32	32	99	114	118	116	110	32
32	32	117	110	97	119	97	101	101	32	116	108
116	114	110	99	110	101	110	109	32	32	105	105
104	101	100	121	100	32	32	111	105	117	108	109
105	101	101	115	116	121	0	0				
116	102	100	32	114	46	0	0				
32	105	32	101	111	0	0	0				
100	110	97	110	112	0	0	0				

The data after the encryption process will be like this and it has a range from 0 to 255 $(0\text{-}FF)_{HEX}$

126	140	15	168	86	178	49	229	9	107	64	6
211	84	224	2	89	28	58	56	74	251	183	174
254	61	70	118	248	191	53	182	66	52	128	12
94	217	94	99	16	148	204	43	187	230	54	75
107	193	153	223	237	30	224	146	140	117	15	134
153	168	53	238	78	186	185	235	255	158	117	58
48	169	56	223	213	222	6	60	68	233	202	15
119	60	43	239	22	99	18	68	100	41	215	253
195	39	170	200	15	152	228	164	223	155	55	174
85	192	78	138	75	212	157	83	106	37	96	132
4	123	54	92	68	194	51	232	158	235	156	44
162	196	249	65	168	2	162	34	92	112	125	163
86	69	213	147	42	176	97	129	191	23	141	148
248	140	197	58	151	25	111	141	139	55	15	210
137	16	31	223	237	214	183	251	167	32	104	96
226	66	42	199	229	134	131	11	248	59	3	215
143	107	36	65	14	232	142	142				
126	101	21	126	199	115	135	183				
149	54	37	126	170	80	161	55				
239	93	28	254	168	207	148	204				

After the above process the text data that appears in the file editor

```
~Óþ^ŒT=ÙàF^¨vcVYø²¿”1:5Ìå8¶+JB»kû4æ@·€6®Kk™0wÁ¨©<™58+ßîßïíNÕ-ºÞcà¹’ë<DŒÿDdužé)uÊ
×†:ýÃ¢'À{ÄªN6ùÈŠ\AKD¨˜ÔÂä•3¢¤Sè"ßjž\›%ëp7`œ}®„,£Vø‰âEŒBÕÅ*“:ßÇ*—íå°Ö†ao·ƒ••û¿‹§ø7
;•h————————————————————”Ò`×•~•ïke6]$%A~~þÇª¨èsPÏŽ‡¡”Ž·7Ì
```

The Steganography Process [*]

To increase the data security and protection steganography is used to conceal the encrypted data within another carrier data file usually true colored losses encoded data file commonly bitmap indexed images, the steganography is based on least significant bit embedding LSB and it is tested to an encrypted data file as follows :

The Data text File to be AES encrypted first

- Input Data : Implementation of a digital communication System

AES encryption with (2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C)$_{HEX}$ cipher key

- Encrypted Data : §•¦<]-
C£Søfs•ÕúeíþG

Embedded Data within bitmap image

- Data Stenography

The Original

The Embedded

The steganography generated key is function of Encrypted Data bits and it is : (2 00) $_{HEX}$

- Extracted Data : §•¦<]-
C£Søfs•ÕúeíþG

- Decrypted Data : Implementation of a digital communication System

The test is done to the both two cryptography and steganography systems and the original data is recovered successfully as shown above.

[*] This process is not essential in the communication system but its added to perform higher protection for data storage purposes

System Implementation

After each function is tested, the whole system functions have to work with each other to perform the total system operations.

The transmitter of the system has a data file as an input, this file is the source coder [***bwtenc***] , [***rundec***]or [***huffenc***] input, the output of this encoder is the compressed data file and it is passed to AES encryption block[***AES( )***] this block take the compressed binary file as a plain text and ciphers it after that the cipher text is encoded with FEC BCH (15, 5) encoder [***bchen***] and it is modulated and transmitted over AWGN channel the channel function performs the last operation to the output BCH encoded file. For storage purposes the file can be protected with another protection layer which is the steganography and it is done by the steganography system that is implemented to perform that.

The following diagram shows the steps described above

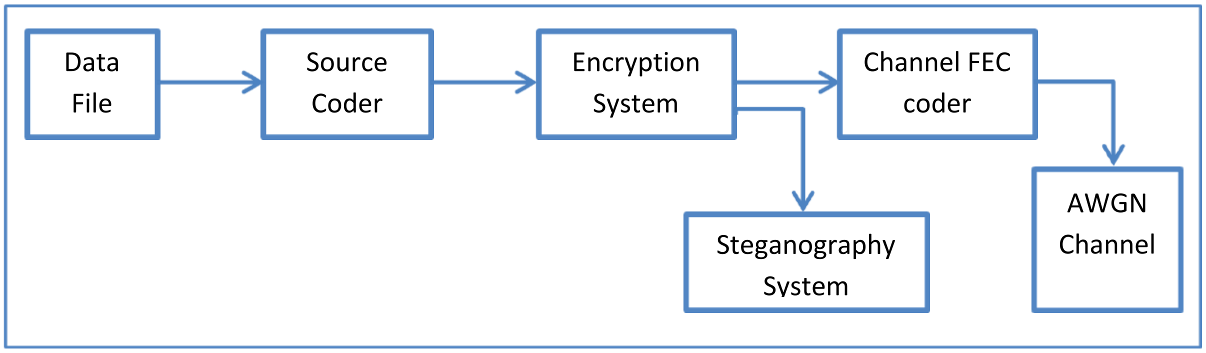


Figure 5   The Implemented System

At the receiving side the data file is obtained by carrying out the reverse operations which are the FEC BCH decoding [***bchde***] then the decoded information is passed to the decryption system to do the inverse operation of AES ciphering [***iAES***] then the data is ready to be decompressed by the source decoder [***bwtdec***], [***huffdec***],or [***rundec***]
If the data is concealed within a carrier file by the steganography system the data could be obtained by the inverse operation.The oerations above described in the next figure :

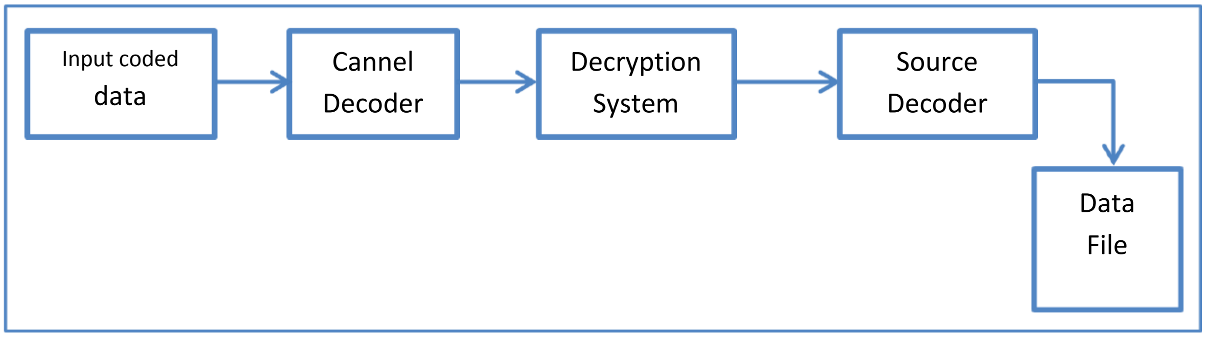


Figure 4.8 The implemented decoding system

System Implementation Test

In this part we need to ensure that our system work properly so we will select a text file that passed through our implementation.

Original file

Size of the file is [1004 byte]

Applying BWT compression and the output file's size is [564 byte]

Now applying Rijndeal encryption to the source codded file [file.BWT] and output file size is the same as the input.

The output of the Rijndael encryption [data.AES] is pass to BCH encoding, the output of BCH [bch.BCH] size is [1120 byte], and this illustrate why source coding is needed sine the BCH output is the same as the original file , the size of the original file and the output of bch is the same approximately with fewer bytes, but with capability to perform the forward error correction

Now [bch.BCH] is needed to be passed through AWGN channel and the most important parameter here is the BER. If we select the 8-psk modulation and by changing the SNR the results of the file reception are illustrated in the table

Signal SNR	13	12	10	9
# of errors	109	233	626	856
BER	0.0123	0.02	0.0704	0.0962
File Recovery	Successfully recovered	Failed	Failed	Failed

The system can recover[*] the signal only if the SNR greater or equal 13 for this file as shown in the table, and the recovered signal is exactly same the original signal [1004 byte] .

The BCH decoder failed to recover the signal at SNR level of 9 since we have in average 7 errors per each block which are over the decoder correction capability.

Finally the system is implemented successfully and it is tested for different file types as it illustrated previously.

[*] It is needed to obtain the signal after BCH decoder with no errors

The MATLAB Functions

The Source Coding and Decoding Functions

Data Analysis with Entropy Function

The Function source code

```
function [H]=ent(a)
if strcmp(a,'im')
[filename1] = uigetfile('*.*', 'choose image');
A=imread(filename1);
A=reshape(A,[],1);
end
if strcmp(a,'txt')
[filename] = uigetfile('*.txt', 'choose a text file');
file_open=fopen(filename,'r');
file_read=fread(file_open,'uint8');
fclose(file_open);
A=file_read;
end
if strcmp(a,'wav')
[filename1] = uigetfile('*.wav', 'choose wav file');
A=wavread(filename1);
end
freq=histc(A,unique(A));
p=freq./sum(freq);
pin=1./p;
plog=log2(pin);
pk=p.*plog;
H=sum(pk);

message=['File Entropy :' num2str(H) 'bit/symbol'];
disp(message);
```

Calling code: **ent(‘ file type’)**

Inputs: Data File , File Type text ,image or wav file

Output: The Entropy of the File in bits/symbol

The function performs the entropy calculations by finding the unique symbols in a given file and then the frequency of occurrence of each symbol then the probability could be found for each symbol and then by applying the entropy equation the function returns the parameter H which is the file entropy, the entropy represents the lower limit of the source coding output average code length.

Run-Length Encoder and Decoder

The Function source code

```
function []=runenc
name=uigetfile('*.*');
file_open=fopen(name,'r');
file_read=fread(file_open,'uint8');
fclose(file_open);
a=file_read;
a=reshape(a,[],1);
code=a;
count=1;k=1;i=1;  % Run Length Encoder
while i~=length(code)
    for j=i+1:length(code)
        if code(i)==code(j)
            count=count+1;
        end
        if code(i)~=code(j)
            Run_Len(k)=code(i);
            Run_Len(k+1)=count;
            k=k+2;
            count=1;
            i=j;
            break
        end
    end
                    if j==length(code);
                         Run_Len(k)=code(i);
            Run_Len(k+1)=count;
            k=k+2;
            count=0;
            i=j;
                        break
                    end
end
file=fopen('File.RUN','w');
fwrite(file,Run_Len,'ubit8');
fclose(file);
```

Calling code: **runenc**

Inputs: Data File

Output: Run-Length Encoded File (**.RUN*)

The function reshapes the data file to a vector of symbols then it holds the symbol and counts the repetitions of that symbol after its position it takes another symbol when the different symbol is found.

If there is no reputations in a given data file the size of this file will be increased as a result.

```
function []=rundec
file_open=fopen('File.run','r');
file_read=fread(file_open,'ubit8');
fclose(file_open);
run_code=file_read;
data=[];
pos=1;k=1;j=1;
for i=1:length(run_code)
    if mod(i,2)~=0
          values(k)=run_code(i);
          k=k+1;
    end
    if mod(i,2)==0
        runs(j)=run_code(i);
        j=j+1;
    end
end
    i=1;k=1;r=1;
    while i~=sum(runs)+1
       data(i)=values(k);
      if runs(r)~=1
          for j=0:runs(r)
            data(i+j)=values(k);

          end
             i=i+runs(r); k=k+1;r=r+1;
             else i=i+1;k=k+1;r=r+1;
      end
    end

file=fopen('file.txt','w');
fwrite(file,char(data)','ubit8');
fclose(file);
```

Calling code: **rundec**

Inputs: File.run

Output: the original data file (file.txt)

The function separates the repetitions and the data into two array to recover the original data and put it into output file.

Huffman Encoder & Decoder

The function source code

Huffman Encoder

```
function []=huffenc(type)
name=uigetfile('*.*');
if strcmp(type,'txt')==1
file_open=fopen(name,'r');
file_read=fread(file_open,'uint8');
fclose(file_open);
a=file_read;
end
if strcmp(type,'wav')==1;
    a=wavread(name);
end
symbols=unique(a);
freq=histc(a,unique(a));
p=freq./sum(freq);
dict = huffmandict(symbols,p); % Create the dictionary.
hcode = huffmanenco(a,dict)'; % Encode the data.
file_Huff=fopen('File.HUFF','w');
fwrite(file_Huff,hcode,'ubit1');
fclose(file_Huff);
if strcmp(type,'wav')==1;
file_Huff=fopen('DictS.HUFF','w');
fwrite(file_Huff,symbols,'double');
fclose(file_Huff);
end
if strcmp(type,'txt')==1;
file_Huff=fopen('DictS.HUFF','w');
fwrite(file_Huff,symbols,'ubit8');
fclose(file_Huff);
end
power=1+ceil(log2(max(freq)));
file_Huff=fopen('DictF.HUFF','w');
command=['ubit' int2str(power)];
fwrite(file_Huff,freq,command);
fclose(file_Huff);
powerL=1+ceil(log2(length(hcode)));
commandL=['ubit' int2str(powerL)];
Len=length(hcode);
file_Huff=fopen('Len.HUFF','w');
fwrite(file_Huff,Len,commandL);
fclose(file_Huff);
sym=length(symbols);
key=[power,sym,powerL];
file_power=fopen('Key.HUFF','w');
fwrite(file_power,key,'ubit8');
fclose(file_power);
AVL=0;
for i=1:length(p)
    AVL=AVL+p(i)*length(dict{i,2});
end
message=['AVL : ' num2str(AVL) 'bits/symbol'];
disp(message)
```

Calling code: **huffenc( )**

Inputs: data file, type of file

Output: Huffman compressed file with header file contains the source symbols frequencies and length of data ,The encoding and decoding is done as it was shown in chapter one

Huffman Decoder

```
function []=huffdec(type)
file_open=fopen('Key.HUFF','r');
file_read=fread(file_open,'ubit8');
fclose(file_open);
[key]=file_read;
power=key(1);
sym=key(2);
powerL=key(3);
if strcmp(type,'wav')==1;
file_open=fopen('DictS.HUFF','r');
file_read=fread(file_open,'ubit8');
fclose(file_open);
symbols(1:sym)=file_read(1:sym);
end
if strcmp(type,'wav')==1;
file_open=fopen('DictS.HUFF','r');
file_read=fread(file_open,'double');
fclose(file_open);
symbols(1:sym)=file_read(1:sym);
end
file_open=fopen('DictF.HUFF','r');
command=['ubit' int2str(power)];
file_read=fread(file_open,command);
fclose(file_open);
freq(1:sym)=file_read(1:sym);
file_open=fopen('Len.HUFF','r');
commandL=['ubit' int2str(powerL)];
file_read=fread(file_open,commandL);
fclose(file_open);
Len=file_read;
file_open=fopen('File.HUFF','r');
file_read=fread(file_open,'ubit1');
fclose(file_open);
hcode(1:Len)=file_read(1:Len);
p(1:sym)=freq(1:sym)/sum(freq);
dict = huffmandict(symbols,p); % Create the dictionary.
dhsig = huffmandeco(hcode,dict); % Decode the code.
if strcmp(type,'wav')==1;
wavwrite(dhsig,44100,'a.wav');
end
if strcmp(type,'txt')==1
file=fopen('file.txt','w');
fwrite(file,char(dhsig)','ubit8');
fclose(file);
end
```

Calling code: **huffdnc( )** *Inputs*: huffman encoded file (*.huff),header files *Output*: data File

BWT encoder & decoder

BWT Encoder

```
function []=bwtenc
name=uigetfile('*.*');
file_open=fopen(name,'r');
file_read=fread(file_open,'uint8');
fclose(file_open);
a=file_read;
Max=max(histc(a,unique(a)));
L=zeros(length(a),1);
x=a;
BWT=zeros(length(a),Max+1);
for i=1:length(a)
L(i)=x(length(a));
x(:,1)=circshift( x,-1);
BWT(i,1:Max)=x(1:Max);
BWT(i,Max+1)=x(length(a));
end
BWT=sortrows(BWT);
for i=1:length(a)
    if BWT(i,1:Max)==transpose(a(1:Max));
        Primary_index=i;
    end
end
%      M  T  F
symbols=unique(a);
dicti=symbols;
MTF_code=zeros(1,length(BWT),'single');

for i=1:length(BWT)  % rearrange the dictionary
    for j=1:length(dicti)
        if BWT(i,Max+1)==dicti(j);
            MTF_code(i)=j-1;
            temp=dicti(j);s=j;
           while s~=1
               dicti(s)=dicti(s-1);
               s=s-1;
           end
           dicti(1)=temp;
        end
    end
end
MTF_symbols=unique(MTF_code);
MTF_freq=histc(MTF_code,unique(MTF_code));

MTF_P(1:length(MTF_symbols))=MTF_freq(1:length(MTF_symbols))/su
m(MTF_freq);
dict = huffmandict(MTF_symbols,MTF_P); % Create the dictionary.
hcode = huffmanenco(MTF_code,dict)'; % Encode the data.
hcode1=hcode';
% BWT output Files
file_Huff=fopen('File.BWT','w');
fwrite(file_Huff,hcode1,'ubit1');
fclose(file_Huff);
```

BWT Encoder con.

```
powerL=1+ceil(log2(length(hcode)));
commandL=['ubit' int2str(powerL)];
Length_index=[length(hcode),Primary_index];
file_Huff=fopen('Length.BWT','w');
fwrite(file_Huff,Length_index,commandL);
fclose(file_Huff);

powerMT=1+ceil(log2(max(MTF_symbols)));
commandMT=['ubit' int2str(powerMT)];
file_Huff=fopen('MTFSYM.MTF','w');
fwrite(file_Huff,MTF_symbols,commandMT);
fclose(file_Huff);

power=1+ceil(log2(max(MTF_freq)));
sym_no=length(MTF_freq);
key=[power,sym_no,powerL,powerMT];
file_power=fopen('key','w');
fwrite(file_power,key,'ubit8');
fclose(file_power);

file_Huff=fopen('MTF.MTF','w');
command=['ubit' int2str(power)];
fwrite(file_Huff,MTF_freq,command);
fclose(file_Huff);

file_Huff=fopen('Dict.MTF','w');
fwrite(file_Huff,symbols,'ubit8');
fclose(file_Huff);

AVL=0;
for i=1:length(MTF_P)
    AVL=AVL+MTF_P(i)*length(dict{i,2});
end
```

Calling code: **bwtenc**

Inputs: data file

Output: compressed file (*.BWT), header files contains MTF symbols & frequencies, Huffman symbols, primary BWT index, source symbols.

BWT Decoder

```
3function []=bwtdec
file_open=fopen('key','r');
file_read=fread(file_open,'ubit8');
fclose(file_open);
[key1]=file_read;
power1=key1(1,1);sym_no1=key1(2,1);powerL1=key1(3,1);powerMT1=k
ey1(4,1);
file_open=fopen('Dict.MTF','r');
file_read=fread(file_open,'ubit8');
fclose(file_open);
[symbols1]=file_read;
file_open=fopen('Length.BWT','r');
commandL1=['ubit' int2str(powerL1)];
file_read=fread(file_open,commandL1);
fclose(file_open);
Length_huffcode=file_read(1);
Primary_Index=file_read(2);
file_open=fopen('File.BWT','r');
file_read=fread(file_open,'ubit1');
fclose(file_open);
hcode1(1:Length_huffcode)=file_read(1:Length_huffcode);
file_open=fopen('MTF.MTF','r');
command1=['ubit' int2str(power1)];
file_read=fread(file_open,command1);
fclose(file_open);
MTF_freq1(1:sym_no1)=file_read(1:sym_no1);
file_open=fopen('MTFSYM.MTF','r');
commandMT1=['ubit' int2str(powerMT1)];
file_read=fread(file_open,commandMT1);
fclose(file_open);
MTF_symbols1(1:sym_no1)=file_read(1:sym_no1);
MTF_P1(1:sym_no1)=MTF_freq1(1:sym_no1)/sum(MTF_freq1);
%MTF_symbols1=unique(MTF_code);
dict1 = huffmandict(MTF_symbols1,MTF_P1); % Create the
dictionary.
dhsig = huffmandeco(hcode1,dict1); % Decode the code.

IMTF_code=zeros(length(dhsig),1);
for i=1:length(dhsig)
    IMTF_code(i)=symbols1(dhsig(i)+1);
    s=dhsig(i)+1;
           while s~=1
               symbols1(s)=symbols1(s-1);
               s=s-1;
           end
           symbols1(1)=IMTF_code(i);
end
% BWT decoding
L=IMTF_code;
F=zeros(length(L),2);
F(:,1)=sortrows(L);
T=zeros(length(L),1);
```

```
for i=1:length(L)
    for j=1:length(F)
        if (L(i)==F(j))&& (F(j,2)==0)
            T(i)=j;
            F(j,2)=1;
            break;
        end
    end
end

S=zeros(length(L),1);
% first step
S(length(L))=L((Primary_Index));
X=T(Primary_Index);
for i=1:length(L)-1
    S(length(L)-i)=L(X);
    X=T(X);
end
file_power=fopen('file.txt','w');
fwrite(file_power,char(S)','ubit8');
fclose(file_power);
message=['AVL : ' num2str(AVL) 'bits/symbol'];
disp(message)
```

Calling code: **bwtdec**

Inputs: compressed file (*.BWT), header files

Output: original Data File

The encoding using BWT, MTF, and Huffman is discussed deeply in chapter one with the decoding process .

This implementation has a good memory utilization for the process of storing the BW transformed file since all permutations in done at separated variable and it is stored according to the maximum frequency of occurrence of certain symbol so BWT matrix has a reduced size from data length by data length to maximum frequency plus one square. As a result, the block size increased and the compression becomes more efficient.

The Channel Coding and Decoding Function

BCH Encoder

```
function []=bchen
name=uigetfile('*.*');
file_open=fopen(name,'r');
file_read=fread(file_open,'ubit1');
fclose(file_open);
a=file_read;
n=input('Enter n: ');
k=input('Enter k: ');
P=input('Enter Generator polynomial: ');
G=zeros(k,n);
temp=zeros(1,n);
temp(1,1:n-k+1)=P(1:length(P));
G(1,1:n)=temp(1:n);
for i=2:k
    temp = circshift(temp, [0, 1]);
    G(i,1:n)=temp(1:n);
end
G=rem(abs(rref(G)),2);
hh=length(a);
for i=1:k-rem(hh ,k)
    a(hh+i)=0;
end
j=1;m=k;
jjj=length(a);
output=zeros(1,jjj+jjj*10/k);
for i=1:jjj/k
input1(j:m)=a(j:m);
j=j+k;
m=m+k;
end
f=1;h=k;
x=1;x1=n;
for i=1:(jjj/k)-1
c=input1(f:h)*G;
nn=1;
for l=x:x1
    output(l)=c(nn);
    nn=nn+1;
end
x=x+n;
x1=x1+n;
f=f+k;
h=h+k;
end
output=rem(output,2);
file=fopen('bch.BCH','w');
fwrite(file,output,'ubit1');
fclose(file);
```

Calling code: **bchen** *Inputs*: the binary file that is encoded by ‘huffenc ‘ or ‘bwtenc’ , *Output*: ‘bch.BCH’

BCH Decoder

```
function []=bchde
name=uigetfile('*.*');
file_open=fopen(name,'r');
file_read=fread(file_open,'ubit1');
fclose(file_open);
output=file_read;
n=input('Enter n: ');
k=input('Enter k: ');
P=input('Enter Generator polynomial: ');
G=zeros(k,n);
temp=zeros(1,n);
temp(1,1:n-k+1)=P(1:length(P));
G(1,1:n)=temp(1:n);
for i=2:k
    temp = circshift(temp, [0, 1]);
    G(i,1:n)=temp(1:n);
end
G=rem(abs(rref(G)),2);

H=zeros(n-k,n);
j=1;
for i=k+1:n
    H(j,k+j)=1;
    j=j+1;
end
parity=G(:,k+1:n);
H(:,1:k)=parity';
t = syndtable(H);
j=1;m=n;u=1;
for i=1:length(output)/n
r(1:n)=output(j:m);
s = rem(r * H', 2);
err = bi2de(fliplr(s));
err_loc = t(err + 1, :);
ccode = rem(err_loc + r, 2);
recoverd(1,u:u+k-1)=ccode(1,1:k);
u=u+k;
j=j+n;
m=m+n;
end
file=fopen('bchde.BCH','w');
fwrite(file,recoverd,'ubit1');
fclose(file);
```

Calling code: **bchde** , *Inputs*: the encoded file ‘bch.BCH’ , *Output*: binary file ’ bchde.BCH’ that will be delivered to bwtdec or huffdec functions.

The Encryption Functions

AES Rijndael

Rijndael encryption

```
function []=AES(a)
 name=uigetfile('*.*');
file_open=fopen(name,'r');
file_read=fread(file_open,'ubit8');
fclose(file_open);
data=file_read;

if strcmp(a,'1')==1
file_open=fopen(name,'r');
file_read=fread(file_open,'ubit1');
fclose(file_open);
data=file_read;
end
cipherkey=[43 40 171 9;126 174 247 207;21 210 21 79; 22 166 136
60;];

if strcmp(a,'1')==1
data=reshape(data,8,[]);
data=data';
data=bi2de(data,'left-msb');
end

k=16-(rem(length(data),16));
j=length(data);

    for i=1:k
        data(j+i)=0;
    end

enc_data=zeros(length(data),1);
l=1;
for i=1:length(data)/16
    enc_data(l:l+15)=rjenc(cipherkey,data(l:l+15));
    l=l+16;
end
 file=fopen('data.AES','w');
fwrite(file,enc_data,'ubit8');
fclose(file);
 if strcmp(a,'1')==1
enc_data=de2bi(enc_data,'left-msb',8);
enc_data=reshape(enc_data',[],1);
file=fopen('data.AES','w');
fwrite(file,enc_data,'ubit1');
fclose(file);
end

end
```

This function needs another function which is [***rjenc***] to perform the Rj. encryption algorithm

Rijndael Algorithm

```
function enc_data=rjenc(cipherkey,state)
%s-box inilization
sBOX={...
'63' '7c' '77' '7b' 'f2' '6b' '6f' 'c5' '30' '01' '67' '2b' 'fe' 'd7' 'ab' '76';
'ca' '82' 'c9' '7d' 'fa' '59' '47' 'f0' 'ad' 'd4' 'a2' 'af' '9c' 'a4' '72' 'c0';
'b7' 'fd' '93' '26' '36' '3f' 'f7' 'cc' '34' 'a5' 'e5' 'f1' '71' 'd8' '31' '15';
'04' 'c7' '23' 'c3' '18' '96' '05' '9a' '07' '12' '80' 'e2' 'eb' '27' 'b2' '75';
'09' '83' '2c' '1a' '1b' '6e' '5a' 'a0' '52' '3b' 'd6' 'b3' '29' 'e3' '2f' '84';
'53' 'd1' '00' 'ed' '20' 'fc' 'b1' '5b' '6a' 'cb' 'be' '39' '4a' '4c' '58' 'cf';
'd0' 'ef' 'aa' 'fb' '43' '4d' '33' '85' '45' 'f9' '02' '7f' '50' '3c' '9f' 'a8';
'51' 'a3' '40' '8f' '92' '9d' '38' 'f5' 'bc' 'b6' 'da' '21' '10' 'ff' 'f3' 'd2';
'cd' '0c' '13' 'ec' '5f' '97' '44' '17' 'c4' 'a7' '7e' '3d' '64' '5d' '19' '73';
'60' '81' '4f' 'dc' '22' '2a' '90' '88' '46' 'ee' 'b8' '14' 'de' '5e' '0b' 'db';
'e0' '32' '3a' '0a' '49' '06' '24' '5c' 'c2' 'd3' 'ac' '62' '91' '95' 'e4' '79';
'e7' 'c8' '37' '6d' '8d' 'd5' '4e' 'a9' '6c' '56' 'f4' 'ea' '65' '7a' 'ae' '08';
'ba' '78' '25' '2e' '1c' 'a6' 'b4' 'c6' 'e8' 'dd' '74' '1f' '4b' 'bd' '8b' '8a';
'70' '3e' 'b5' '66' '48' '03' 'f6' '0e' '61' '35' '57' 'b9' '86' 'c1' '1d' '9e';
'e1' 'f8' '98' '11' '69' 'd9' '8e' '94' '9b' '1e' '87' 'e9' 'ce' '55' '28' 'df';
'8c' 'a1' '89' '0d' 'bf' 'e6' '42' '68' '41' '99' '2d' '0f' 'b0' '54' 'bb' '16';};
sBOXd=zeros(16,16);% initilization of 16x16 decimal s-box
for i=1:16
    for j=1:16
        sBOXd(i,j)=hex2dec(sBOX{i,j});
    end
end
%---- KEY SCHEDULE ------
matrix inilizatin
state=reshape(state,4,4);  % 4x4 Data matrix=state
cipher_key=reshape(cipherkey,4,4);
rcon=zeros(4,10);                              % initlization of
RCON matrix
rcon(1,:)=[1 2 4 8 16 32 64 128 27 54;];      % initlization of
RCON matrix
round_key=zeros(4,44);
round_key(1:4,1:4)=cipher_key(1:4,1:4);
level=5;pre_level=level-4;L=0;                  %W in file
while(level~=45)    % all levels=columns to 44+1

 for i=1:4      % total rows =4
    if
(level==5)||(level==9)||(level==13)||(level==17)||(level==21)||(level=
=25)||(level==29)||(level==33)||(level==37)||(level==41)
        if i==1
            round_key(:,level)=circshift(round_key(:,level-
1),3); % circular shift onece at level
            L=L+1;  % rcon index increment
        end
        s_pointer=de2bi(round_key(i,level),8,'left-msb');
        r=bi2de(s_pointer(1:4),'left-msb');
        c=bi2de(s_pointer(5:8),'left-msb');
        round_key(i,level)=sBOXd(r+1,c+1);
        round_key(i,level)=bi2de(xor((de2bi(rcon(i,L),8,'left-
msb')),(de2bi(round_key(i,level),8,'left-msb'))),'left-msb');
    else
        round_key(i,level)=round_key(i,level-1);  % if it is
not from above values 5,9,13 ...
    end

round_key(i,level)=bi2de(xor((de2bi(round_key(i,pre_level),8,'l
eft-msb')),(de2bi(round_key(i,level),8,'left-msb'))),'left-
msb');
 end
level=level+1;
  pre_level=pre_level+1;
end
```

```
  %----- ENCRYPTION ---------
temp_column=zeros(4,1);
mul_temp=zeros(4,1);
mix_colum=[02 03 1 1;1 2 3 1;1 1 2 3;3 1 1 2];
enc=zeros(4,4);
enc_temp=zeros(4,4);
enc_temp2=zeros(4,4);
xorAns=zeros(4,1);
for m=1:4   %INITIAL ROUND ENCRYPTION
   for i=1:4
    enc(i,m)=bi2de(xor((de2bi(state(i,m),8,'left-msb')),
(de2bi(round_key(i,m),8,'left-msb'))),'left-msb');
   end
end
for R=1:10 % total rounds
    for i=1:4   %cloumns
        for j=1:4   % rows
        s_pointer=de2bi(enc(i,j),8,'left-msb');
        r=bi2de(s_pointer(1:4),'left-msb');
        c=bi2de(s_pointer(5:8),'left-msb');
        enc(i,j)=sBOXd(r+1,c+1);
        end
    end
    enc_temp(1:4,1:4)=enc(1:4,1:4); % transfer enc to enc_temp
for shifting
    for f=1:4  % shift Rows
      enc_temp(f,:)=circshift(enc_temp(f,:),[0 -(f-1)]); %
circular shift for rows
    end
    enc(1:4,1:4)=enc_temp(:,:);  % return the enc_temp to Enc
    %----Mix Columns -------
    if R~=10 % Last Round dont have mix column step
for t=1:4
   for i=1:4
        temp_column=enc(:,i);
        for j=1:4
            for k=1:4
            mul_temp(k)=rjmul(temp_column(k,1),mix_colum(j,k));
            end
            mul_xor1=bi2de(xor((de2bi(mul_temp(1),8,'left-
msb')),(de2bi(mul_temp(2),8,'left-msb'))),'left-msb'); % RJ.
Filed mul. of mix.col.mat&enc
            mul_xor2=bi2de(xor((de2bi(mul_temp(3),8,'left-
msb')),(de2bi(mul_temp(4),8,'left-msb'))),'left-msb'); % RJ.
Filed mul .of mix.col.mat&enc
            xorAns(j,1)=bi2de(xor((de2bi(mul_xor1,8,'left-
msb')),(de2bi(mul_xor2,8,'left-msb'))),'left-msb'); % Xor the
mul. results
        end
        enc_temp(:,i)=xorAns;
   end
end
    end
round_key_temp(1:4,1:4)=round_key(1:4,4*R+1:4*R+4);
    for m=1:4
        for i=1:4
        enc_temp2(i,m)=bi2de(xor((de2bi(enc_temp(i,m),8,'left-
msb')),(de2bi(round_key_temp(i,m),8,'left-msb'))),'left-msb');
        end
    end
    enc(1:4,1:4)=enc_temp2(:,:);  % return the enc_temp to Enc
end
enc_data=reshape(enc,[],1);
```

Rijndael Field Multiplication , GF(8)

```
function [rj]=rjmul(a,b)
apoly = gf(de2bi(a ,'left-msb'),8,283);
bpoly = gf(de2bi(b ,'left-msb'),8,283);
xpoly = gf([1 0 0 0 1 1 0 1 1],8,283);
cpoly = conv(apoly,bpoly);
[a2,remd] = deconv(cpoly,xpoly);
rj=bi2de(double(remd.x),'left-msb');
```

These three function['AES' , 'rjenc' , 'rjmul'] are working together to perform the encryption process

Calling code: **AES('Type')** *Input*: data file *Output*: encrypted file (*.AES)

Rijndael Decryption

```
function []=iAES
name=uigetfile('*.*');
file_open=fopen(name,'r');
file_read=fread(file_open,'ubit1');
fclose(file_open);
enc_data1=file_read;
enc_data1=enc_data1';
enc_data1=reshape(enc_data1,8,[]);
enc_data1=enc_data1';
enc_data1=bi2de(enc_data1,'left-msb');
enc_data=enc_data1;
%k=length(enc_data1)-(rem(length(enc_data1),16));
%enc_data=enc_data1(1:k);


cipherkey=[43 40 171 9;126 174 247 207;21 210 21 79; 22 166 136
60;];

data=zeros(length(enc_data),1);
l=1;
for i=1:length(data)/16
    data(l:l+15)=rjdec(cipherkey,enc_data(l:l+15));
    l=l+16;
end

data=de2bi(data,'left-msb',8);
data=data';
data=reshape(data,1,[]);
file=fopen('data1.iAES','w');
fwrite(file,data,'ubit1');
```

also in decryption process we need two function to carry out the process

```
function data=rjdec(cipherkey,enc_data)
%s-box inilization
sBOX={...
'63' '7c' '77' '7b' 'f2' '6b' '6f' 'c5' '30' '01' '67' '2b' 'fe' 'd7' 'ab' '76';
'ca' '82' 'c9' '7d' 'fa' '59' '47' 'f0' 'ad' 'd4' 'a2' 'af' '9c' 'a4' '72' 'c0';
'b7' 'fd' '93' '26' '36' '3f' 'f7' 'cc' '34' 'a5' 'e5' 'f1' '71' 'd8' '31' '15';
'04' 'c7' '23' 'c3' '18' '96' '05' '9a' '07' '12' '80' 'e2' 'eb' '27' 'b2' '75';
'09' '83' '2c' '1a' '1b' '6e' '5a' 'a0' '52' '3b' 'd6' 'b3' '29' 'e3' '2f' '84';
'53' 'd1' '00' 'ed' '20' 'fc' 'b1' '5b' '6a' 'cb' 'be' '39' '4a' '4c' '58' 'cf';
'd0' 'ef' 'aa' 'fb' '43' '4d' '33' '85' '45' 'f9' '02' '7f' '50' '3c' '9f' 'a8';
'51' 'a3' '40' '8f' '92' '9d' '38' 'f5' 'bc' 'b6' 'da' '21' '10' 'ff' 'f3' 'd2';
'cd' '0c' '13' 'ec' '5f' '97' '44' '17' 'c4' 'a7' '7e' '3d' '64' '5d' '19' '73';
'60' '81' '4f' 'dc' '22' '2a' '90' '88' '46' 'ee' 'b8' '14' 'de' '5e' '0b' 'db';
'e0' '32' '3a' '0a' '49' '06' '24' '5c' 'c2' 'd3' 'ac' '62' '91' '95' 'e4' '79';
'e7' 'c8' '37' '6d' '8d' 'd5' '4e' 'a9' '6c' '56' 'f4' 'ea' '65' '7a' 'ae' '08';
'ba' '78' '25' '2e' '1c' 'a6' 'b4' 'c6' 'e8' 'dd' '74' '1f' '4b' 'bd' '8b' '8a';
'70' '3e' 'b5' '66' '48' '03' 'f6' '0e' '61' '35' '57' 'b9' '86' 'c1' '1d' '9e';
'e1' 'f8' '98' '11' '69' 'd9' '8e' '94' '9b' '1e' '87' 'e9' 'ce' '55' '28' 'df';
'8c' 'a1' '89' '0d' 'bf' 'e6' '42' '68' '41' '99' '2d' '0f' 'b0' '54' 'bb' '16';};
sBOXd=zeros(16,16);% initilization of 16x16 decimal s-box
for i=1:16
    for j=1:16
        sBOXd(i,j)=hex2dec(sBOX{i,j});
    end
end
 %s-box inverse inilization
IsBOX={...
'52' '09' '6a' 'd5' '30' '36' 'a5' '38' 'bf' '40' 'a3' '9e' '81' 'f3' 'd7' 'fb';
'7c' 'e3' '39' '82' '9b' '2f' 'ff' '87' '34' '8e' '43' '44' 'c4' 'de' 'e9' 'cb';
'54' '7b' '94' '32' 'a6' 'c2' '23' '3d' 'ee' '4c' '95' '0b' '42' 'fa' 'c3' '4e';
'08' '2e' 'a1' '66' '28' 'd9' '24' 'b2' '76' '5b' 'a2' '49' '6d' '8b' 'd1' '25';
'72' 'f8' 'f6' '64' '86' '68' '98' '16' 'd4' 'a4' '5c' 'cc' '5d' '65' 'b6' '92';
'6c' '70' '48' '50' 'fd' 'ed' 'b9' 'da' '5e' '15' '46' '57' 'a7' '8d' '9d' '84';
'90' 'd8' 'ab' '00' '8c' 'bc' 'd3' '0a' 'f7' 'e4' '58' '05' 'b8' 'b3' '45' '06';
'd0' '2c' '1e' '8f' 'ca' '3f' '0f' '02' 'c1' 'af' 'bd' '03' '01' '13' '8a' '6b';
'3a' '91' '11' '41' '4f' '67' 'dc' 'ea' '97' 'f2' 'cf' 'ce' 'f0' 'b4' 'e6' '73';
'96' 'ac' '74' '22' 'e7' 'ad' '35' '85' 'e2' 'f9' '37' 'e8' '1c' '75' 'df' '6e';
'47' 'f1' '1a' '71' '1d' '29' 'c5' '89' '6f' 'b7' '62' '0e' 'aa' '18' 'be' '1b';
'fc' '56' '3e' '4b' 'c6' 'd2' '79' '20' '9a' 'db' 'c0' 'fe' '78' 'cd' '5a' 'f4';
'1f' 'dd' 'a8' '33' '88' '07' 'c7' '31' 'b1' '12' '10' '59' '27' '80' 'ec' '5f';
'60' '51' '7f' 'a9' '19' 'b5' '4a' '0d' '2d' 'e5' '7a' '9f' '93' 'c9' '9c' 'ef';
'a0' 'e0' '3b' '4d' 'ae' '2a' 'f5' 'b0' 'c8' 'eb' 'bb' '3c' '83' '53' '99' '61';
'17' '2b' '04' '7e' 'ba' '77' 'd6' '26' 'e1' '69' '14' '63' '55' '21' '0c' '7d';};
IsBOXd=zeros(16,16);% initilization of 16x16 decimal inv. s-box
for i=1:16
    for j=1:16
        IsBOXd(i,j)=hex2dec(IsBOX{i,j});
    end
end
%---- KEY SCHEDULE ------
cipher_key=reshape(cipherkey,4,4);
rcon=zeros(4,10);                             % initlization of
RCON matrix
rcon(1,:)=[1 2 4 8 16 32 64 128 27 54;];      % initlization of
RCON matrix
round_key=zeros(4,44);
round_key(1:4,1:4)=cipher_key(1:4,1:4);
level=5;pre_level=level-4;L=0;                  %W in file
```

```
while(level~=45)    % all levels=columns to 44+1

 for i=1:4      % total rows =4
    if
(level==5)||(level==9)||(level==13)||(level==17)||(level==21)||
(level==25)||(level==29)||(level==33)||(level==37)||(level==41)
        if i==1
            round_key(:,level)=circshift(round_key(:,level-
1),3); % circular shift onece at level
            L=L+1;  % rcon index increment
        end
        s_pointer=de2bi(round_key(i,level),8,'left-msb');
        r=bi2de(s_pointer(1:4),'left-msb');
        c=bi2de(s_pointer(5:8),'left-msb');
        round_key(i,level)=sBOXd(r+1,c+1);
        round_key(i,level)=bi2de(xor((de2bi(rcon(i,L),8,'left-
msb')),(de2bi(round_key(i,level),8,'left-msb'))),'left-msb');
    else
        round_key(i,level)=round_key(i,level-1);  % if it is
not from above values 5,9,13 ...
    end

round_key(i,level)=bi2de(xor((de2bi(round_key(i,pre_level),8,'l
eft-msb')),(de2bi(round_key(i,level),8,'left-msb'))),'left-
msb');
 end
  level=level+1;
  pre_level=pre_level+1;
end
%------DECRYPTION--------------
temp_column=zeros(4,1);
mul_temp=zeros(4,1);
mix_colum=[14 11 13 9;9 14 11 13;13 9 14 11;11 13 9 14];
%inverse mix_colum
encrypted_data=reshape(enc_data,4,4);
state=zeros(4,4);
enc_temp=zeros(4,4);
enc_temp2=zeros(4,4);
xorAns=zeros(4,1);
enc=encrypted_data;
for h=1:10 % total rounds
    R=11-h;
    enc_temp2(:,:)=enc(:,:);  % return the enc_temp to Enc
    %------ Add Round Key -------
    round_key_temp(1:4,1:4)=round_key(1:4,4*R+1:4*R+4);
    for m=1:4
        for i=1:4
        enc_temp(i,m)=bi2de(xor((de2bi(enc_temp2(i,m),8,'left-
msb')),(de2bi(round_key_temp(i,m),8,'left-msb'))),'left-msb');
        end
    end
    enc(:,:)=enc_temp(:,:);
    %----Mix Columns -------
    if R~=10 % Last Round dont have mix column step
for t=1:4
   for i=1:4
        temp_column=enc(:,i);
        for j=1:4
            for k=1:4

mul_temp(k)=rjmul(temp_column(k,1),mix_colum(j,k));
            end
            mul_xor1=bi2de(xor((de2bi(mul_temp(1),8,'left-
msb')),(de2bi(mul_temp(2),8,'left-msb'))),'left-msb'); % RJ.
Filed mul. of mix.col.mat&enc
            mul_xor2=bi2de(xor((de2bi(mul_temp(3),8,'left-
msb')),(de2bi(mul_temp(4),8,'left-msb'))),'left-msb'); % RJ.
Filed mul .of mix.col.mat&enc
            xorAns(j,1)=bi2de(xor((de2bi(mul_xor1,8,'left-
msb')),(de2bi(mul_xor2,8,'left-msb'))),'left-msb'); % Xor the
mul. results
        end
        enc_temp(:,i)=xorAns;
   end
end
    end
     enc(:,:)=enc_temp(:,:);
     for f=1:4  % shift Rows
      enc(f,:)=circshift(enc(f,:),[0 (f-1)]); % circular shift
for rows
     end
     for i=1:4   %cloumns
        for j=1:4   % rows
        s_pointer=de2bi(enc(i,j),8,'left-msb');
        r=bi2de(s_pointer(1:4),'left-msb');
        c=bi2de(s_pointer(5:8),'left-msb');
        enc(i,j)=IsBOXd(r+1,c+1);
        end
     end
end
for m=1:4 %FINAL STEP
   for i=1:4
    state(i,m)=bi2de(xor((de2bi(enc(i,m),8,'left-
msb')),(de2bi(round_key(i,m),8,'left-msb'))),'left-msb');
   end
end
data=reshape(state,[],1);
```

These three function['iAES' , 'rjdec' , 'rjmul'] that work together to perform the decryption part.

Calling code: i**AES('Type') ,** *Inputs*: encrypted file'data.AES' , *Output*:the decrypted file ('data1.iAES').

Steganography

LSB Steganography encoder

```
function []=textsteg(mode)
[filename1,path] = uigetfile('*.*', 'choose the carrier');
cd(path)
carrierfile=imread(filename1);
[filename] = uigetfile('*.txt', 'choose a file to be
Embedded');
file_open=fopen(filename,'r');
file_read=fread(file_open,'uint8');
fclose(file_open);
text=file_read;clear file_open;clear file_read;
binary_text_bytes=de2bi(text,8);
[tex_L,tex_W]=size(binary_text_bytes);
[c_L,c_W,c_d]=size(carrierfile);
binary_text=reshape(binary_text_bytes,[],1);

for i=1:length(binary_text)
        if mode==8
        pix_bi=de2bi(carrierfile(i),8,'left-msb');
        pix_bi(8)=binary_text(i);        %Least sig. bit 1st
        i=i+1;if i>length(binary_text),break;end
        pix_bi(7)=binary_text(i-1);      %Least sig. bit 2nd
        i=i+1;if i>length(binary_text),break;end
        pix_bi(6)=binary_text(i-2);      %Least sig. bit 3th
        i=i+1;if i>length(binary_text),break;end
        pix_bi(5)=binary_text(i-3);      %Least sig. bit 4th
        i=i+1;if i>length(binary_text),break;end
        pix_bi(4)=binary_text(i-4);      %Least sig. bit 5th
        i=i+1;if i>length(binary_text),break;end
        pix_bi(3)=binary_text(i-5);      %Least sig. bit 6th
        i=i+1;if i>length(binary_text),break;end
        pix_bi(2)=binary_text(i-6);      %Least sig. bit 7th
         i=i+1;if i>length(binary_text),break;end
        pix_bi(1)=binary_text(i-7);      %Least sig. bit 8th
        pix_val=bi2de(pix_bi,'left-msb');
        carrierfile(i-7)=pix_val;
        end
       if mode==7
        pix_bi=de2bi(carrierfile(i),8,'left-msb');
        pix_bi(8)=binary_text(i);        %Least sig. bit 1st
        i=i+1;if i>length(binary_text),break;end
        pix_bi(7)=binary_text(i-1);      %Least sig. bit 2nd
        i=i+1;if i>length(binary_text),break;end
        pix_bi(6)=binary_text(i-2);      %Least sig. bit 3th
        i=i+1;if i>length(binary_text),break;end
        pix_bi(5)=binary_text(i-3);      %Least sig. bit 4th
        i=i+1;if i>length(binary_text),break;end
        pix_bi(4)=binary_text(i-4);      %Least sig. bit 5th
```

```
i=i+1;if i>length(binary_text),break;end
pix_bi(3)=binary_text(i-5);      %Least sig. bit 6th
i=i+1;if i>length(binary_text),break;end
pix_bi(2)=binary_text(i-6);      %Least sig. bit 7th
pix_val=bi2de(pix_bi,'left-msb');
carrierfile(i-6)=pix_val;
 end
if mode==6
pix_bi=de2bi(carrierfile(i),8,'left-msb');
pix_bi(8)=binary_text(i);        %Least sig. bit 1st
i=i+1;if i>length(binary_text),break;end
pix_bi(7)=binary_text(i-1);      %Least sig. bit 2nd
i=i+1;if i>length(binary_text),break;end
pix_bi(6)=binary_text(i-2);      %Least sig. bit 3th
i=i+1;if i>length(binary_text),break;end
pix_bi(5)=binary_text(i-3);      %Least sig. bit 4th
i=i+1;if i>length(binary_text),break;end
pix_bi(4)=binary_text(i-4);      %Least sig. bit 5th
i=i+1;if i>length(binary_text),break;end
pix_bi(3)=binary_text(i-5);      %Least sig. bit 6th
pix_val=bi2de(pix_bi,'left-msb');
carrierfile(i-5)=pix_val;
end
if mode==5
pix_bi=de2bi(carrierfile(i),8,'left-msb');
pix_bi(8)=binary_text(i);        %Least sig. bit 1st
i=i+1;if i>length(binary_text),break;end
pix_bi(7)=binary_text(i-1);      %Least sig. bit 2nd
i=i+1;if i>length(binary_text),break;end
pix_bi(6)=binary_text(i-2);      %Least sig. bit 3th
i=i+1;if i>length(binary_text),break;end
pix_bi(5)=binary_text(i-3);      %Least sig. bit 4th
i=i+1;if i>length(binary_text),break;end
pix_bi(4)=binary_text(i-4);      %Least sig. bit 5th
pix_val=bi2de(pix_bi,'left-msb');
carrierfile(i-4)=pix_val;
end
if mode==4
pix_bi=de2bi(carrierfile(i),8,'left-msb');
pix_bi(8)=binary_text(i);        %Least sig. bit 1st
i=i+1;if i>length(binary_text),break;end
pix_bi(7)=binary_text(i-1);      %Least sig. bit 2nd
i=i+1;if i>length(binary_text),break;end
pix_bi(6)=binary_text(i-2);      %Least sig. bit 3th
i=i+1;if i>length(binary_text),break;end
pix_bi(5)=binary_text(i-3);      %Least sig. bit 4th
pix_val=bi2de(pix_bi,'left-msb');
carrierfile(i-3)=pix_val;
end
if mode==3
pix_bi=de2bi(carrierfile(i),8,'left-msb');
pix_bi(8)=binary_text(i);        %Least sig. bit 1st
i=i+1;if i>length(binary_text),break;end
pix_bi(7)=binary_text(i-1);      %Least sig. bit 2nd
i=i+1;if i>length(binary_text),break;end
pix_bi(6)=binary_text(i-2);      %Least sig. bit 3th
pix_val=bi2de(pix_bi,'left-msb');
carrierfile(i-2)=pix_val;
end
```

```
if mode==2
        pix_bi=de2bi(carrierfile(i),8,'left-msb');
        pix_bi(8)=binary_text(i);        %Least sig. bit 1st

        i=i+1;if i>length(binary_text),break;end
        pix_bi(7)=binary_text(i-1);      %Least sig. bit 2nd

        pix_val=bi2de(pix_bi,'left-msb');
        carrierfile(i-1)=pix_val;
        end
         if mode==1
        pix_bi=de2bi(carrierfile(i),8,'left-msb');
        pix_bi(8)=binary_text(i);        %Least sig. bit 1st

        pix_val=bi2de(pix_bi,'left-msb');
        carrierfile(i)=pix_val;
        end
end

imwrite(carrierfile,'Steg-image-with-text.bmp');
A=num2str(length(binary_text));
out_data=['The Key For Extracting The Data is : ',A];
file=fopen('Key.txt','w');
fwrite(file,out_data,'uint8');
fclose(file);
figure
imshow(carrierfile);clc;message=['DONE With carrier
:',filename1,' to text file :',filename,' the key is :',A ];
disp(message);disp('M.H & M.A');
```

Calling code: *textsteg*(**‘mode’) ,** *Inputs*: the carrier file [picture] and the text file to be embedded , *Output*: ‘'Steg-image-with-text.bmp’

LSB Steganography decoder

```
function []=textstegde(mode)
Key=input('enter the key :');
a=uigetfile('*.bmp','choose stego-image with text');
carrierfile_Em=imread(a);
binary_text_reconstructed=zeros(1,Key);
for i=1:Key
    tt=i;
    if mode==8
        pix_bi=de2bi(carrierfile_Em(i),8,'left-msb');
        binary_text_reconstructed(i)=pix_bi(8);
        i=i+1;if i>Key,break;end   %Least sig. bit 2
        binary_text_reconstructed(i)=pix_bi(7);
        i=i+1;if i>Key,break;end   %Least sig. bit 3
        binary_text_reconstructed(i)=pix_bi(6);
        i=i+1;if i>Key,break;end   %Least sig. bit 4
        binary_text_reconstructed(i)=pix_bi(5);
        i=i+1;if i>Key,break;end   %Least sig. bit 5
        binary_text_reconstructed(i)=pix_bi(4);
        i=i+1;if i>Key,break;end   %Least sig. bit 6
        binary_text_reconstructed(i)=pix_bi(3);
        i=i+1;if i>Key,break;end   %Least sig. bit 7
        binary_text_reconstructed(i)=pix_bi(2);
        i=i+1;if i>Key,break;end   %Least sig. bit 8
        binary_text_reconstructed(i)=pix_bi(1);
    end
```

```
if mode==7
        pix_bi=de2bi(carrierfile_Em(i),8,'left-msb');
        binary_text_reconstructed(i)=pix_bi(8);
        i=i+1;if i>Key,break;end   %Least sig. bit 2
        binary_text_reconstructed(i)=pix_bi(7);
         i=i+1;if i>Key,break;end   %Least sig. bit 3
        binary_text_reconstructed(i)=pix_bi(6);
         i=i+1;if i>Key,break;end   %Least sig. bit 4
        binary_text_reconstructed(i)=pix_bi(5);
         i=i+1;if i>Key,break;end   %Least sig. bit 5
        binary_text_reconstructed(i)=pix_bi(4);
       i=i+1;if i>Key,break;end   %Least sig. bit 6
        binary_text_reconstructed(i)=pix_bi(3);
        i=i+1;if i>Key,break;end   %Least sig. bit 7
        binary_text_reconstructed(i)=pix_bi(2);
    end
    if mode==6
        pix_bi=de2bi(carrierfile_Em(i),8,'left-msb');
        binary_text_reconstructed(i)=pix_bi(8);
        i=i+1;if i>Key,break;end   %Least sig. bit 2
        binary_text_reconstructed(i)=pix_bi(7);
         i=i+1;if i>Key,break;end   %Least sig. bit 3
        binary_text_reconstructed(i)=pix_bi(6);
         i=i+1;if i>Key,break;end   %Least sig. bit 4
        binary_text_reconstructed(i)=pix_bi(5);
         i=i+1;if i>Key,break;end   %Least sig. bit 5
        binary_text_reconstructed(i)=pix_bi(4);
       i=i+1;if i>Key,break;end   %Least sig. bit 6
        binary_text_reconstructed(i)=pix_bi(3);
    end
    if mode==5
        pix_bi=de2bi(carrierfile_Em(i),8,'left-msb');
        binary_text_reconstructed(i)=pix_bi(8);
        i=i+1;if i>Key,break;end   %Least sig. bit 2
        binary_text_reconstructed(i)=pix_bi(7);
         i=i+1;if i>Key,break;end   %Least sig. bit 3
        binary_text_reconstructed(i)=pix_bi(6);
         i=i+1;if i>Key,break;end   %Least sig. bit 4
        binary_text_reconstructed(i)=pix_bi(5);
         i=i+1;if i>Key,break;end   %Least sig. bit 5
        binary_text_reconstructed(i)=pix_bi(4);
    end
```

```
   if mode==4
         pix_bi=de2bi(carrierfile_Em(i),8,'left-msb');
         binary_text_reconstructed(i)=pix_bi(8);
         i=i+1;if i>Key,break;end   %Least sig. bit 2
         binary_text_reconstructed(i)=pix_bi(7);
          i=i+1;if i>Key,break;end   %Least sig. bit 3
         binary_text_reconstructed(i)=pix_bi(6);
          i=i+1;if i>Key,break;end   %Least sig. bit 4
         binary_text_reconstructed(i)=pix_bi(5);
     end
     if mode==3
         pix_bi=de2bi(carrierfile_Em(i),8,'left-msb');
         binary_text_reconstructed(i)=pix_bi(8);
         i=i+1;if i>=Key,break;end   %Least sig. bit 2
         binary_text_reconstructed(i)=pix_bi(7);
          i=i+1;if i>=Key,break;end   %Least sig. bit 3
         binary_text_reconstructed(i)=pix_bi(6);
     end
     if mode==2
         pix_bi=de2bi(carrierfile_Em(i),8,'left-msb');
         binary_text_reconstructed(i)=pix_bi(8);
         i=i+1;if i>Key,break;end   %Least sig. bit 2
         binary_text_reconstructed(i)=pix_bi(7);

     end
     if mode==1
         pix_bi=de2bi(carrierfile_Em(i),8,'left-msb');
         binary_text_reconstructed(i)=pix_bi(8);

     end
     i=tt;
end
binary_text_reconstructed=reshape(binary_text_reconstructed,[],
8);
text_reconstructed=bi2de(binary_text_reconstructed);
A=char(text_reconstructed)';
disp('The Extracted Text is :')
disp(char(text_reconstructed)');
out_data=[A];
file=fopen('text.txt','w');
fwrite(file,out_data,'uint8');
fclose(file);
```

Calling code: textstegde **('mode') ,** *Input :* embedded file 'Steg-image-with-text.bmp' , *Output*: the recovered file ('text.txt').

The previous system is implemented so an ASCII represented symbols could be concealed in an image file using the simple LSB algorithm with controlling the number of employed bit positions and with key generated as a function of data length.

The Channel and Modulation function

MATLAB performs the modulation and demodulation process and AWGN channel modeling by the functions [**awgn()**] and [**pskmod**] .

```
function []=noisych(M,snr)

name=uigetfile('*.*');
file_open=fopen(name,'r');
file_read=fread(file_open,'ubit1');
fclose(file_open);
output=file_read;

sig=pskmod(output,M);
rsig=awgn(sig,snr);
dsig=pskdemod(rsig,M);

file=fopen('noisych.CH','w');
fwrite(file,dsig,'ubit1');
fclose(file);
```

Calling code: ***noisych*** **(modulation order , SNR)**

Input : BCH coded file (*.BCH)

Output: BCH file with white Gaussian noise added.

References

[1] Joan Daemen, Vincent Rijmen , *The design of Rijndael: AES-the advanced encryption standard.* Springer, 2002

[2] S. Lin and D.J. Costello, Jr. *Error ControlCoding: Fundamentals and Applications*, Englewood Cliffs, NJ: Prentice Hall, 1983.

[3] Ingemar J. Cox ,*Digital watermarking and steganography*, Morgan Kaufmann, 2008

[4] Donald Adjeroh, Timothy C. Bell, Amar Mukherjee, *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching, Springer, 2008*

[5] David Salomon,*Data compression: the complete reference*, Springer, 2004

[6] Ian Glover, Peter M. Grant, *Digital Communications*, Prentice Hall, 2009

[7] Bernard Sklar*, Digital communications: fundamentals and applications*, Prentice Hall

[8] John B. Anderson, Seshadri Mohan*, Source and channel coding: an algorithmic approach*, Springer, 1991

[9] Hans Dobbertin, Vincent Rijmen, *Aleksandra Sowa, Advanced encryption standard – AES, Springer, 2005*

[10] Lin, Shu, *Error control coding : fundamentals and applications*, Pearson-Prentice Hall, 2004

[11] S.R. Kodituwakku, U. S.Amarasinghe*, Comparison Of Lossless Data Compression Algorithms For Text Data*, Indian Journal of Computer Science and Engineering