



**NORTH-WEST UNIVERSITY
YUNIBESITHI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT**

**Secure Multi-Level Steganographic System as Security
Measure to Safeguard Data Transmission and Sharing
over the Internet**

by

**Adeniji Adedayo Adeolu
(Student Number: 25171895)**

LIBRARY
MAFIKENG CAMPUS
CALL NO.:
2021 -02- 1 1
ACC.NO.:
NORTH-WEST UNIVERSITY

**Department of Computer Science
School of Mathematical and Physical Sciences
Faculty of Agriculture, Science and Technology
North West University, Mafikeng Campus
South Africa**

**Submitted in Fulfillment for the Award of Master of Science Degree
in Computer Science**

Supervisor: Prof. Esiefarienrhe M.B

Co-Supervisor: Dr. Gasela N.

March 2015

DECLARATION

I ADENIJI ADEDAYO ADEOLU hereby declare that this dissertation titled Secure Multi-Level Steganographic System as Security Measure to Safeguard Data Transmission and Sharing over the Internet is my own work carried out at North West University, Mafikeng Campus, and has never been presented for the award of any degree in any university. All the information used has been duly acknowledged both in text and in the references.

Signature.....

Date.....

Adeniji Adedayo Adeolu

APPROVAL:

Signature *For N. B. Esiefarienrhe*.....

Date *01/10/2015*.....

Supervisor: Prof. Michael B. Esiefarienrhe

Department of Computer Science,

School of Mathematical and Physical Science,

Faculty of Agriculture, Science and Technology,

North West University, Mafikeng Campus

South Africa

Signature *reas*.....

Date *01/10/2015*.....

Co-Supervisor: Dr. Naison Gasela

Head of Department of Computer Science,

School of Mathematical and Physical Science

Faculty of Agriculture, Science and Technology,

North West University, Mafikeng Campus,

South Africa.

DEDICATION

This research project is dedicated to Almighty God the beginning and ending, and to my wife and children, Olanike Oladayo Adeniji, Praise Peace and Precious Adeniji.

ACKNOWLEDGEMENTS

First and foremost, I would like to give my deepest expression of gratitude to the King of kings and the Lord of lords, the Almighty God, for allowing me to go through this research process fully resourced, and for the people He placed around me. Almighty God, I thank You for the successful completion of this research work, all the wisdom and knowledge you gave me. Without Your help none of this would have been possible.

Several people have contributed one way or the other to the results of this dissertation, and I would like to use this opportunity to thank them all for their guidance and support. I thank my supervisor Prof. Michael B. Esiefarienrhe and co supervisor Dr. Naison Gasela for all their guidance.

I am also grateful to the lecturers and staff of the Department of Computer Science, North West University, Mafikeng Campus, for their help and support. I want to give special thanks to my parents Chief J.A Ade Adeniji, Mrs V.T Ade Adeniji, Dr. I.S Aiyegoro and Mrs E.F. Aiyegoro. Beside I wish to express my special appreciation to Prof. Eno E. Ebenso and to my loving brothers Dr. Olayinka Aiyegoro and Mr Adewole Adesina Adeniji,

I will like to thank my lovely wife, Olanike Oladayo Adeniji, who gave me all the support since we got married. She is my inspiration, motivator, sister, mother and helper. My wife sacrifices all her precious time to train the children in the course of my absence from home while chasing the dream.

To my children -Praise, Peace and Precious, thank you for your support and love. I love you all.

Finally, my thanks go to the management of Osun State Polytechnic Iree, and the Director of MIS for giving me the opportunity to continue in my academic career, I sincerely thank you all.

ABSTRACT

Recent advances in steganography systems have shown good results in bringing reliable data security over the Internet. However, these systems have their unique vulnerabilities since they are faced with a variety of threats from attackers. Effective steganography systems must be able to conceal hidden data completely as if there were nothing hidden on them. That is, they must be undetectable to steganalysis procedures from unauthorized individuals. This work proposes a secure multi-level steganography system to safeguard data transmission and sharing over the Internet, wherein cryptographic and steganographic techniques are combined. It also proposes compressing the cipher-text file using the Lempel-Ziv algorithm to reduce its size so that it uses less space in the cover image after data embedding. In this work, the use of Discrete Cosine Transform and Inverse Discrete Cosine Transform for image processing is explained. Since they will be used in the process of image compression and decompression respectively. The results show this technique allows more data to be embedded and the quality of the reconstructed (stego) image is better compared to the previous stego applications.

Keywords: Steganography Cryptography Cipher-text Stego Image Decompression Discrete Cosine Transformation Lempel-Ziv Algorithm.

TABLE OF CONTENTS

DECLARATION	ii
DEDICATION	iii
ACKNOWLEDGEMENTS	iv
ABSTRACT	v
LIST OF FIGURES	ix
LIST OF TABLES	x
DEFINITION OF CONCEPTS	xi
LIST OF ACRONYMS	xii
CHAPTER ONE	1
1.1 Introduction and Background	1
1.2 Problem Statement	2
1.3 Research Rationale	3
1.4 Research Questions	4
1.5 Research Goal	4
1.6 Research Objective	4
1.7 Research Methodology	4
1.7.1 Literature Survey	6
1.7.2 Model Design	6
1.7.3 Model Implementation	6
1.7.4 Proof of Concept	6
1.7.5 Included Publication	7
1.7.6 Chapter Summary	7
CHAPTER TWO	8
2.0 Literature Review	8
2.1 Chapter Overview	8
2.2 Cryptography	9
2.2.1 Encryption and Decryption	9

2.2.2	Types of Cryptography Algorithms	10
2.3	Data Encryption Standard (DES).....	11
2.4	Data Compression.....	12
2.4.1	Types of Data Compression	13
2.5	Information Hiding	14
2.6	Steganography	15
2.6.1	Principles of Steganography	16
2.6.2	Types of Steganography.....	17
2.6.3	Steganography Media	17
2.6.4	Steganography Techniques.....	19
2.6.5	Steganography Applications	20
2.7	Digital Images.....	20
2.8	Concept of an Image	21
2.8.1	Image Representation.....	22
2.8.2	Image Processing	23
2.8.3	Image Compression	23
2.9	Existing Steganography Techniques.....	24
2.10	Two steganography applications : P2P and Quickstego	28
2.10.1	QuickStego Steganography Application.....	29
2.11	Tools used in the Implementation of this Design	30
2.11.1	The .NET Framework.....	30
2.11.2	Visual Studio	31
2.12	Chapter Summary	32
CHAPTER THREE		33
3.0	Research Methodology	33
3.1	Chapter Overview	33
3.2	System Design and Architecture.....	33
3.3	System Model	33

3.3.1	Message	34
3.3.2	Data Encryption Algorithm	35
3.3.3	Compression	38
3.4	Image Compression and Encoding	41
3.4.1	The Discrete Cosine Transform	42
3.4.2	Quantization	43
3.5	Information Concealment in an Image	44
3.6	Steganography Application.....	48
3.6.1	Stego-Encoder and Stego-Decoder.....	49
3.7	Data Modeling	50
3.7.1	The Use Case Diagram	50
3.7.2	Class Diagram.....	52
3.8	Quality of Reconstructed Image	54
3.9	Summary.....	54
CHAPTER FOUR		55
4.0	System Implementation.....	55
4.1	System Architecture.....	55
4.2	Interface Design.....	56
4.2.1	Message Processing Section	56
4.2.2	Image Processing Section	57
4.3	Experimental Setup and Results	59
4.4	Chapter Summary	63
CHAPTER FIVE		64
5.0	Summary, Conclusion and Recommendation.....	64
5.1	Summary.....	64
5.2	Conclusion	65
5.3	Recommendation.....	65
REFERENCES.....		66

APPENDIX 70

LIST OF FIGURES

Figure 2-1 Encryption and decryption using a key 10

Figure 2-2 The DES algorithm 12

Figure 2-3 Information hiding system features 14

Figure 2-4 Graphical Version of the Steganography system 15

Figure 2-5 RGB Colour Cube 18

Figure 2-6 Structure of a digital image 21

Figure 2-7 zig-zag ordering of JPEG image components 24

Figure 2-8 Snapshot of P2P application..... 29

Figure 2-9 Snapshot of QuickStego steganography application..... 29

Figure 2-10 The .NET Framework 31

Figure 2-11 Visual Studio IDE 32

Figure 3-1 The system design of the Multi-Level steganography 33

Figure 3-2 Phases in DES Algorithm 35

Figure 3-3 Image Compression Process 42

Figure 3-4 Quantization table 43

Figure 3-5 The Steganography Encoder Architecture 49

Figure 3-6 The steganography decoder architecture..... 50

Figure 3-7 Use case diagram representing the receiver end of the system 52

Figure 3-8 Class diagram of the steganography system 53

Figure 3-9 Class diagram of the steganography system 53

Figure 4-1 The Architecture of Steganographic System.....55

Figure 4-2 Interface design of the Steganography system.....56

Figure 4-3 Message processing section of the Steganography system 56

Figure 4-4 Image processing section of the Steganography System 57

Figure 4-5 Stego Image section of the Steganography System 58

Figure 4-6 Stego Image processing section of the Steganography system.....58

Figure 4-7 Message processing and retrieving section of the Steganography System 59

Figure 4-8 Images showing Hydrangea, Jellyfish and Lena.....60

Figure 4-9 Graph showing the size of the text file and the stego images 63

Figure 4-10 Graph showing the size of the text file, compressed text file and the stego images.....63

LIST OF TABLES

Table 4-1 Size of the text file and the stego images 60

Table 4-2 Size of the text file, compressed file and the stego images 61

Table 4-3 PSNR comparisons between multi-level application, P2P application, and Quickstego application.....62

DEFINITION OF CONCEPTS

Colour Image:	is a digital image that includes colour information for each pixel.
Steganography:	is the science of hiding information by embedding the hidden (secret) message within other, seemingly harmless images, audio, video files or any other media. It is the process of hiding of a secret message within an ordinary message and extracting it at its destination. Anyone else viewing the message will fail to know it contains hidden/encrypted data.
System:	a set of detailed methods, procedures and routines created to carry out a specific activity, perform a duty, or solve a problem.
Data:	is a distinct piece of information, usually formatted in a special way.
Data Security:	data security basically means protection of data from unwanted and unauthorized access.
Data transmission:	also known as digital transmission, or digital communications, is the physical transfer of data (a digital bit stream) over a point-to-point or point-to-multipoint communication channel.
Data Sharing:	is the practice of making data used for scholarly research available to other investigators.
Internet:	is a global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to link several billion devices worldwide.

LIST OF ACRONYMS

TCP/IP:	Transmission Control Protocol/Internet Protocol
DT:	Data Transmission
PSNR:	Peak Signal-to-Noise Ratio
LSB:	Least Significant Bits
IDCT:	Inverse Discrete Cosine Transform
ASCII:	American Standard Code for Information Interchange
MATLAB:	Matrix Laboratory
BIT:	Binary Digit
DES:	Data Encryption Standard
DCT:	Discrete Cosine Transform
DSAD:	Digitally Sampled Analog Data
SKC:	Secret Key Cryptography
PKC:	Public Key Cryptography
JPEG:	Joint Photographic Experts Group
HVS:	Human Visual System
GIF:	Graphics Interchange Format
RGB	Red, Green, Blue.

CHAPTER ONE

Introduction and Background

1.1 Introduction and Background

The Internet has transformed the present world and there are many Internet based applications that are available these days, which have added to the comfort and connectivity in every aspect of human life. It is estimated that, over 3,035,749 people worldwide use Internet for various purposes, ranging from accessing information for educational needs to financial transactions, procurement of goods and services [1]. However, data security is important these days. Data security means protection of data from unwanted and unauthorized access. Data can be a random signal, text, voice, image, video, etc [4, 5]. Many times, sensitive and crucial data needs to be transmitted securely to the receiver through the Internet. For this, network specific encryption is not reliable, and hence an added technique to secure the data is needed, so that the crucial data transmitted could not be accessed even if someone is able to access the network routers [2, 3].

For primitive protection, cryptography techniques have been widely used to encrypt the plaintext data, transfer the ciphertext over the Internet and decrypt the ciphertext to extract the plaintext at the receiver side [2]. However, with the ciphertext not really making much sense when interpreted as it is, a hacker or an intruder can easily perceive that the information being sent on the channel has been encrypted and is not the plaintext. This can raise the curiosity level of a malicious hacker or intruder to conduct cryptanalysis attacks on the ciphertext (i.e., analyze the ciphertext through the encryption algorithms and decrypt the ciphertext completely or partially) [2]. Therefore, the need arises for a more prudent and secured way to send the confidential information, either in plaintext or ciphertext, by cleverly embedding it as part of a cover media such as an image, audio or video file, in such a way that the hidden information cannot be easily perceived to exist for the unintended recipients of the cover media. This idea forms the basis for steganography.

Steganography is the science of hiding information by embedding the hidden (secret) message within other, seemingly harmless images, audio, video files or any other media.

To secure one to one communication, steganography is an important technique. The principle is that the third party must not have a hint of what is being transferred. Steganography enables information transfer in a covert manner such that it does not draw the attention of the unintended recipients [6]. The cover image is used to carry the secret message, because the human eye is insensitive to very

minute changes in colours, and hence secret message intelligently embeds the data and transmits it to the receivers [4, 7]. When received, the data is securely retrieved by extraction. by decrypt the message through the secret key and decompressing al

There are two types of steganography techniques

- i Spatial domain steganography
- ii Transform domain steganography.

Each one of them has several data embedding and compatible extraction techniques. After data embedding, some additional optimization techniques can be implemented to reduce the error even further [4, 8]. There are various techniques to hide and extract random data, text, numbers, voice and even images into cover images, with negligible error and good Peak Signal-to-Noise Ratio (PSNR).

Steganographic technologies are a very important part of the present and future of security and privacy on open systems such as the Internet [9]. Steganographic research is primarily driven by the lack of strength in the cryptographic systems on their own and many governments (especially in developed countries) have created laws that either limit the strength of cryptosystems or prohibit them completely. This has been done primarily for fear by law enforcement agent not to be able to gain intelligence by wiretaps, etc. This unfortunately leaves the majority of the Internet community either with relatively weak, and often breakable, encryption algorithms, or none at all [10]. This is where steganography is useful, as it can be used to hide important data inside another file. To improve the security of the information to be hidden, it is good practice to use cryptography and steganography together, because both technologies can provide a very acceptable amount of privacy in any system [11].

1.2 Problem Statement

In this highly digitalized world, the Internet plays an important role for data transmission and sharing. However, due to its (Internet) open nature, some confidential data might be stolen, copied, modified, or destroyed by an attacker. Therefore, security is pertinent to prevent confidential data from being detected during transmission. Encryption is a well-known procedure for secured data transmission. Although encryption achieves certain security effects, it makes the secret messages unreadable and unnatural, and this protection can be broken with enough computational power.

An alternate approach to encrypting data would be to hide it by making this information look like something else and to hide the existence of the communication; this way only the intended recipient would realize its true content [12]. Most of the current steganography approaches lack depth in

security and capacity, that is, they are easily detected by steganalysis and the amount of information they can effectively hide is low. Therefore, the need to develop a system that does not necessarily replace the existing one but instead improves its security and hidden capacity is very important [13].

1.3 Research Rationale

The Internet as a whole does not use secured links, thus information in transit may be vulnerable to interception. The importance of reducing the chances of the information being detected during the transmission has been an issue in recent time [14].

There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. One essential aspect for secure communications is that of cryptography.

Cryptography is useful in protecting against a wide variety of attacks on the communications between two parties. It is the art and science of preparing coded or protected communications intended to be intelligible only to the person possessing a secret key. Cryptography techniques have been widely used to encrypt the plaintext data to ciphertext, transfer the ciphertext over the Internet and decrypt the ciphertext to extract the plaintext at the receiver side. However, with the ciphertext not really making much sense when interpreted as it is, a hacker or an intruder can easily perceive that the information being sent on the channel has been encrypted and is not the plaintext. This can naturally arouse the curiosity level of a malicious hacker or intruder to conduct cryptanalysis attacks on the ciphertext (that is, analyze the ciphertext through the encryption algorithms and decrypt the ciphertext completely or partially). But it is important to note that while cryptography is necessary for secure communications, it is not by itself sufficient.

The need arises for a more secure way to send secret information, either in plaintext or ciphertext, by cleverly embedding it as part of a cover media such as an image, audio or video file in such a way that the hidden information cannot be easily perceived to exist for the unintended recipients of the cover media. This idea forms the basis for Steganography, which is the science of hiding information by embedding the hidden (secret) message within other, seemingly harmless images, audio, video files or any other media. Steganography protects the intellectual property rights and enables information transfer in a covert manner such that it does not draw the attention of unintended recipients [11].

1.4 Research Questions

This research work provides answer to the following questions:

- RQ1: How can data be hidden in such a way that it will be difficult for attackers to detect the existence of a secret message(s)?
- RQ2: How can the amount of data hidden be increased in a file?
- RQ3: If someone intercepts a file, what should be done to prevent easy removal or modification of data?
- RQ4: What model can be developed to provide undetectability, robustness and the high capacity of the hidden data?

1.5 Research Goal

The main goal of this research is to develop a secure multi-level steganographic system as a security measure to safeguard data transmission and sharing over the Internet.

1.6 Research Objective

To achieve the main goal of this research, the following objectives are employed:

- i Development of compression techniques to increase the amount of information that can be hidden.
- ii Design of a secured multi-level steganography model.
- iii Combination of objectives one and two to prevent easy removal or modification of data by the intruder.
- iv Implementation of the model designed in (ii).

1.7 Research Methodology

In this research, the following research methodologies were used:

Method

Steganography is the art of concealing information in ways that prevent the detection of hidden messages. For communication to occur between a sender and a receiver, the sender supplies a message M (which can either be in the form of plaintext or file), a key K and cover image C ; combines them to generate a stego image Z , and sends it to the receiver.

$$Z = f(M, K, C) \quad (1.1)$$

All data (Original Plaintext) is encoded. This means that the data is originally a combination of elements, e , from some alphabet, A . This combination of elements is a message, M . This message

defined over an alphabet, A , is encoded into the binary alphabet, B . The string of bits, binary digits (0's and 1's), is the encoded data. The encoded message B (in plaintext form P) will be encrypted using data encryption standard (DES) with key K to generate ciphertext C .

$$E(K, P) = C \quad (1.2)$$

where E represent encryption

The ciphertext file is compressed using Lempel-Ziv algorithm to shrink down the size so that it takes up less space in the cover image after data embedding. The Lempel-Ziv algorithm constructs its dictionary on the fly, only going through the data once. After the cipher text has been compressed, it will be embedded into the cover image.

A digital image is a rectangular grid of pixels, or "picture elements". Image processing is started by breaking the image into 8×8 blocks of pixels, working from left to right, and top to bottom, the Discrete Cosine Transform is applied to each block. The array of compressed blocks that constitute the image is stored in a drastically reduced amount of space. When desired, the image is reconstructed through decompression, a process that uses the Inverse Discrete Cosine Transform ($IDCT$).

The DCT takes the image input and transforms it into a linear combination of weighted basis functions. These basis functions are commonly the frequency. The 2D Discrete Cosine Transform is just a one dimensional DCT applied twice, once in the x direction, and again in the y direction. The DCT equation computes the i th, j th entry of the DCT of an image.

$$D(i, j) = \frac{1}{\sqrt{2N}} C(i) C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x, y) \cos \left[\frac{(2x+1)i\pi}{2N} \right] \cos \left[\frac{(2y+1)j\pi}{2N} \right] \quad (1.3)$$

$$C(u) = \begin{cases} \frac{1}{2} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{cases} \quad (1.4)$$

$p(x, y)$ is the x, y th element of the image represented by the matrix p . N is the size of the block that the DCT is done on. The equation calculates one entry i, j th of the transformed image from the pixel values of the original image matrix.

For the information to be concealed in a cover image, the capacity of the image is determined as shown in equation (1.5) and then we proceed to information concealment. The process of a message insertion in a cover image uses a key-dependent permutation. In this sense, the secret key given by the sender is used to encode the cover image before the message embedding process takes place.

$$\text{Capacity} = DCT - DCT_1 - DCT_0 \quad (1.5)$$

where DCT is the number of DCT coefficients, DCT_0 is the zero DCT coefficients and DCT_1 is the number of DCT coefficients with value 1.

The secret key is converted to its bits equivalence (in groups of 8-bits) and the cover image is converted to its array of pixels (each pixel is a combination of three components: R, G, and B). The bits of the first component (blue component) of pixels of the image are replaced with secret bits group and then encoded with the green component of the pixel until all the entire secret key has been used.

The peak signal to noise ratio (PSNR) values of the proposed system and that of the existing systems (P2P and Quickstego) are computed and compared with one another. The Tools used for the system implementation include the C# programming language and the Microsoft .NET framework.

1.7.1 Literature Survey

A thorough literature survey was conducted on the previous research activities within the study area with a view to improving acquaintance with the subject matter and to serve as a ground work to the present investigation.

1.7.2 Model Design

A secured multi-level model was developed based on the literature survey to provide undetectability, robustness and high capacity of the hidden data.

1.7.3 Model Implementation

There is provision of evidence on applicability of the model designed by showing results in the form of simulation. The model developed simulated using data collected from various computer images with extension jpg and text files. The results obtained are attached as evidence.

1.7.4 Proof of Concept

The performance of the implemented model was evaluated using an amount of data that can easily be hidden. This technique increases the amount of data that can be hidden. The system also makes it difficult for data to be detected, removed or modified by an intruder. This is evident from the results obtained. The results show that the application developed has higher PSNR than both P2P and Quickstego. Hence, the proposed method provides a better stego image quality. The stego images remain unchanged after evaluating the proposed approach.

1.7.5 Included Publication

Part of the research reported in this dissertation has been published and another are also accepted for publication and they are under review by an accredited journals. These are the papers:

- (i) Adedayo Adeolu Adeniji, Micheal Esiefarienrhe and Naison Gasela: Architectural Design of Multi Level Steganography system for Data Transmission. *International Conference on Chemical Engineering and Advance Computational Technologies (ICCEACT' 2014)*, Nov. 24-25, 2014 Pretoria, South Africa.
- (ii) Adedayo Adeolu Adeniji Bukohwo Michael Esiefarienrhe and Naison Gasela: Analysis of Multi Level Steganography system for Data Security, *Global Academic Network International Conference*, Nov 12th - 15th Miami, USA.
- (iii) Adeniji Adedayo Adeolu, Michael Esiefarienrhe and Naison Gasela: Implementation of a Secure Multi-Level Steganography System for Data Security, *International Journal of Computer Science and Applications USA, Volume 2, Number 4, 2015, pp. 119-126 ISSN: 2333-9071 (Print); ISSN: 2333-908X (Online) 2015.*

1.7.6 Chapter Summary

This chapter presented a brief formal summary of this project work. The chapter is systematically arranged for easy comprehension. It contains background to the research work followed by the problem statement, aims and objectives of the research. It also discussed the research methodology employed in this dissertation and gave a summary of the dissertation in terms of chapters. The next chapter will discuss in detail the components, technologies and concepts of steganographic systems and review the literature of other researchers in this area of study.

CHAPTER TWO

2.0 Literature Review

2.1 Chapter Overview

The Internet has brought great changes to many organizations. Governments use the Internet for internal communication, such as dissemination of information, automated tax processing, etc. In addition to offering goods and services online to customers, businesses use the Internet to interact with other businesses. Individuals use the Internet for communicating through electronic mail (e-mail), retrieving news, researching information, shopping, paying bills, banking, listening to music, watching movies, playing games, and even making telephone calls. Educational institutions use Internet for research and deliver of online courses and course material to students among other uses [13].

The Internet as a whole does not use secure links, thus information in transit may be vulnerable to interception by imposter. The important of reducing the chance of the information being detected during transmission has been an issue in recent times [14].

Cryptography and Steganography are both technologies used to protect information over the Internet. Cryptography has been widely used to encrypt plaintext data, transfer the ciphertext over the Internet and decrypt the ciphertext to extract the plaintext at the receiver side. The ciphertext is not really secured however because hackers or intruders can easily observe that the information being sent on the channel has been encrypted. However, a steganography technique proves more prudent and secure when compared with cryptography techniques. steganography is not intended to replace cryptography but rather to complement it. If a message is encrypted and hidden with a steganographic method it provides an additional layer of protection and reduces the chance of the hidden message being detected.

Steganographic technologies are important part of the present and future of security and privacy on open systems such as the Internet. The following points can be attributed to the growth of steganography [15]:

- i. Government ban on digital cryptography so that their electronic eavesdropping will continue to be effective. Individuals and companies who seek confidentiality look to steganography as an important complementary technique since combining cryptography and steganography can help in avoiding suspicion and protecting privacy.

- ii. The increased need to protect intellectual property rights by digital content owners, using efficient watermarking.
- iii. The trend towards electronic communications and human desire to conceal messages from curious eyes. Steganographic software is becoming effective in hiding information in image, audio or text files.

2.2 Cryptography

Cryptography (Greek *kryptos*, “secret”; *graphos*, “writing”) refers both to the process or skill of communicating in or deciphering secret writings (codes, or ciphers). Cryptography can be defined as the study of information hiding and verification which includes the protocols, algorithms and strategies to securely and consistently prevent or delay unauthorized access to sensitive information and enable verifiability of every component in a communication. In essence, the implementing of security using cryptography concerns four main goals which are:

- i. message confidentiality (or privacy): Only an authorized recipient should be able to extract the contents of the message from its encrypted form, resulting from steps to hide, stop or delay free access to the encrypted information.
- ii. message integrity: The recipient should be able to determine if the message has been altered.
- iii. sender authentication: The recipient should be able to verify from the message, the identity of the sender, the origin or the path it traveled (or combinations) so as to validate claims from the emitter or to validate the recipient expectations.
- iv. sender non-repudiation: The sender should not be able to deny sending the message.

2.2.1 Encryption and Decryption

A message in human readable form is referred to in cryptographic terms as plaintext or cleartext. The process of scrambling a message in such a way it becomes unintelligible called encryption and the resulting message is referred to as ciphertext. As shown in Figure 2.1, the reverse process (decryption) takes ciphertext as input and restores the original plaintext.

Plaintext is denoted by P , ciphertext is denoted by C and key is denoted by K . The encryption function E operates on P to produce C :

$$E(K, P) = C \quad (2.1)$$

In the reverse process, the decryption function D operates on C to produce P :

$$D(K, C) = P \quad (2.2)$$

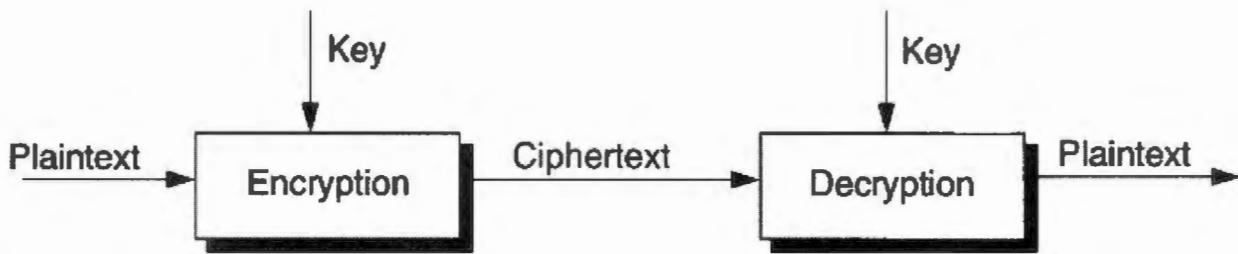


Figure 2-1 Encryption and decryption using a key

2.2.2 Types of Cryptography Algorithms

There are several ways of classifying cryptographic algorithms. The categories below are categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms are [16] :

- i. Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption;
- ii. Public Key Cryptography (PKC): Uses one key for encryption and another for decryption;
- iii. Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information.

(i) Secret Key Cryptography

With secret key cryptography, a single key is used for both encryption and decryption. The sender uses the key to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key to decrypt the message and recover the plaintext. Secret key cryptography is also called symmetric encryption. The key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, is the distribution of the key.

Secret key cryptography schemes are generally categorized as being either stream ciphers or block ciphers. Stream ciphers operate on a single bit (byte or computer word) at a time and implement some form of feedback mechanism so that the key is constantly changing. A block cipher is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will always encrypt to the same ciphertext when using the same key in a block cipher whereas the same plaintext will encrypt to different ciphertext in a stream cipher. Secret key cryptography algorithms include DES, AES, RC, Blowfish, etc.

(ii) Public Key Cryptography

Public key cryptography employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work. Because a pair of secret keys is required, this approach is also called asymmetric cryptography.

In this scheme, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. Public-key cryptography algorithms include RSA, DSA, KEA, etc.

(iii) Hash Functions

Hash functions, also called message digests and one-way encryption, are algorithms that, in some sense, use no key. Instead, a fixed-length hash value is computed based upon the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered. Hash algorithms are typically used to provide a digital fingerprint of a file's contents, often used to ensure that the file has not been altered by an intruder or virus. Hash algorithms include among others Message Digest (MD), and Secure Hash Algorithm (SHA).

2.3 Data Encryption Standard (DES)

DES is a block cipher. This means that it operates on a single chunk of data at a time, encrypting 64 bits (8 bytes) of plaintext to produce 64 bits of ciphertext. The key length is 56 bits, often expressed as an eight-character string with the extra bits used as a parity check [18]. The algorithm has 19 distinct stages. The first stage reorders the bits of the 64-bit input block by applying a fixed permutation. The last stage is the exact inverse of this permutation. The stage penultimate to the last one exchanges the leftmost 32 bits with the rightmost 32 bits. The remaining 16 stages (called rounds) are functionally identical but take as an input a quantity computed from the key K_i and the old right half R_i , where i is the current round number. K_i is derived from the original 56-bit key passed as input to the algorithm. Figure 2.2 shows the overall process [18].

At each iteration, the algorithm takes in two 32-bit inputs and produces two 32-bit outputs. The left output is simply a copy of the right input. The right output is an exclusive OR (XOR) of the left input and a function of the right input and the key for the stage K_i . All the complexity lies in the function f ,

which does a number of substitutions and permutations using simple hardware elements called S-boxes (for substitution) and P-boxes (for permutation). Decryption in the DES algorithm uses the same sequence of steps, but the keys used at each of the 16 stages (K1 to K16) are applied in reverse order [17].

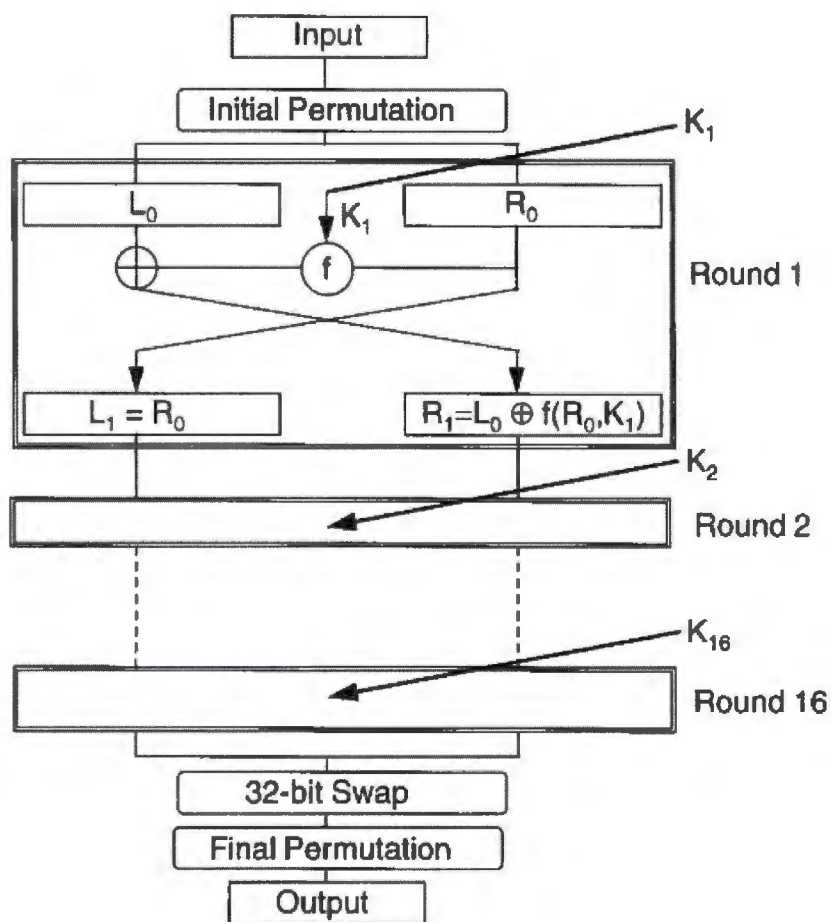


Figure 2-2 The DES algorithm [18].

2.4 Data Compression

Data Compression shrinks down a file so that it takes up less space. This is desirable for data storage and data communication. Storage space on disks is expensive so a file which occupies less disk space is "cheaper" than an uncompressed file. Smaller files are also desirable for data communication, because the smaller a file the faster it can be transferred [19]. A compressed file appears to increase the speed of data transfer over an uncompressed file. Data compression can be defined as the process of encoding information using fewer bits than the original representation would use in which the primary objective is to minimize the amount of data to be transmitted. Data compression is often referred to as coding [18].

A simple characterization of data compression is that it involves transforming a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. Data compression has important applications in the areas of data transmission and data storage. Many data processing applications require storage of large volumes of data, and the number of such applications is constantly increasing as the use of computers extends to new disciplines. At the same time, the proliferation of computer communication networks is resulting in massive transfer of data over communication links. Compressing data to be stored or transmitted reduces storage and/or communication quantity. When the amount of data to be transmitted is reduced, the effect is that of increasing the capacity of the communication channel. Similarly, compressing a file to half of its original size is equivalent to doubling the capacity of the storage medium. It may then become feasible to store the data at a higher, thus faster, level of the storage hierarchy and reduce the load on the input/output channels of the computer system [19].

The design of data compression schemes therefore involves trade-offs among various factors, including the degree of compression, the amount of distortion introduced (if using a lossy compression scheme), and the computational resources required to compress and uncompress the data. Compression was one of the main drivers for the growth of information during the past two decades.

2.4.1 Types of Data Compression

There are two main types of data compression: lossy and lossless. After one applies lossy data compression to a message, the message can never be recovered exactly as it was before it was compressed. When the compressed message is decoded it does not give back the original message. This means that data has been lost.

Lossy compression cannot be decoded to yield the exact original message. It is not a good method of compression for critical data, such as textual data. It is most useful for Digitally Sampled Analog Data (DSAD). DSAD consists mostly of sound, video, graphics, or picture files. Algorithms for lossy compression of DSAD vary, but many use a threshold level truncation. This means that a level is chosen past which all data is truncated. In a sound file, for example, the very high and low frequencies, which the human ear cannot hear, may be truncated from the file. Some examples of lossy data compression algorithms are JPEG and MPEG.

In a lossless data compression file, the original message can be exactly decoded. Lossless data compression works by finding repeated patterns in a message and encoding those patterns in an efficient manner. For this reason, lossless data compression is also referred to as redundancy reduction. Because redundancy reduction is dependent on patterns in the message, it does not work well on random messages. Lossless data compression is ideal for text. Most of the algorithms for lossless compression are based on the LZ compression method (Lempel Ziv) developed by Lempel and Ziv [19].

2.5 Information Hiding

Information hiding, steganography and watermarking are three closely related fields that have a great deal of overlap and share many technical approaches. However, there are fundamental philosophical differences that affect the requirements and thus the design, of a technical solution [21].

Information hiding (or data hiding) is a general term encompassing a wide range of problems beyond that of embedding messages in content. The term hiding can refer to either making the information imperceptible (as in watermarking) or keeping the existence of the information secret.

An information hiding system is characterized by having three different aspects that contend with each other as shown in Figure 2.3: capacity, security and robustness. Capacity refers to the amount of information that can be hidden in the cover medium, security to an eavesdropper's inability to detect hidden information, and robustness to the amount of modification the stego medium can withstand before an adversary can destroy hidden information.

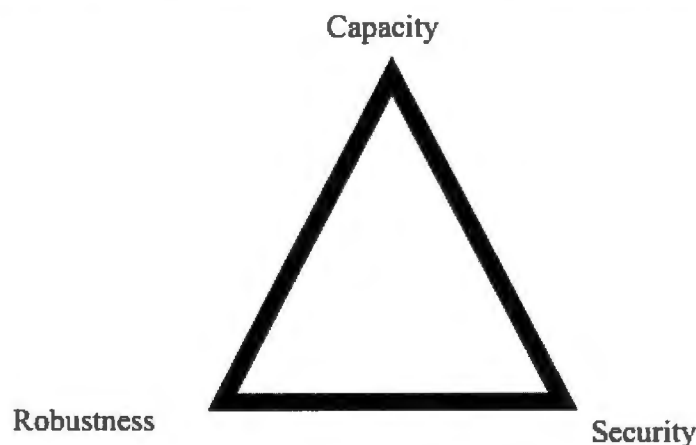


Figure 2-3 Information hiding system features [21].

Generally speaking, information hiding relates to both watermarking and steganography. A watermarking system's primary goal is to achieve a high level of robustness; that is, it should be impossible to remove a watermark without degrading the data object's quality. Steganography,

on the other hand, strives for high security and capacity, which often entails that the hidden information is fragile. Even trivial modifications to the stego medium can destroy it.

Digital watermarking is the process of embedding information into a digital signal which may be used to verify its authenticity or the identity of its owners. In digital watermarking, the signal may be audio, pictures, or video. If the signal is copied, then the information also is carried in the copy. A signal may carry several different watermarks at the same time [21].

2.6 Steganography

One of the reasons that intruders can be successful in manipulate secret message is that most of the information they acquire from a system is in a form that they can see, read and sometimes comprehend. Intruders may reveal the information to others, modify it to misrepresent an individual or organization, or use it to launch an attack. One solution to this problem is, the use of steganography [14].

Steganography is the art and science of writing hidden messages in such a way that no one apart from the intended recipient knows of the existence of the message. In contrast to cryptography, where the enemy is allowed to detect, intercept and modify messages without being able to violate certain security premises guaranteed by a cryptosystem, the goal of steganography is to hide messages inside other harmless messages in a way that does not allow any enemy even to detect that there is a second message present [14].

This involves selecting an appropriate carrier file such as an image, text or audio file, removing the less important information from that file and injecting the hidden message in its place. When the cover message and the secret message are combined, a stego image is created. Figure 2.4 illustrates a typical steganography system. Steganographic messages may be encrypted before they are inserted into the cover image for increased security. The receiver needs to know the technique used to encrypt the message before he can recover the message.

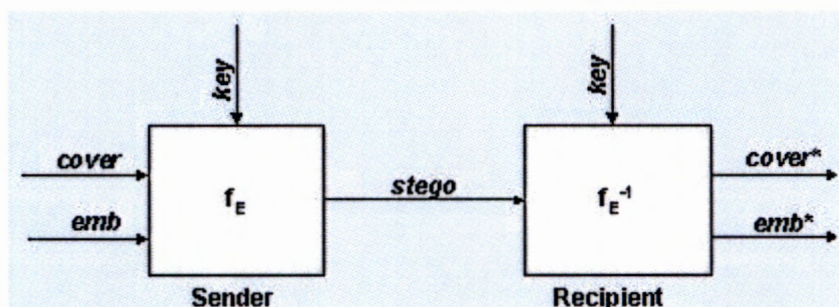


Figure 2-4 Graphical Version of the Steganography system

A famous illustration of steganography is Simmons' "Prisoners' Problem": Alice and Bob are in jail, locked up in separate cells far apart from each other, and wish to devise an escape plan. They are allowed to communicate by means of sending messages via trusted couriers, provided they do not deal with escape plans. But the couriers are agents of the warden Eve (who plays the role of the adversary here) and will leak all communication to her. If Eve detects any sign of conspiracy, she will thwart the escape plans by transferring both prisoners to high-security cells from which nobody has ever escaped. Alice and Bob are well aware of these facts, so that before getting locked up, they have shared a secret codeword that they are now going to exploit for embedding hidden information into their seemingly innocent messages. Alice and Bob succeed if they can exchange information allowing them to coordinate their escape and Eve does not become suspicious.

The purpose of steganography is not to keep others from knowing the very existence of the secret message. If a steganographic method causes someone to suspect the carrier medium, the method has failed [22]. Although messages embedded into an image are often imperceptible to the human eye, they often disturb the statistical nature of the image [23]. The choice of the cover image in steganography is very important as it influences the security of the technique in a huge way.

The cover-object (cover) is a carrier or medium to embed a message (emb). There are several suitable media that can be used as cover-objects, such as network protocols, audio, file and disk, a text file and an image file. A message is the data that the sender wishes to keep confidential and will be embedded into the cover-object, by using a stego-system encoder. Message can be a plain text, a ciphertext, an image, or anything that can be embedded in a bit stream such as a copyright mark or a serial number. A key (stego-key) is a password, which ensures that only the recipient who knows the corresponding decoding key will be able to extract the message from a cover-object. The output of the stego system encoder is known as the stego-object.

A stego system encoder (f_E) can be represented by using the relation:

$$I' = f_E(I, m, k) \quad (2.3)$$

where I' is the stego-object, I is the cover-object, m is the message and k is the stego-key.

2.6.1 Principles of Steganography

Three principles can be used to measure the effectiveness of a steganography technique. The principles are amount of data, difficulty of detection and difficulty of removal [24].

- i. Amount of data suggests that the more data you can hide the better the technique.

- ii. Difficulty of detection relates to how easy it is for someone to detect that a message has been hidden. Once you increase the amount of data hidden in a file, the risk that someone will be able to detect the message also becomes higher.
- iii. Difficulty of removal suggests that someone intercepting your file should not be able to remove the data easily.

2.6.2 Types of Steganography

Three steganography protocols exist, namely: Pure Steganography, Secret Key Steganography and Public Key Steganography [25]. In Pure Steganography, the embedding and extraction algorithms should only be known by the message sender and the intended receiver.

With Secret Key Steganography, it is assumed that a party other than the sender and intended receiver knows the embedding and extraction algorithms. The sender embeds a message in a cover object using a secret key known as a stego key. If a third party intercepts the stego object and extracts the information, the result will be scrambled. Only the intended receiver who possesses the same stego key can extract the original message [25].

Public Key Steganography is based on the principles of Public Key Cryptography. In Public Key Steganography both a public key and a private key are used. The public key is used in the embedding process and the private key is used in the extraction process. This allows the sender and the receiver to avoid exchanging a secret message which might be compromised. However, this method is susceptible to a man-in-the-middle attack [25].

2.6.3 Steganography Media

This refers to the cover objects that steganography uses during transmission. it include, images, texts, audio and video among others.

a. Steganography in Images

Coding secret messages in digital images is by far currently the most widely used of all methods [26]. Digital images take advantage of the limited human visual perception of colour. This field is expected to grow continually as computer graphics power also grows [27]. In a computer, images are represented as arrays of values or numbers that represent light intensities at various point or pixels (these pixels make up the images raster data). These values represent the intensities of the three colours that include red, green and blue where the value of each of the three colours describes a pixel

[28]. These pixels are represented horizontally row by row. The number of bits in a colour scheme, called the bit depth, refers to the number of bits used for each pixel. The least number of bits in a current colour scheme is eight which means 8-bits are used to describe the colour of each pixel [29]. Monochrome and grey scale images use 8-bits for each pixel and are able to display 256 different colours or shades of grey [28]. Digital colour images are typically stored in 24-bit files and use the RGB colour cube [29]. The RGB Colour cube is illustrated in Figure 2.5

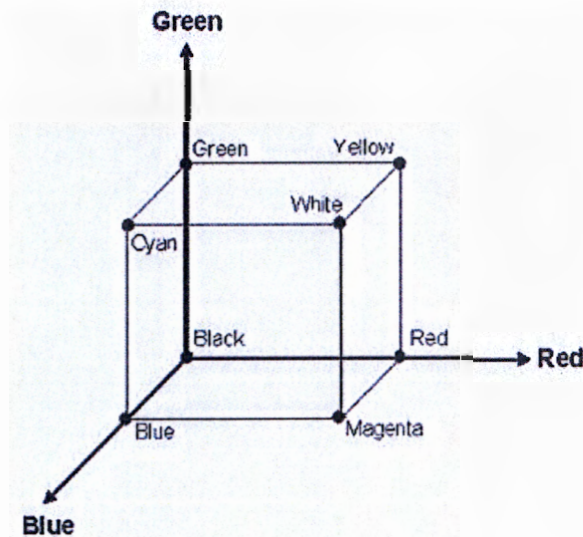


Figure 2-5 RGB Colour Cube [29].

b. Steganography in Audio

In a computer-based audio steganography system, secret messages are embedded in digital sound. The secret message is embedded by slightly altering the binary sequence of a sound file. Messages can be embedded in WAV, AU, and even in MP3 sound files [30]. When hiding information in an audio file, the technique which is commonly used is Low Bit Encoding which is similar to Least Significant Bit Insertion used in image steganography. The problem with Low Bit Encoding is that it is generally noticeable to the human ear. It is a risky method to use for masking information inside an audio file [30].

Spread Spectrum is another method used to conceal information inside an audio file. The method spreads the secret message over the sound file's frequency spectrum, using a code that is independent of the actual signal. As a result, the final signal occupies a bandwidth in excess of what is actually required for transmission [25]. Echo Data Hiding is another method of hiding information in an audio file. This method uses the echoes in sound files in order to try to hide information. By simply adding extra sound to an echo inside an audio file, information can be concealed. The advantage of this method is that it can improve the sound of the audio file. Due to the range of the Human

Auditory System (HAS), data hiding in audio signals is challenging. The HAS perceives over a range of power greater than one billion to one and a range of frequencies greater than one thousand to one [31]. The auditory system is very sensitive to additive random noise. However, while the HAS has a large dynamic range, it has a very small differential range – large sounds tend to drown quiet sounds. When performing steganography in audio files, the steganographer must exploit the weakness of the HAS while being aware of the extreme sensitivity of the human auditory system [31].

c. Steganography in Video

When information is hidden inside a video, the steganographer will usually use the DCT (Discrete Cosine Transform) method [30]. DCT works by slightly changing each of the images in the video, only so much that it will not be noticeable to the human eye. DCT alters the values of certain parts of the images, usually rounding up the values. Steganography in videos is similar to that of steganography in images apart from the fact that information is hidden in each frame of the video. As with image steganography, the less information you hide in the video, the smaller the chance that someone will notice, and the more information you hide, the higher the risk of discovery.

d. Steganography in Documents

The use of steganography in documents works by simply adding white space and tabs to the end of the lines of a document. This type of steganography is very effective because the use of white space and tabs is not noticeable to the human eye. White space and tabs occur naturally in documents; therefore this method of steganography will not arouse suspicion [30].

2.6.4 Steganography Techniques

Over the past few years, numerous steganography techniques that embed hidden messages in multimedia objects have been proposed. There have been many techniques for hiding information or messages in images in such a manner that the alterations made to the image are perceptually indiscernible. Common approaches include:

- i. Least significant bit (LSB)
 - ii. Masking and filtering
 - iii. Transform techniques
- i. Least significant bit (LSB) insertion is a simple approach to embedding information in an image file. The simplest steganographic techniques embed the bits of the message directly into the least significant bit plane of the cover-image in a deterministic sequence. Modulating the least

significant bit does not result in human-perceptible difference because the amplitude of the change is small. LSB encoding is by far the most popular of the coding techniques used for digital images. Depending on the color palette used for the cover image (e.g, all gray), it is possible to take 2 LSBs from one byte without the human visual system (HVS) being able to tell the difference. The only problem with this technique is that it is vulnerable to attacks such as image changes and formatting (i.e., changing from .GIF to .JPEG).

ii. Masking and filtering techniques for digital image encoding such as Digital Watermarking (that is integrating a company's logo on their web content) are more popular with techniques such as JPEG. This technique extends an image data by masking the secret data over the original data as opposed to hiding information inside the data. Some experts argue that this is definitely a form of Information Hiding, but not technically steganography. The beauty of these techniques is that they are immune to image manipulation, which makes their possible uses very robust. The technique performs the analysis of the image, thus embedding the information in significant areas, so that the hidden message is more integral to the cover image than just hiding it in the noise level.

iii. Transform techniques embed the message by modulating coefficients in a transform domain, such as the Discrete Cosine Transform (DCT) used in JPEG compression, Discrete Fourier Transform, or Wavelet Transform. These methods hide messages in significant areas of the cover-image, which make them more robust to attack. Transformations can be applied over the entire image, to block throughout the image, or other variants.

2.6.5 Steganography Applications

There are many applications for digital steganography of images, including copyright protection, feature tagging, and secret communication. A copyright notice or watermark can be embedded inside an image to identify it as intellectual property. If someone attempts to use this image without permission, we can prove it by extracting the watermark.

In feature tagging, captions, annotations, time stamps, and other descriptive elements can be embedded inside an image. Copying the stego-image also copies the embedded features and only parties who possess the decoding stego-key will be able to extract and view the features.

2.7 Digital Images

Pictures are the most common and convenient means of conveying or transmitting information. A picture is worth a thousand words. Pictures concisely convey information about positions, sizes and inter-relationships between objects. They portray spatial information that we can recognize as

objects. Human beings are good at deriving information from such images, because of our innate visual and mental abilities. About 75% of the information received by humans is in pictorial form [32].

A digital remotely sensed image is typically composed of picture elements (pixels) located at the intersection of each row i and column j in each K band of imagery. Associated with each pixel is a number known as a Digital Number (DN) or Brightness Value (BV), which depicts the average radiance of a relatively small area within a scene (Figure 2.6). A smaller number indicates low average radiance from the area while a high number is an indicator of high radiant properties of the area [32].

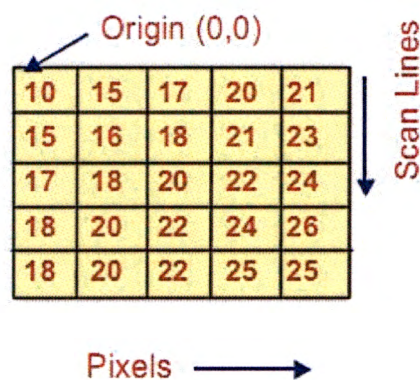


Figure 2-6 Structure of a digital image [32].

2.8 Concept of an Image

An image is a collection of numbers that constitute different light intensities in different areas of the image [33]. This numeric representation forms a grid and the individual points are referred to as pixels. Most of the images consist of a rectangular map of the image pixel known as the bits where each pixel is located and its colour. These pixels are displayed horizontally row by row. The number of bits in a colour scheme, called the bit depth, refers to the number of bits used for each pixel. The smallest bit depth in current color schemes is 8, meaning that there are 8 bits used to describe the color of each pixel [33]. Monochrome and grayscale images use 8 bits for each pixel and are able to display 256 different colors or shades of gray. Digital colour images are typically stored in 24bit files and use the RGB color model, also known as true color. All color variations for the pixels of 24 bit image are derived from three primary colors: red, green and blue, and each primary color is represented by 8 bits. Thus in one given pixel, there can be 256 different quantities of red, green and blue, adding up to more than 16 million combinations, resulting in more than 16 million colors. Not surprisingly the larger the number of colours that can be displayed, the larger the file size [33].

2.8.1 Image Representation

Let $C(x, y, t, \lambda)$ represent the spatial energy distribution of an image source of radiant energy at spatial coordinates (x, y) , at time t and wavelength λ . Because light intensity is a real positive quantity, that is, because intensity is proportional to the modulus squared of the electric field, the image light function is real and nonnegative [34]. Hence, it is assumed that

$$0 < C(x, y, t, \lambda) \leq A \quad (2.4)$$

where A is the maximum image intensity. A physical image is necessarily limited in extent by the imaging system and image recording media. For mathematical simplicity, all images are assumed to be nonzero only over a rectangular region for which

$$-L_x \leq x \leq L_x \quad (2.5)$$

$$-L_y \leq y \leq L_y \quad (2.6)$$

The physical image is, of course, observable only over some finite time interval.

$$-T \leq t \leq T \quad (2.7)$$

The image light function $C(x, y, t, \lambda)$ is, therefore, a bounded four-dimensional function with bounded independent variables. As a final restriction, it is assumed that the image function is continuous over its domain of definition.

The intensity response of a standard human observer to an image light function is commonly measured in terms of the instantaneous luminance of the light field as defined by

$$Y(x, y, t) = \int_0^\infty C(x, y, t, \lambda) V(\lambda) d\lambda \quad (2.8)$$

where $V(\lambda)$ represents the relative luminous efficiency function, that is, the spectral response of human vision. Similarly, the colour response of a standard observer is commonly measured in terms of a set of tristimulus values that are linearly proportional to the amounts of red, green, and blue light needed to match a coloured light. For an arbitrary red–green–blue coordinate system, the instantaneous tristimulus values are

$$R(x, y, t) = \int_0^\infty C(x, y, t, \lambda) R_S(\lambda) d\lambda \quad (2.9)$$

$$G(x, y, t) = \int_0^\infty C(x, y, t, \lambda) G_S(\lambda) d\lambda \quad (2.10)$$

$$B(x, y, t) = \int_0^\infty C(x, y, t, \lambda) B_S(\lambda) d\lambda \quad (2.11)$$

Where $R_S(\lambda)$, $G_S(\lambda)$, $B_S(\lambda)$ are spectral tristimulus values for the set of red, green, and blue primaries. The spectral tristimulus values are, in effect, the tristimulus values required to match a unit amount of narrowband light at wavelength λ . In a multispectral imaging system, the image field observed is modeled as a spectrally weighted integral of the image light function.

2.8.2 Image Processing

The majority of today's steganographic systems use images as cover media because people often transmit digital pictures over email and other Internet communications. Moreover, after digitalization, images contain the so-called quantization noise which provides space to embed data [34]. In digital image processing systems, one usually deals with arrays of numbers obtained by spatially sampling points of a physical image. After processing, another array of numbers is produced, and these numbers are then used to reconstruct a continuous image for viewing [35].

2.8.3 Image Compression

Image compression is a commonly used method for reducing the file size of an image, without reducing the aesthetic qualities enough to become noticeable to the naked eye. The compression of JPEG images contains several processes:

- i. Colour space transformation (Converting pixel values to YCbCr)
- ii. Downsampling the chrominance values
- iii. Transforming values to frequencies
- iv. Quantization
- v. Zig-Zag ordering
- vi. Lossless Compression

i. Colour Space Transformation

The first step is to convert the RGB colour layers of the image into three different components (Y, Cb, and Cr). The Y component relates to the luminance (brightness). The chrominance coefficients of an image (Cb and Cr) are determined by a 2D grid that has blue to yellow on one axis, and red to green on another.

ii. Downsampling the Chrominance Values

This phase is to downsample (or subsampling) the image. The human eye is more sensitive to changes in brightness than to changes in colour. This means that it is possible to remove a lot of colour information from an image without losing a great deal of quality. As a result of this, much of the compression takes place by downsampling the chrominance data to reduce the overall file size.

JPEG compression often downsamples by taking four adjacent pixels and averaging them to one value. Doing this for the entire image removes roughly half the file size. All of this can be done without any noticeable drop in the quality of the image.

iii. Transforming Values to Frequencies

The Discrete Cosine Transform (DCT) is used for JPEG images to transform them into frequencies. DCT is a mathematical transform (typically a cosine function) that converts the pixels by seemingly spreading the location of the pixel values over part of the image.

It does this by grouping the pixels into 8×8 blocks and transforming them from 64 values into 64 frequencies (DCT coefficients). By modifying just a single DCT coefficient, the entire 64 pixels in that block are affected.

iv. Quantization

The next step is arguably the most important one when compressing JPEG images. The aim is to quantize the values that represent the image after the previous stage. Quantization is the process of taking the remaining coefficients and dividing them individually against a pre-determined set of values and then rounding the results to the nearest real number value. The goal is to eliminate the high frequency (lower-right) values.

v. Zig-Zag Ordering

Most of the quantized DCT coefficients are fairly typical. There are only a few values that hold numbers other than zero - the majority will always be zeros. It is also typical to see the non-zero numbers in the upper left, and zeros as you get towards the lower-right corner. As this is the case, this stage of JPEG compression reorders the values using a 'zig-zag' type motion so that similar frequencies are grouped together.

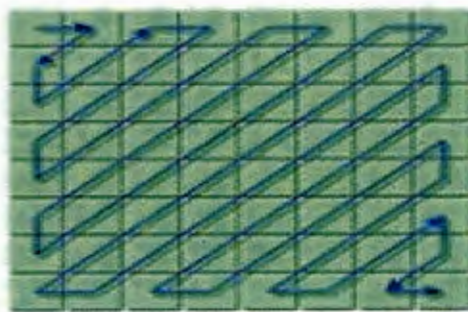


Figure 2-7 Zig-zag ordering of JPEG image components

2.9 Existing Steganography Techniques.

The authors in [14] discussed Information Hiding using Steganography. The implementation of their work focused on Least Significant Bits. In their work, the size of the image and the message must be defined by the system. This is important to ensure the image can support the message to be embedded. The ideal image size is 800×600 pixels, which can embed up to 60kB messages. This process simply embedded the message into the cover-image without supplying any password or stego-key.

At the sender side, the data is converted into bytes; that is, each character in the message is converted into its ASCII equivalent. Then the message bit is embedded in the digital image. Each pixel of the image typically has three numbers associated with it, one each for red, green and blue intensities, and their values often range from 0 – 255. In order to hide the message, the data is first converted into byte format and stored in a byte array. Then each bit is embedded into the LSB position of each pixel position [14].

The weakness in Information Hiding using Steganography is that an attacker can easily destroy the message by removing or zeroing the entire LSB plane, with very little change in the perceptual quality of the modified stego-image. This may cause someone to suspect that something is hidden in the stego-image.

The authors in [33] hides messages in the least significant bits of the 8-bit binary strings representing the colour numbers. Grayscale image considered here as the cover image. Each character in a message is converted into bits which denote the binary representation under the ASCII character system, which assigns characters with integer values between 0 and 255. In this work, each bit of the message is mapped to a single pixel of image. The remaining bytes from the image pixel are used to encode the length of the message. Hence in this work, to hide an N length message, the image must have $8 * (N + 16)$ pixels.

The limitation of this work is that the resulting image will still look mostly like the original, except that a few points seem a little out of place if it is examined very closely.

The authors in [36] implemented a model that combines cryptography and steganography along with an extra layer of security that was imposed in between them. Their work deals with the security of a text message at the time of sending it over the network. In their algorithm, they used asymmetric key cryptography, which means different keys are needed to encrypt and decrypt the data. They divided the domain of the key selection into different sub domains (a random prime number, a randomly generated number, decimal value of the pixel (only R) from the cover picture). The strength of their approach is the division of the domain together with the key length. They encrypted the original text message letter by letter, applying a function which involves certain mathematical operations using corresponding letters and also numbers from the original image. Then they used two public keys and one private key for encryption and decryption. These keys are generated randomly following some constraints and equations.

In their method, they used the code_matrix mapping method where the encrypted code is first broken digit by digit and converted into a binary matrix having size DP (Depth of the cover image). After

this they used the matrix_pix mapping method where the matrices obtained previously are mapped into a zone of pixels having area DP and then used LSB to modify the image pixel bytes. This work developed a steganography system with an introduction of another level of security layer that make the existing technique a stringent one.

The authors in [37] introduced a new method to steganography called White processing Picture in Three Colours (RGB). In their work, hiding a message involves embedding the message into a digital image. It uses the first pixel to hide the length of a message (number of characters). They use four colours in two pixels to store an 8 bits character. The first colour in first the pixel is: r7 r6 r5 r4 r3 r2 r1 r0, the second colour in the first pixel is: g7 g6 g5 g4 g3 g2 g1 g0, the third colour in the first pixel is: b7 b6 b5 b4 b3 b2 b1 b0, the first colour in the second pixel is: r7 r6 r5 r4 r3 r2 r1 r0.

Their algorithm measures the intensity of the pixel and then hides data by random pixel selection, with the goal of hiding the maximum data in each pixel without creating extra unnatural noise. To perform the operation and find pixels with higher intensity, they obtained average colour elements in this image. The number is a boundary to determine the elements with higher intensity. These are elements that have greater average colour intensity. Thus the intensity of pixels in the image which are selected and scatter in a pixel selection is created. In order to perform this algorithm, they find the total number of pixels, and that number of pixels is marked to determine the pixel with most intensity. This work created a cross platform that effectively hides a message inside a digital image file.

The authors in [38] discussed "A New Efficient Approach towards Steganography". In their work, every character in the text (message) is converted into its integer value, and that integer value is mapped to the single pixel value of the image. Index arrays are the only information required to recover the message back from the image. There is absolutely no change in the image quality because pixel value of the image is changed. They used MATLAB for their simulation. Encoding was done at the sender end using the encoding module and the information is sent to the receiver on the basis of which the receiver, using the decoding module, retrieves the original message from the key.

In their encoding procedure, first, they read the image file and then read the text file. The text is converted file into ASCII values. The procedure then checks the length of the message and saves it with variable name, it starts a loop and opens the image file in an array format. It will then search for the ASCII value in the matrix of the grey image by increasing the index location. When the character

is found in any location, it saves the location in a variable known as index. When the loop part is completed with length variable zero, it then finds the mean of all index values and saves it in the last position, after dividing the index values with mean. It now saves the index variable with 'info.mat'. Thus 'info.mat' file is used for the decoding process. Using an image as a key has the advantage that they can use an image of indefinite size.

The limitation of their work is that it only allows the use of a grey scale image as the cover object. Also the sender and receiver must use the same grey scale image.

The authors in [39] discussed the Hybrid Domain in LSB Steganography. The payload (message data) is embedded into the cover image by segmentation, and they used DCT/DWT to generate the stego image for secret information to be transported to the destination over the communication channel confidentially. Then the cover image (CI) and payload (PL) are divided into two cells of desired dimension (segmentation). The RGB components are separated in the cover image and payload cells in order to preserve the statistical characteristics of the colour image after manipulating CI and PL. The RGB components of the cover image cell I are individually transformed from spatial domain to frequency domain using DCT and DWT whereas the components of cell II of the cover image are being retained in spatial domain itself (transformation).

The embedding process requires the four MSB bits of each pixel in the payload cell 1 and cell 2 to be embedded in the second and fourth LSB positions of cover image cell I and cell II respectively to increase the security of the payload and generate the stego image in the transform domain. The stego image in the transform domain, after the payload is embedded into the cover image, is converted into spatial domain by changing cell I from transform domain into the spatial domain. The final stego image in spatial domain is obtained by combining cell I and cell II [39]. Their work provides a model that has better PSNR and security than existing techniques. The limitation of their work is that the message (payload) to be sent is always in image format.

The authors in [40] designed some Imperceptible Data Hiding methodologies using Steganographic Techniques. In their method, the message first encrypted gives an extra layer of protection. Then they proceeded to the cover image and secret data analysis where they checked the size of the data, and based on that they moved to different steps. Their main aim was to conceal a large amount of data with negligible visual changes of the cover image. They analyzed and found different portions of the cover image where they can directly store the data. This direct replacement of the pixel value to the corresponding secret data gives a freedom to store three characters in one pixel, whereas the LSB methods require nine pixels to store the same three characters. The portion selection can be

done either manually or through a selection technique. The manual selection of the image portion depends on the user. The marked area is taken as a good place for direct storing of the secret data. The alternative selection process starts with the formation of a histogram. The image histogram is a valuable tool used to view the intensity profile of a cover image. The histogram provides information about the contrast and overall intensity distribution of a cover image [40]. It develops a steganographic system that can embed large data with very small changes in the cover image.

The author in [41] discussed Moderate Bit Insertion for hiding crypto-data in digital image for steganography. In their work, message ciphering and embedding starts with reading the secret message character-wise starting with the first character from a saved text file. Then they encrypt each character into eight crypto-bits using flexible matrix, repeating this process for all characters of the saved text file to obtain a series of crypto-bits. Each pixel of the cover image is read starting from the pixel $Z(i,j)$ where $i = 1:256$ and $j = 1:256$. They now convert each pixel into the equivalent eight-bit binary number, and embed one cipher-bit of message into the fourth LSB of pixel of the cover image, repeating the last two steps until all the cipher message bits are embedded into the cover image [41].

Post pixel adjustment is done by checking if the encrypted embedded data bit is equal to the fourth LSB of pixel of cover image. If it is, then no pixel adjustment is required, and they go to next pixel. If the encrypted embedded data bit is one, not equal to the fourth LSB of pixel of the cover image, then they modify the pixel by $Z(i, 1 \text{ to } 3)$. If the encrypted embedded data bit is zero, not equal to the fourth LSB of pixel of cover image, then modify the pixel by $Z(i, 1 \text{ to } 3) = 1$. They developed a model with a new type of crypto-data hiding method with some distortion tolerance in moderate significant bit.

2.10 Two steganography applications: P2P and Quickstego

(a). P2P Steganography Application

This is a steganography application for hiding information in any image file. The scope of the project is the implementation of steganography tools for hiding information, including any type of information file and image files, and the path where the user wants to save the image and an extended file. The application was developed using C# and .Net framework [42].

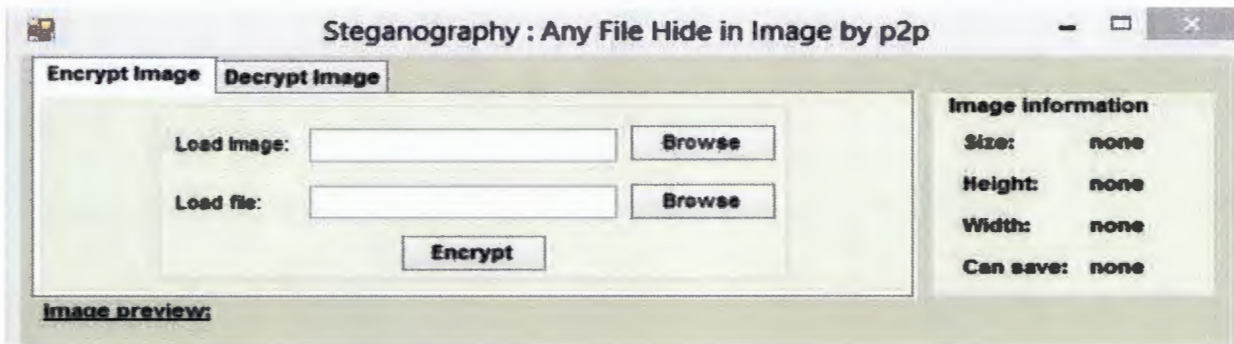


Figure 2-8 Snapshot of P2P application [42]

The algorithm used for Encryption and Decryption in the application was achieved using several layers instead of using only the LSB layer of image. Writing data starts from the last layer (8th or LSB layer) because the significance of this layer is the least, and every upper layer has double the significance of the layer beneath it. In order for a user to run the application, the user has two tab options – encrypt and decrypt. If the user select encrypt, the application displays the screen to select image file, information file and option to save the image file. If the user selects decrypt, the application displays the screen to select only image file and asks the path where the user wants to save the secret file [42].

2.10.1 QuickStego Steganography Application

This is a steganography application that allows users to hide text in pictures so that only other users of it can retrieve and read the hidden secret messages. Once text is hidden in an image, the saved picture is loaded just like any other image and appears as it did before. The image can be saved, emailed, or uploaded on the web .

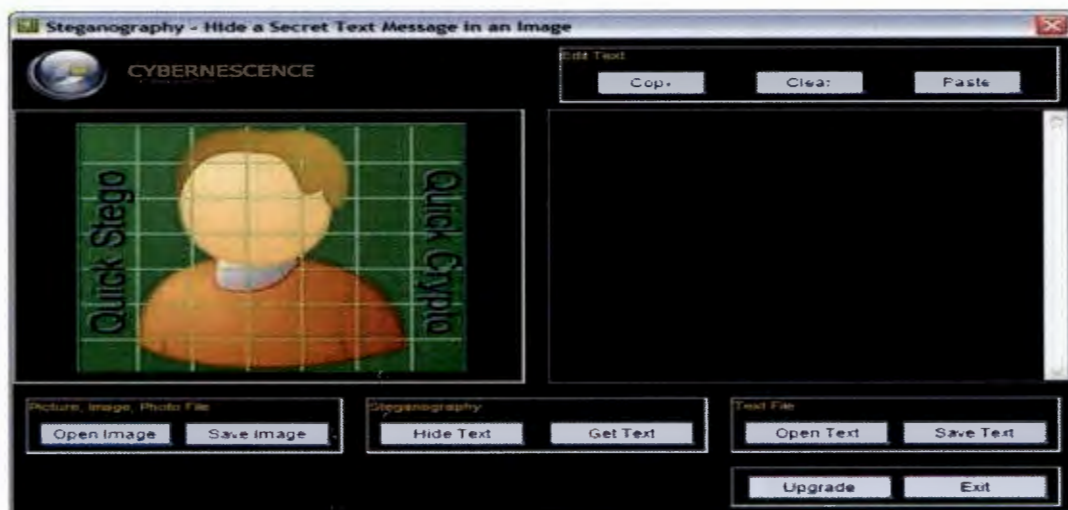


Figure 2-9 Snapshot of QuickStego steganography application [43].

The principle used in this application is hiding pure text under an image. The application enables a user to select a secret text file or copy its content into a textbox provided, after which the user will select the cover image. The text is hidden under the selected image and the stego image would be generated. The user can also save the resulting stego image [43].

2.11 Tools used in the Implementation of this Design

The tools required in the implementation of the steganographic design are .NET framework, and visual studio.

2.11.1 The .NET Framework

The .NET Framework provides a common set of services that application programs written in .NET language such as C# can use to run on various operating systems and hardware platforms. The .NET Framework is divided into two main components: the .NET Framework Class Library and the Common Language Runtime [44].

The .NET Framework Class Library consists of segments of pre-written code called classes that provide many of the functions needed for developing .NET applications. For instance, the window forms classes are used for developing window form applications. The ASP.NET classes are used for developing Web Forms applications, and other classes let us work with databases, manage security, access files and perform many other functions [44].

Although not apparent in Figure 2.10, the classes in the .NET framework class library are organized in a hierarchical structure. Within this structure, related classes are organized into groups called namespaces. Each namespace contains the classes used to support a particular function. For example, the *System.Window.Forms* namespace contains the classes used to create forms and *System.IO* contains the classes used for work Input-Output operations such as File and Streams operation.

The Common Language Runtime (CLR) provides the services that are needed for executing any application that is developed with one .NET languages. This is possible because all of the .NET languages are compiled to a common intermediate language. The CLR also provides the Common Type System that defines the data types that are used by all .NET languages [44].

The .NET Framework

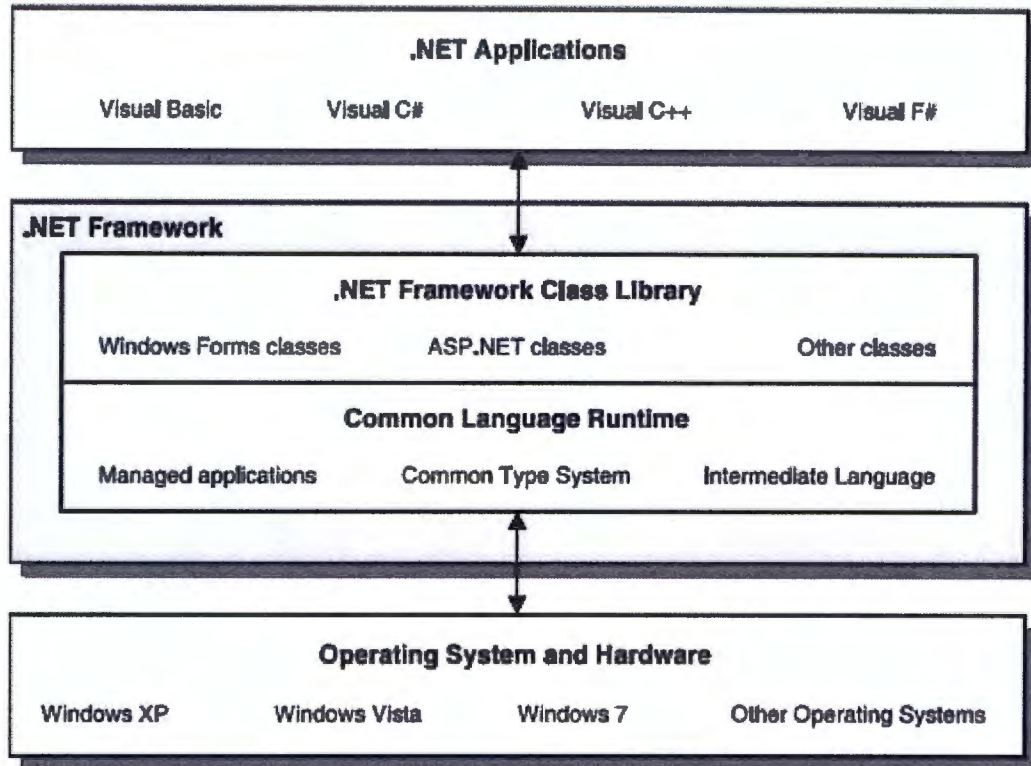


Figure 2-10 The .NET Framework [44].

2.11.2 Visual Studio

The Visual Studio is Microsoft's integrated programming environment. It lets programmers edit, compile, run, and debug a C# program, all without leaving its well thought-out environment. Visual Studio offers convenience and helps manage programs. It is most effective for larger projects, but it can be used to great success with smaller programs.

Visual Studio 2008 is a fully integrated development environment. It is designed to make the process of writing code, debugging it, and compiling it to an assembly language to be shipped as easy as possible. What this means is that Visual Studio gives a very sophisticated multiple - document - interface application in which one can do just about everything related to developing code. It offers these features:

The IDE allows a programmer you to edit, compile, and run a C# program and other programs written in any of .NET languages such as J#, C++.Net, VB.Net etc.

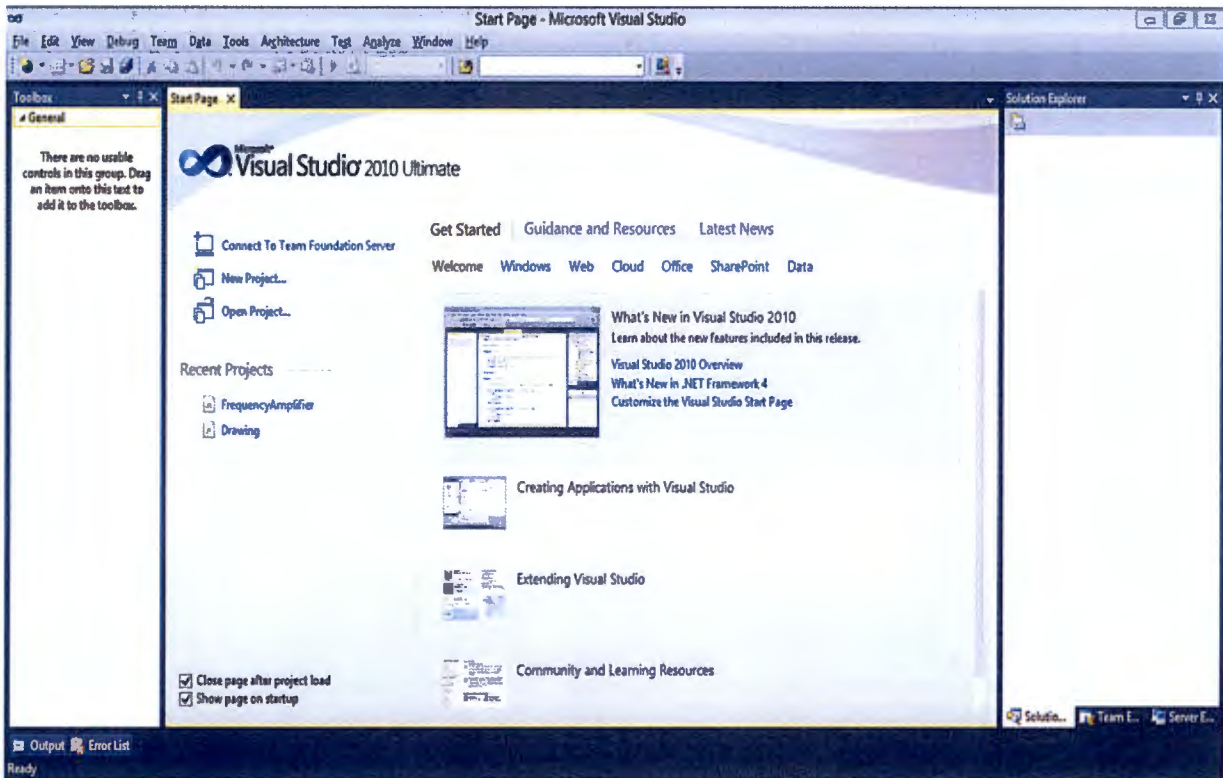


Figure 2-11 Visual Studio IDE

2.12 Chapter Summary

This chapter is a review of existing and related literature in security measures to safeguard data transmission and sharing over the Internet. It also explains the key concepts and terminologies as used in this project. Cryptography and steganography, which form fundamental technologies for the rising demand in Internet security, were also explicitly explained. Data encryption, compression and encoding, as well as the concept of image processing with regard to model design, and an overview of security measures of related work were also explained.

CHAPTER THREE

3.0

Research Methodology

3.1 Chapter Overview

This chapter discusses the system design by describing each component of the system in accordance with the function it performs. These are divided into four stages, which are: the input, message processing, message encoding and image encoding, and information concealment stages.

3.2 System Design and Architecture

Figure 3.1 shows the schematic representation of the proposed system which is categorized into two major phases: the sender phase and the receiver phase. In the sender phase, the system has the following categories or activities: (encryption), compression, encoding, and embedding; the end product here is a stego-image. In the receiver phase, the system has the following activities: retrieve, decode, decompress, and decrypt, and the end product here is the original (plaintext) message.

At the compression phase, the message that the sender intends to send is stored inside a file and then compressed using an appropriate compression algorithm to generate compressed information. This is done to reduce the size of the file being sent and also to increase the speed (rate) at which the message is delivered to the destination.

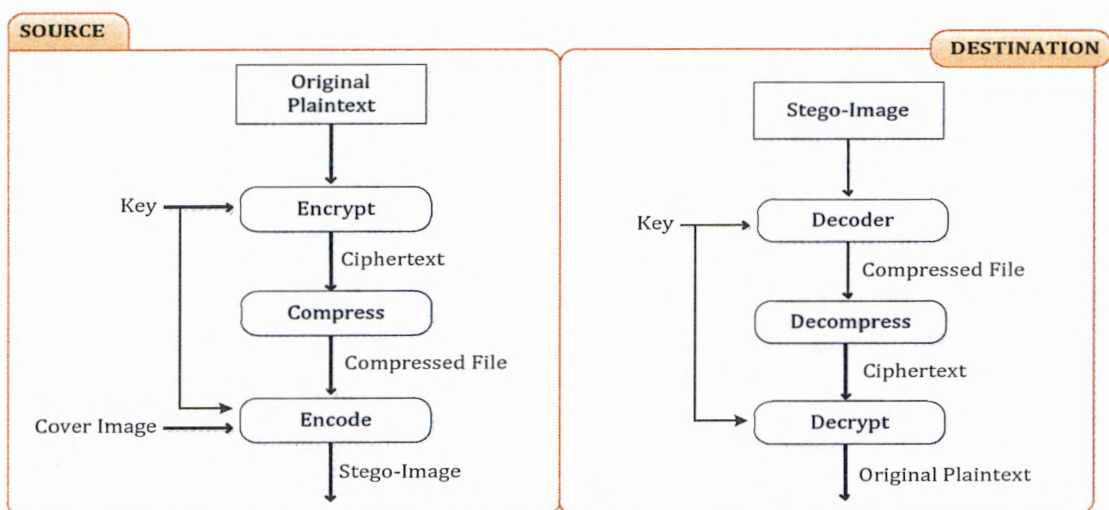


Figure 3-1 The system design of the Multi-Level steganography

3.3 System Model

Models are physical objects that can be used to represent anything. A model is a smaller or larger physical copy of an object. The object being modeled may be small or large. The multi-level

steganography system model can be represented with the schematic representation in Figure 3.1 and this section shall explain each system component one by one.

3.3.1 Message

This is the data or information that the user wants to send. It is in the form of a plain text or file. The average information content of a message is called its entropy (E). The entropy of a source is inversely proportional to the log of its probability of occurrence:

$$E = -\log P \quad (3.1)$$

For an element, e , in a message, M . Its entropy can be defined as:

$$E:e = -\log(P:e) \quad (3.2)$$

$P:e$ is the probability of an element in a message. It is equal to that element's frequency, $F:e$, divided by the frequency of the entire message, $F:M$:

$$P:e = F:\frac{e}{F}:M \quad (3.3)$$

The average information content or average entropy for a message, $E:M$, can now be defined. The entropy for each element in the message is $E:e$. We index the elements, e , in a message, M , by assigning them integer positions, i , according to their place in the message. For instance, the first element, e , in message M will be assigned 1, the second will be assigned 2, the i th will be assigned i . The entropy of the entire message, $E:M$, can now be written, given that there are i elements, as:

$$E:M = (P:1 * E:1) + (P:2 * E:2) + (P:3 * E:3) + \dots + (P:i * E:i) \quad (3.4)$$

For instance, in one time unit, a user (sender) can send an S symbol, which as far as information content is concerned, is equivalent to $s \log_2 n$ bits, let this be denoted by H , the amount of information, measured in bits transmitted in each time unit [45].

Since symbol a_i occurs a fraction P_i of the time in the data, it occurs on the average sP_i times each time unit, so its contribution to H is $sP_i \log_2 P_i$. The sum of the contributions of all n symbols to H is therefore

$$H = -s \sum_1^n P_i \log_2 P_i \quad (3.5)$$

H is the amount of information in bits that can be sent in one time unit. The amount of information contained in one base- n symbol is therefore H/s or $-\sum_1^n P_i \log_2 P_i$. This quantity is called the entropy of the data being transmitted.

3.3.2 Data Encryption Algorithm

Data encryption standard is a block cipher. It operates on a single chunk of data at a time, encrypting 64 bits (8 bytes) of plaintext to produce 64 bits of ciphertext. The key length is 56 bits, often expressed as an eight-character string with the extra bits used as a parity check. The algorithm has 19 distinct stages. The first stage reorders the bits of the 64-bit input block by applying a fixed permutation. The last stage is the exact inverse of this permutation. The stage penultimate to the last one exchanges the leftmost 32 bits with the rightmost 32 bits. The remaining 16 stages (called rounds) are functionally identical but take as an input a quantity computed from the key K_i and the old right half K_i , where i is the current round number. K_i is derived from the original 56-bit key passed as input to the algorithm. Figure 3.2 shows the overall process [46].

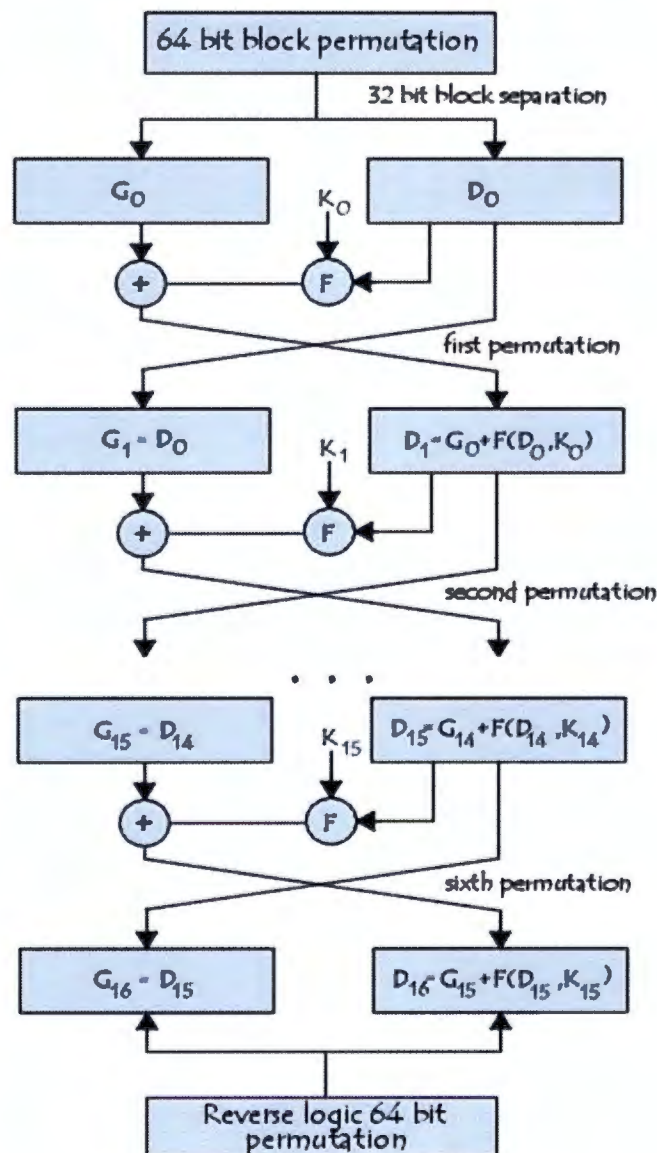


Figure 3-2 Phases in DES Algorithm [46].

At each iteration, the algorithm takes in two 32-bit inputs and produces two 32-bit outputs. The left output is a copy of the right input. The right output is an exclusive OR (XOR) of the left input and a function of the right input and the key for the stage K_i . All the complexity lies in the function f , which does a number of substitutions and permutations using simple hardware elements called S-boxes (for substitution) and P-boxes (for permutation). Decryption in the Data Encryption standard (DES) algorithm uses the same sequence of steps, but the keys used at each of the 16 stages (K_1 to K_{16}) are applied in reverse order [46].

If plaintext = initial 64 bits block of plaintext intended for encryption.

Let A be the string of text that has not undergone any permutations.

B is the string of text found by performing the permutation, IP on A .

$$B = IP(A) \quad (3.6)$$

We will be splitting the plaintext message in half, labeling these halves L_i and R_i respectively. We will be swapping these halves in an interactive method for $i = 1, 2, 3, \dots, 16$.

$L_0(32 \text{ bitblock}) = \text{left halves}$ of the initial 64 bit block of the plaintext after it has been initially permuted.

$R_0(32 \text{ bitblock}) = \text{right halves}$ of the initial 64 bit block of the plaintext after it has been initially permuted.

ROUNDS OR ITERATIONS

$i^{\text{th}} \text{round} = i^{\text{th}} \text{iteration}$ as described below.

$$L_i := R_{i-1} \quad \text{for } 1 \leq i \leq 16$$

$$R_i := L_{i-1} \oplus f(R_{i-1}, K_i) \quad \text{for } 1 \leq i \leq 16 \quad (\text{Both the function } f \text{ and the subkey } K_i \text{ are described.})$$

KEY AND THE 16 SUBKEYS DERIVED FROM IT

The encryption key (K) will be used to generate “subkeys” K_i that will be used in rounds or iterations

$$K := \text{Key}(64 \text{ bits})$$

$$= \{56 \text{ bits determining how the transformations of the text is to be carried out}\}$$

$$\cup \{8 \text{ bits use } f \text{ or parity}\}$$

For instance if the message A to be encrypted is “Multi Level Steganography System” and its hexadecimal format is $M = \dots “123456789ABCDEF” \dots$, we get the 64-bit block of text:

$$M = 0001 \ 0010 \ 0011 \ 0100 \ 0101 \ 0110 \ 0111 \ 1000 \ 1001 \ 1010 \ 1011 \ 1100 \ 1101 \ 1110 \ 1111,$$

Applying the Initial Permutation (IP) to the block of text, M , we get $IP =$
 $1100 \ 1100 \ 0000 \ 0000 \ 1100 \ 1100 \ 1111 \ 1111 \ 1 \ 32 \text{ bits} \ 111 \ 0000 \ 1010 \ 1010 \ 1111 \ 0000 \ 1010 \ 1010.$

Next divide the permuted block IP into a left half L_0 of, and a right half R_0 32 bits.

$$L_0 = 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111$$

$$R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$

Let K be the hexadecimal key $K = 133457799BBCDFF1$. This gives us as the binary key:

$$K = 00010011\ 00110100\ 01010111\ 01111001\ 10011011\ 10111100\ 11011111\ 11110001$$

The 64-bit key is permuted according to the table PC-1 to obtain 56-bit permutation $K+ = 1111000\ 0110011\ 0010101\ 0101111\ 0101010\ 1011001\ 1001111\ 0001111$.

Next, split this key into left and right halves, C_0 and D_0 , where each half has 28 bits to get $C_0 = 1111000\ 0110011\ 0010101\ 0101111$ and $D_0 = 0101010\ 1011001\ 1001111\ 0001111$.

With C_0 and D_0 defined, we now create sixteen blocks C_n and D_n , $1 \leq n \leq 16$. Each pair of blocks C_n and D_n is formed from the previous pair C_{n-1} and D_{n-1} , respectively. We now form the keys K_n , for $1 \leq n \leq 16$, by applying the permutation table PC-2 to each of the concatenated pairs $C_n D_n$. Each pair has 56 bits, but PC-2 only uses 48 of these. After which we have $K_1 \dots K_{16}$.

ITERATIVE FUNCTION

$$f = f(R_{i-1}, K_i): \{R_i \mid i \in [0, 15]\} \times \{K_i \mid i \in [1, 16]\} \rightarrow \{a \text{ subset of the initial plaintext}\}$$

= the function used to determine the iterative sequence R_i

Next in the f calculation, we XOR the output $E(R_{i-1})$ with the key K_n :

$$K(n) \oplus E(R_{i-1}) \tag{3.7}$$

To this point we have expanded R_{i-1} from 32 bits to 48 bits, using the selection table and XORed the result with the key K_n . We now have 48 bits, or eight groups of six bits.

We now use each group of six bits as addresses in tables called "S boxes". Each group of six bits will give us an address in a different S box

Let us write the previous result, which is 48 bits, in the form:

$$K(n) \oplus E(R_{i-1}) = B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 \tag{3.8}$$

Where each B_i is a group of six bits. We now calculate

$$S_1(B_1) S_2(B_2) S_3(B_3) S_4(B_4) S_5(B_5) S_6(B_6) S_7(B_7) S_8(B_8) \tag{3.9}$$

Where $S_i(B_i)$ refers to the output of the i th S box.

We now proceed through 16 iterations, for $1 \leq n \leq 16$, using a function F which operates on two blocks—a data block of 32 bits and a key K_n of 48 bits—to produce a block of 32 bits.

For instance, For $n = 1$, we have

$$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$$

$$L_1 = R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$

$$R_1 = L_0 + f(R_0, K_1) = 1110\ 1111\ 0100\ 1010\ 0110\ 0101\ 0100\ 0100$$

The final stage in the calculation of f is to do a permutation P of the S-box output to obtain the final value of f :

$$f = P(S_1(B_1)S_2(B_2) \dots S_8(B_8)) \quad (3.10)$$

If all 16 blocks are processed using the method defined previously, we get, on the 16th round,

$$L_{16} = 0100\ 0011\ 0100\ 0010\ 0011\ 0010\ 0011\ 0100$$

$$R_{16} = 0000\ 1010\ 0100\ 1100\ 1101\ 1001\ 1001\ 0101$$

reversing the order of these two blocks and applying the final permutation to

$$R_{16}L_{16} = 00001010\ 01001100\ 11011001\ 10010101\ 01000011\ 01000010\ 00110010\ 00110100$$

$$IP^{-1} = 10000101\ 11101000\ 00010011\ 01010100\ 00001111\ 00001010\ 10110100\ 00000101$$

which in hexadecimal format is `85E813540F0AB405`.

This is the encrypted form of $M = 0123456789ABCDEF$: namely, $C = 85E813540F0AB405$.

Decryption is the inverse of encryption and it follows the same steps as above, but reversing the order in which the subkeys are applied [47].

3.3.3 Compression

Data Compression shrinks down a file so that it takes up less space. This is desirable for data storage and data communication. Storage space on disks is expensive so a file which occupies less disk space is "cheaper" than an uncompressed file. Smaller files are also desirable for data communication, because the smaller a file is the faster it can be transmitted. A compressed file appears to increase the speed of data transfer than an uncompressed file.

All data is encoded which means that the data is originally a combination of elements, e , from some alphabet A . This combination of elements is a message, M . This message from the alphabet, A , is encoded into the binary alphabet, B . The string of bits, binary digits (0's and 1's), is the encoded data. So essentially encoding is just transferring a message, M , from the alphabet A into the alphabet B [20].

The Lempel Ziv algorithm constructs its dictionary on the fly, by only going through the data once. Suppose we have broken the string up into $c(n)$ phrases, where n is the length of the string. Each phrase is broken up into a reference to a previous phrase and a letter of our alphabet. The previous phrase can be represented by at most $\log_2 c(n)$ bits, since there are $c(n)$ phrases, and the letter can

be represented by at most $\log_2 \alpha$ bits, where α is the size of the alphabet [48]. We have thus used at most the following total bits in encoding.

$$c(n)(\log_2 c(n) + \log_2 \alpha) \quad (3.11)$$

Lempel-Ziv algorithms do not use too much memory for the dictionary. Thus, most implementations of Lempel-Ziv type algorithms have some maximum size for the dictionary. When it gets full, they will drop a little-used word from the dictionary and replace it with the current word [20]. This also helps the algorithm adapt to encode messages with changing characteristics. We need to use some deterministic algorithm for which word to drop, so that both the sender and the receiver will drop the same word. Now, quantities to show how well does the Lempel-Ziv algorithm work. First, how well Lempel Ziv Algorithm works in the worst case, and second, how well it works in the random case where each letter of the message is chosen uniformly and independently from a probability distribution, where the i th letter appears with probability p_i . In both cases, the compression is asymptotically optimal [20]. That is, in the worst case, the length of the encoded string of bits is $n + o(n)$. Since there is no way to compress all length- n strings to fewer than n bits, this can be counted as asymptotically optimal. In the second case, the source is compressed to length

$$H(p_1, p_2, \dots, p_\alpha)n + o(n) = n \sum_{i=1}^{\alpha} (-p_i \log_2 p_i) + o(n) \quad (3.12)$$

To do the worst case scenario, the maximum number of distinct phrases that a string of length n can be parsed into needs to be known. There are some strings which are clearly worst case strings. These are the ones in which the phrase is all possible strings of length at most k . The length of such string is

$$n_k = \sum_{j=k}^k j 2^j \quad (3.13)$$

If we let $c(n_k)$ be the number of distinct phrases in this string of length n_k , we get that

$$c(n_k) = \sum_{i=1}^k 2^i = 2^{k+1} - 2 \quad (3.14)$$

For n_k , we thus have

$$c(n_k) = 2^{k+1} - 2 \leq \frac{(k-1)2^{k+1}}{k-1} \leq \frac{n_k}{k-1} \quad (3.15)$$

Now, for an arbitrary length n , we can write $n = n_k + \Delta$. To get the case where $c(n)$ is largest, the first n_k bits can be parsed into $c(n_k)$ distinct phrases, containing all phrases of length at most k , and the remaining Δ bits can be parsed into phrases of length $k + 1$. This is clearly the most distinct phrases a string of length n can be parsed into [19], so we have that for a general string of length n , the number of phrases is

$$c(n) \leq \frac{n_k}{k-1} + \frac{\Delta}{(k+1)} \leq \frac{n_k + \Delta}{k+1} = \frac{n}{k+1} \leq \frac{n}{\log_2 c(n) - 3} \quad (3.16)$$

Now, we have that a general bit string is compressed to around $c(n) \log_2 c(n) + c(n)$ bits, and if we substitute $c(n) \leq \frac{n}{\log_2 c(n) - 3}$

We get

$$c(n) \log_2 c(n) + c(n) \leq n + 4c(n) = n + O\left(\frac{n}{\log_2 n}\right) \quad (3.17)$$

So asymptotically, LZ does not use much more than n bits for compressing any string of length n .

To show that in the case of random strings, the Lempel Ziv algorithm's compression rate asymptotically approaches the entropy [19], let us assume that we construct a sequence such that, at each position, we independently choose letter α_i with probability p_i . This gives us a sequence

$$x = \alpha_{x(1)} \alpha_{x(2)} \alpha_{x(3)} \dots \alpha_{x(n)} \quad (3.18)$$

If $Q(x)$ is defined to be the probability of seeing this sequence. That is,

$$Q(x) = \prod_{i=1}^n p_{x(i)} \quad (3.19)$$

Now, suppose x is broken into distinct phrases

$$x = y_1 y_2 y_3 \dots y_{c(n)} \quad (3.20)$$

Then we have

$$Q(x) = \prod_{i=1}^{c(n)} Q(y_i) \quad (3.21)$$

Now, let c_l be the number of phrases y_i of length l .

$$-\log Q(x) \geq \sum_l c_l \log c_l \quad (3.22)$$

Rewriting equation 3.21, we have

$$Q(x) = \prod_l \prod_{|y(i)|=l} Q(y_i) \quad (3.23)$$

Looking at the inner product, since the y_i with length l are all distinct, they are mutually independent events [48], so the probabilities $Q(y_i)$ sum to at most 1 as shown in equation 3.24.

$$\sum_{|y_i|=l} Q(y_i) \leq 1 \quad (3.24)$$

Now, there is an inequality that says that to maximize a product of k terms, given a fixed sum of these terms, the best thing to do is make all of these terms equal [19]. There are c_l terms $Q(y_i)$ with $|y_i| = l$. Setting them all equal, we have

$$\prod_{|y_i|=l} Q(y_i) \leq \left(\frac{1}{c_l}\right)^{c_l} \quad (3.25)$$

Taking logs, and adding over all phrase lengths l , we get Ziv's inequality [48].

$$-\log Q(x) \geq \sum_l c_l \log c_l \quad (3.26)$$

Suppose we have m_i occurrences of letter α_i in a string x . Then,

$$Q(x) = \prod_i p_i^{m_i} \quad (3.27)$$

By the law of large numbers, there are approximately np_i occurrences of letter i in a string x [48]. Taking logs, we get

$$\log Q(x) \cong np_i \log p_i = nH(p_1, p_2, p_3, \dots, p_\alpha) \quad (3.28)$$

This is the entropy which is the optimal compression ratio for this probability distribution, the compression ratio of the Lempel-Ziv algorithm was approximately $c(n) \log c(n)$. All that is needed to do is to show $\sum c_l \log c_l \approx c(n) \log c(n)$.

Abbreviating $c(n)$ by c , we have $\sum_l c_l = c$. This gives us

$$\sum c_l \log c_l = c \log c + c \sum_l \frac{c_l}{c} \log \frac{c_l}{c} \quad (3.29)$$

The last term is $c = -\sum_l \frac{c_l}{c} \log \frac{c_l}{c}$

This is the entropy of the probability distribution $\Pi_l = \frac{c_l}{c}$. The l are positive integers, and the expected value of l is

$$l = \sum_l l \frac{c_l}{c} \log \frac{n}{c} \quad (3.30)$$

Now, the maximum possible amount of entropy for a probability distribution Π_l on positive integers whose expected value is $\frac{n}{c}$ is $0 \left(\log \frac{n}{c} \right)$.

So we have

$$-\log Q(x) \geq c(n) \log c(n) - 0 \left(\log \frac{n}{c(n)} \right) \quad (3.31)$$

which shows that the compression (the right side) is approximately n times the entropy (the left side), and so is optimal.

These compression algorithms are lossless compression algorithms. This means that the reconstructed message is exactly the same as the original message.

3.4 Image Compression and Encoding

An image is an array of numbers that represent light intensities at various points, or pixels. These pixels make up the images raster data. When dealing with digital images for use with Steganography, 8-bit and 24-bit per pixel image files are typical. 8-bit images are a great format to use because of their relatively small size. The drawback is that only 256 possible colours can be used which can be a potential problem during encoding. However, in 24-bits image, large numbers of colours (over 16 millions) can be encoded, which can go well beyond the Human Visual System (HVS), which makes it very hard to detect a message embedded in such an image.

Uncompressed image data requires considerable storage capacity and transmission bandwidth. A common characteristic for most digital images is that the neighboring pixels are correlated and

contain redundant information. Therefore the most important task when compressing an image is to find less correlated and yet recognizable representation of the image.

Image compression aims at reducing the number of bits needed to represent an image by removing the spatial and spectral redundancies as much as possible. JPEG is a widely used form of lossy image compression that centers on the Discrete Cosine Transform (DCT). The DCT works by separating images into parts of different frequencies. During a step called Quantization, where part of the compression actually occurs, the less important frequencies are discarded, hence the use of the term “lossy.”

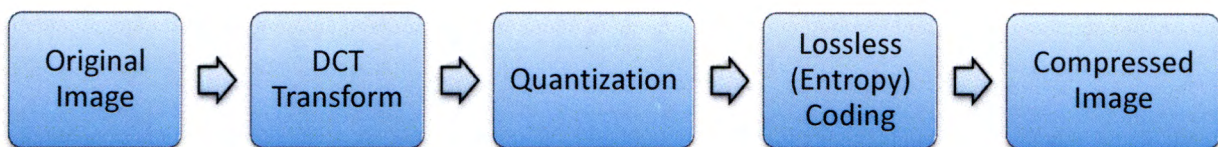


Figure 3-3 Image Compression Process

3.4.1 The Discrete Cosine Transform

The Discrete Cosine Transform (DCT) represents an image as a sum of sinusoids of varying magnitudes and frequencies. The DCT function computes the two-dimensional Discrete Cosine Transform (DCT) of an image. The DCT has the property that, for a typical image, most of the visually significant information about the image is concentrated in just a few coefficients of the DCT. For this reason, the DCT is often used in image compression applications.

The encoding process in image compression consists of several steps which are summarized below:

- i. The representation of the colours in the image is converted from RGB to $Y'C_B C_R$, consisting of one luma component (Y'), representing brightness, and two chroma components, (C_B and C_R), representing colour. This step is sometimes skipped.
- ii. The resolution of the chroma data is reduced, usually by a factor of 2. This reflects the fact that the eye is less sensitive to fine colour details than to fine brightness details.
- iii. The image is split into blocks of 8×8 pixels, and for each block, each of the Y , C_B , and C_R data undergoes a Discrete Cosine Transform (DCT).
- iv. The amplitudes of the frequency components are quantized. Human vision is much more sensitive to small variations in color or brightness over large areas than to the strength of

high-frequency brightness variations. Therefore, the magnitudes of the high-frequency components are stored with a lower accuracy than the low-frequency components.

- v. The resulting data for all 8×8 blocks is further compressed with a lossless algorithm, a variant of Huffman encoding.

The two-dimensional DCT of an M -by- N matrix A is defined as follows:

$$G_{u,v} = \alpha_u \alpha_v \sum_{y=0}^{M-1} \sum_{x=0}^{N-1} g_{x,y} \cos \left[\frac{\pi(2x+1)u}{2M} \right] \cos \left[\frac{\pi(2y+1)v}{2N} \right], \quad \begin{matrix} 0 \leq u \leq M-1 \\ 0 \leq v \leq N-1 \end{matrix} \quad (3.32)$$

$$\text{where } \alpha_u = \begin{cases} 1/\sqrt{M}, & u = 0 \\ \sqrt{2/M}, & 1 \leq u \leq M-1 \end{cases} \quad \text{and} \quad \alpha_v = \begin{cases} 1/\sqrt{N}, & v = 0 \\ \sqrt{2/N}, & 1 \leq v \leq N-1 \end{cases}$$

where u is the horizontal spatial frequency, for integers $0 \leq u \leq 8$,

v is the vertical spatial frequency, for the integers $0 \leq v \leq 8$,

$g_{x,y}$ is the pixel value at coordinate (x,y) , $G_{u,v}$ is the DCT coefficient at coordinates (u,v)

The reason for using DCT is that it has the ability to deal with the boundary coefficients. In this work, we apply the DCT to each distinct 8×8 block of the 2D image.

3.4.2 Quantization

A typical quantization matrix, as specified in the original JPEG Standard is shown in Figure 3.4:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

Figure 3-4 Quantization equation

The quantized DCT coefficients are computed using equation (3.33)

$$B_{j,k} = \text{round} \left(\frac{G_{j,k}}{Q_{j,k}} \right) \quad (3.33)$$

for $j = 0, 1, 2, \dots, 7$ and $k = 0, 1, 2, \dots, 7$

where G is the unquantized DCT coefficient; Q is the quantized equation above;

B is the quantized DCT coefficient

For two-dimensional inverse DCT, which is given by:

$$f_{u,v} = \alpha_u \alpha_v \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F_{u,v} \cos \left[\frac{\pi(2x+1)u}{2M} \right] \cos \left[\frac{\pi(2y+1)v}{2N} \right], \quad \begin{matrix} 0 \leq u \leq M-1 \\ 0 \leq v \leq N-1 \end{matrix} \quad (3.34)$$

where x is the pixel row, for the integers $0 \leq x \leq 8$,

y is the pixel row, for the integers $0 \leq y \leq 8$,

$\alpha(u)$ is defined, for the integers $0 \leq u \leq 8$,

$F_{u,v}$ is the reconstructed approximate coefficient at coordinate (u, v) ,

$f_{u,v}$ is there constructed pixel value at coordinate (x, y) ,

3.5 Information Concealment in an Image

Coding secret messages in digital images is the most widely used of all methods in the digital world of steganography today? This is because it can take advantage of the limited power of the Human Visual System (HVS). Almost any plain text, cipher text, image and any other media that can be encoded into a bit stream can be hidden in a digital image.

Just before the information concealment stage, the system requires a user to supply a message file which should be a text file, a secret key (this key is shared by both the sender and the receiver; the sender encrypts the message with this key and sends it to receiver. The receiver is expected to supply the exact same key to extract the message from the image, decrypt and read the message), and the image file (which can be in JPEG format). The user may also specify if the message file is to be compressed in order to reduce its size and to hide more information in an image.

After all the inputs have been collected by the system, it goes to the message processing and encoding stage. In this stage, the text message file is encrypted with the key supplied using the Data Encryption Standard (DES) algorithm to generate the ciphertext. The ciphertext file may be compressed to shrink down a file so that it takes up less space in the cover image. In the message encoding stage, the message is encoded in order to improve the security of the system. Here, a constant is generated which is gotten from the system date (that is the Exclusive OR of day and month) as shown in equation (3.35). The key and the message are converted into their bits equivalence respectively and arranged in groups of eight bits (a byte), each group is encoded using the constant bits.

$$\text{Constant C} = (\text{Day bit}) \text{OR} (\text{Month Bit}) \quad (3.35)$$

A digital image is a rectangular grid of pixels or "picture elements", which are represented as arrays of values (bytes) in a computer's memory. These values represent the intensities of the three colours R (Red), G (Green) and B (Blue), where a value for each of the three colours describes a pixel [49].

During the process of information concealment, the capacity of the JPG image file is calculated as shown in equation (3.36) which determine if the image is large enough to hide the information.

$$Capacity = DCT - (DCT_1 + DCT_0) \quad (3.36)$$

where DCT is the number of DCT coefficients, DCT_0 is the zero of DCT coefficients and DCT_1 is the number of DCT coefficient with value 1

The process of message insertion starts with inserting the secret key bit in the image, after which the message is inserted. The proposed approach scans through each pixel (each pixel has R, G, and B components) that make up the image, and the blue object of each pixel component selected is replaced with the message bit. The blue object of the first pixel component is used to hide the secret key bits and the blue object of the next pixel component is used to hide the constant bits which serves as the terminator. The essence of this approach is that, at the receiver end, a secret key is first collected and compared with the one stored in the image. If the keys match, the process proceeds to message retrieval and if it fails, it does not [49].

Each pixel that made up the cover image has red (R), green (G) and blue (B) components and each of these components has 8-bits (since 24 bits per pixel image is being used). For instance, let us say the message to be hidden is grouped in 8 bits 10100101, 11001101, 10110101 . If the first three pixels of the cover image in RGB components are (11001100 10110100 01100111), (10110101 10001100 01010010), and (10010011 11001001 01001001). The proposed model replaces the blue object of each components with each group of the message bits and yields (11001100 10110100 10100101), (10110101 10001100 11001101), and (10010011 11001001 10110101). Each of these replaced blue objects is encoded with the green object of the pixel components as illustrated in equation (3.37) which makeup the stego bits.

$$stego\ bit = (Blue\ bit)OR\ (Green\ Bit) \quad (3.37)$$

Then the resulting stego pixels components will be (11001100 10110100 00010001), (10110101 10001100 01000001), and (10010011 11001001 01111100).

If the cover image is an *image* in the algorithm, *MsgFile* represent the message in group of 8 bits and *Byte_Len* is the number of groups in the message. Algorithm 1 summarizes the procedure used in information concealment and Algorithm 2 shows the procedure used in retrieving the information.

Algorithm 1: Embedding algorithm

Start

```
Read Message, Key, Image;
Msg_Enc = Encrypt (Message, Key);
const = XOR(Day, Month);
c = GetByte(const);
Msg_Cmp = Compress (Msg_Enc); isCompressed = true;
IF (isCompressed == true)
    Msg = Msg_Cmp;
else
    Msg = Msg_Enc;
End If
Img_Capacity = Size(Image);
byte[] msgFile; i=0;
msgFile[i++] = c; // Terminator
while ((byteRead = Key.ReadByte() != -1)
{
    msgFile[i++] = byteRead XOR c;
}
msgFile[i++] = c;
while ((byteRead = Message.ReadByte() != -1)
{
    msgFile[i++] = byteRead;
}
msgFile[i++] = c;
Msg_Size = Size(msgFile);
If (Img_Capacity > Msg_Size)
    Img_Pxl[row, column] = ExtractPixel(Image);
    Byte_Len = MsgFile.Length;
    len = 0;
    for (i=0; i<Height; && (len < Byte_Len); i++)
    {
        for (j=0; j<Width; j++)
```

```

        {
            pixel = Image.GetPixel[j, i];
            r = pixel.Red;
            g = pixel.Green;
            b = pixel.Blue;
            msgBit = (byte) ((MsgFile[len] & (1 << j)) >> j);
            b = (byte)(msgBit == 1 ? b | (1 << 0) : b & ~(1 << 0));
            len++;
        }
    }
End If

```

End

Algorithm 2: Embedding algorithm

Start

```

Read stegoImage, Key;
kByte = ReadByte(Key);
cPixel = stegoImage.GetPixel[0, 0]; // Retrieve Terminator i.e constant bits
cb = cPixel.Blue;
bit = (byte) (( cb & (1 << 0)) >> 0);
b = (byte)(bit == 1 ? b | (1 << j) : b & ~(1 << j));
cbit = b;
int k = 0;
byte[] keyRetrieved, msgRetrieved;
for (i=0; i<Height; i++)
{
    for (j=0; j<Width; j++)
    {
        pixel = Image.GetPixel[j, i];
        r = pixel.Red;
        g = pixel.Green;
        b = pixel.Blue;
        bit = (byte) (( cb & (1 << 0)) >> 0);
    }
}

```

```

    b = (byte)(bit == 1 ? b | (1 << j) : b & ~(1 << j));
    If (b == cBit)
        count += 1;
        isTerm = true;
    Else
        isTerm = false;
    End If
    If (count == 1 && isTerm == false)
        keyRetrieved[k++] = b;
    End If
    If (count == 2 && isTerm == true)
        k = 0;
        If (kByte != keyRetrieved)
            STOP; Display Error;
        End If
    Else If (count == 2 && isTerm == false)
        msgRetrieved[k++] = c XOR cBit;
    Else
        Break Loop;
    End If
}
}
msg = OutputMessage(msgRetrieved);
If (msg is Compressed)
    Decompress;
End If
msg_Decr = Decrypt (msg, keyRetrieved);

```

End

3.6 Steganography Application

The Steganography application will allow the user to load an image into the system for steganography encoding, supply the message file, encrypt the file content using a specified key and

compress the file if required. Once the user has supplied all these parameters, the stego-Encoder takes over during the process of sending the message via the communication channel (Internet).

3.6.1 Stego-Encoder and Stego-Decoder

This is the section that show the steganography process at the sender end and decodes the process at the destination end. Stego-encode as in Figure 3.5 collects user information such as the message file, image file and the secret key to be used for the message encryption and performs the necessary action depending on the user requirement. The encoder encrypts the message file content using the key provided, then may compress the file before encoding the message under the original image using the key supplied to form the stego-image and then sends it via the communication medium to the destination.

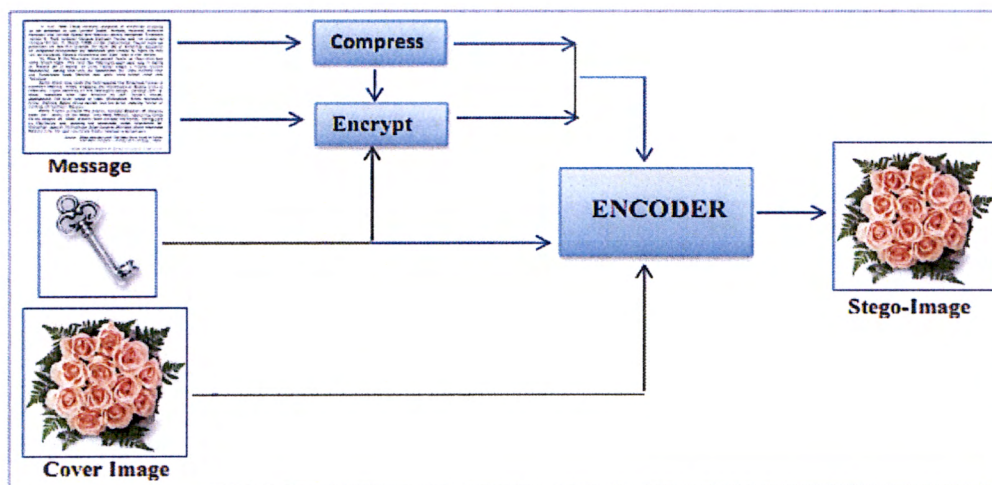


Figure 3-5 The Steganography Encoder Architecture

The Stego-decoder as in Figure 3.6 on the other hand, decodes the stego-image file using the same key as specified by the sender to get the message file, decompress the message file if need be and then decrypt it with the suitable key to get the message content.

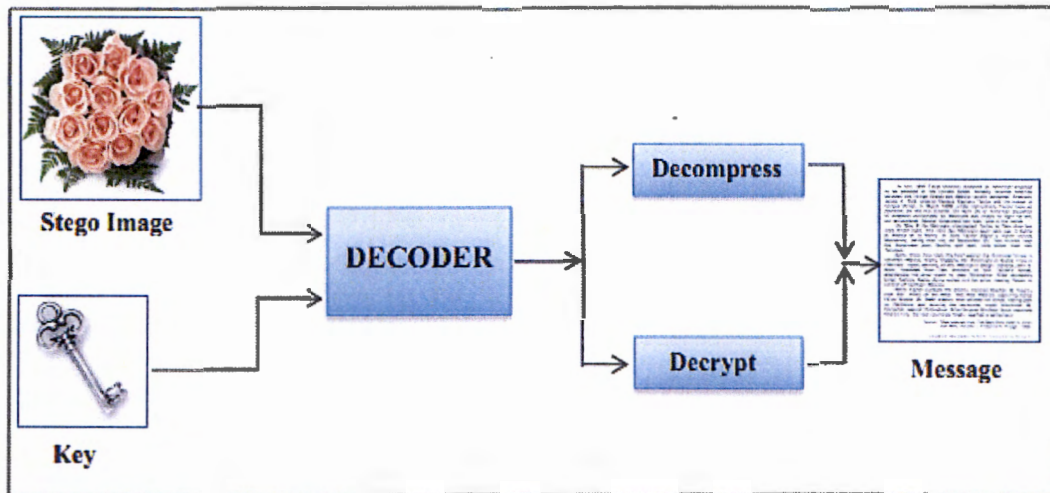


Figure 3-6 The steganography decoder architecture

3.7 Data Modeling

The purpose of data design is to show how the individual parts of the system will interact with each other. After analyzing the various models in this chapter, this section details the use case diagrams and class diagrams required for the system.

3.7.1 The Use Case Diagram

The use case diagram is used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The use case diagram shows which actors interact with each use case. A use case diagram captures the functional aspects of a system.

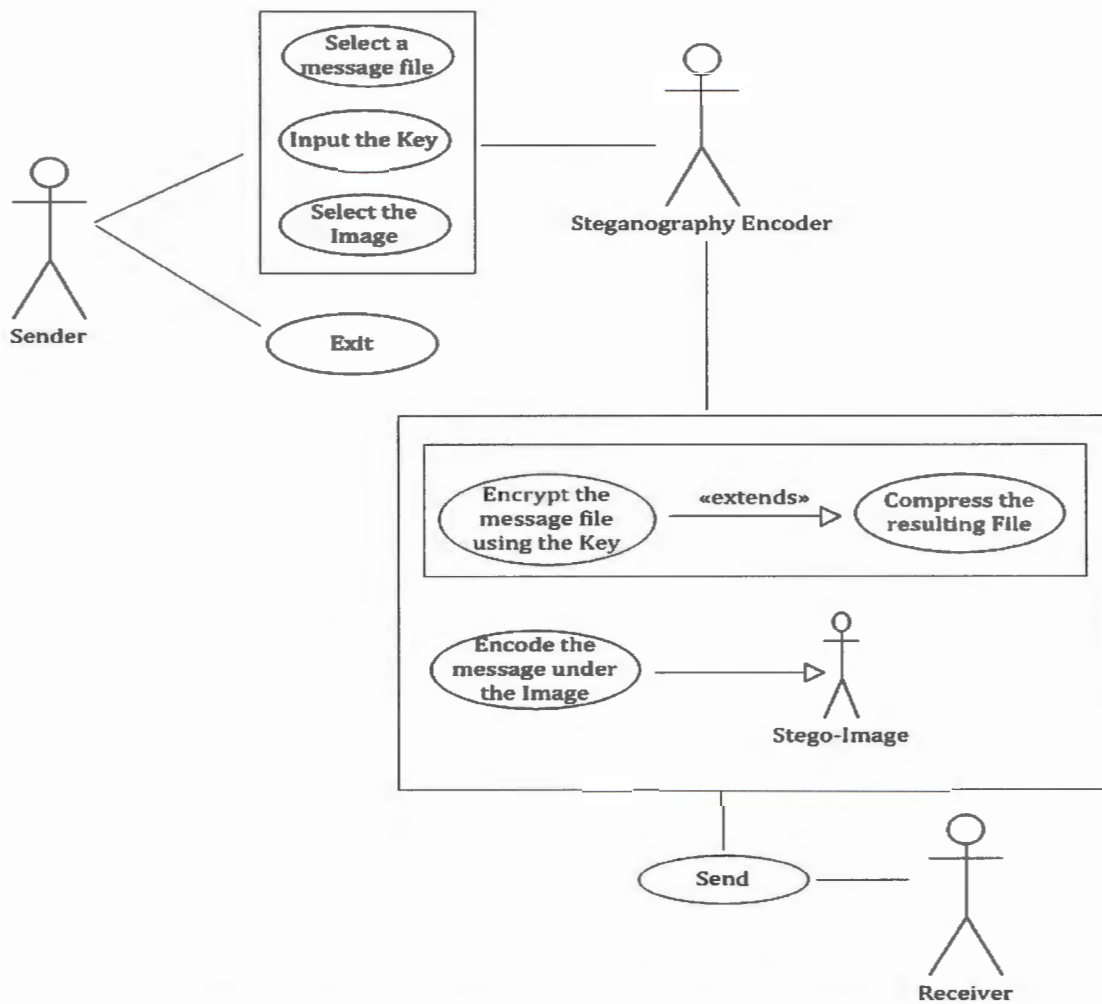


Figure 3-7 Use case diagram representing the sender end of System

Figure 3.7 represents all actions taken by a user (sender) when he tries to send a hidden message to another user (receiver) using the steganography system. Within the steganography application, the user must specify a message file and enter an encryption key and an image and submit it to the steganography encoder for processing. The steganography encoder then encodes the message and embeds it in the image before being forwarding it to the receiver.

In Figure 3.8, after receiving the stego image the receiver applies the key (which must be equivalent to the one used by the sender) to decode it in order to get the original message.

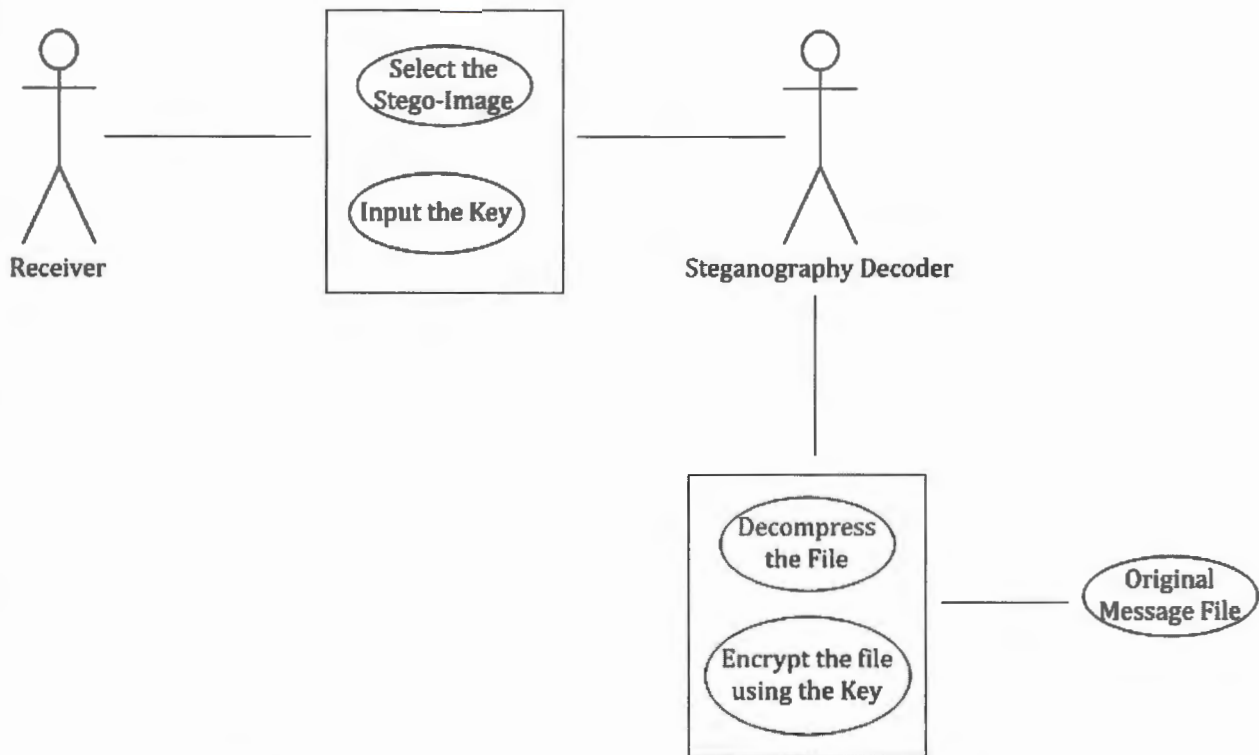


Figure 3-8 Use case diagram representing the receiver end of the system

3.7.2 Class Diagram

A class diagram is a diagram showing a collection of classes and interfaces, along with the collaborations and relationships among classes and interfaces. A class diagram is a pictorial representation of the detailed system design. Figure 3.9 and Figure 3.10 illustrate the class diagram for the overall system.

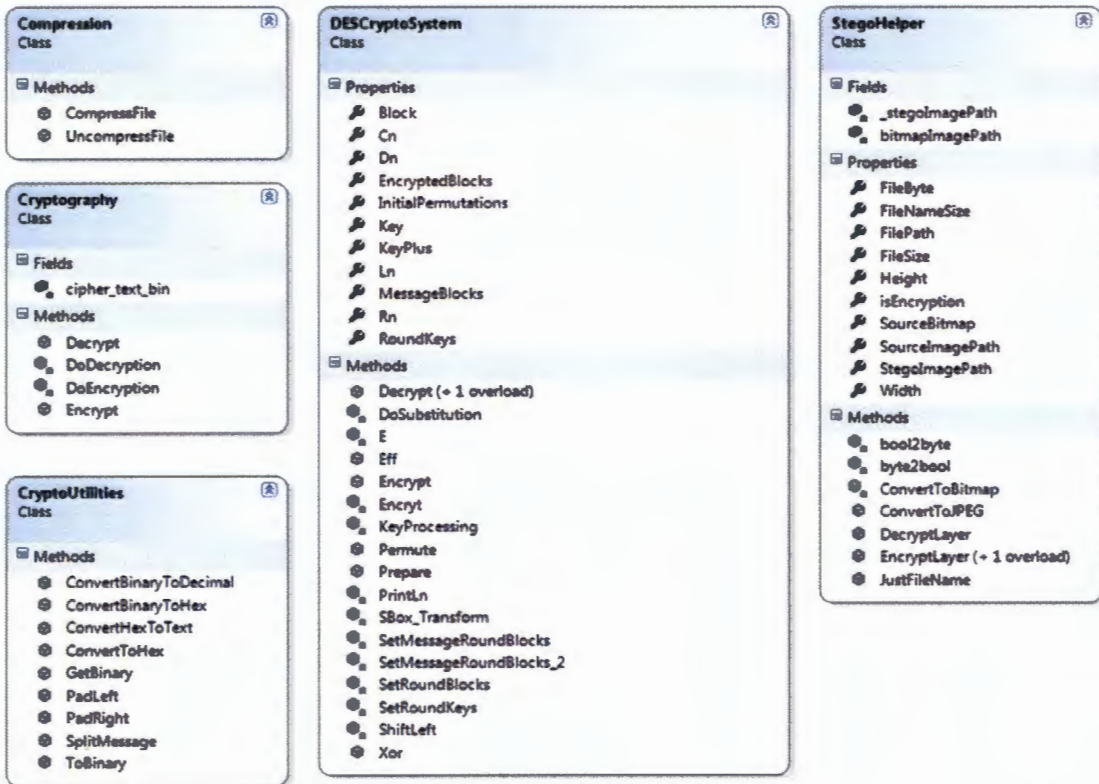


Figure 3-9 Class diagram of the steganography system

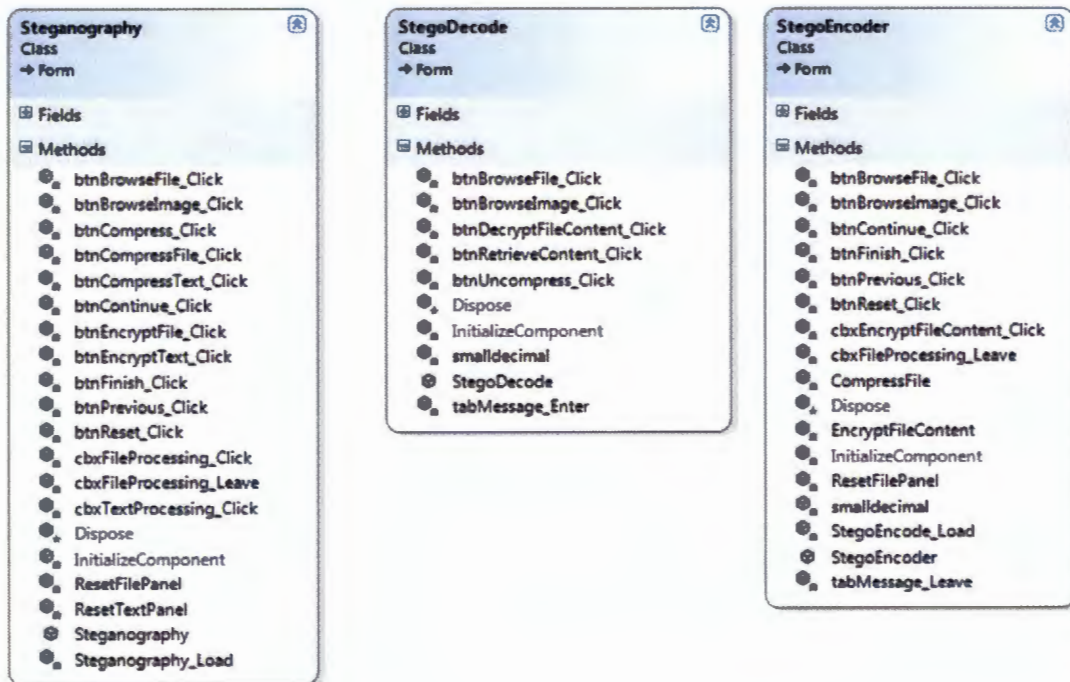


Figure 3-10 Class diagram of the steganography system

3.8 Quality of Reconstructed Image

Peak Signal-to-Noise Ratio (PSNR) measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is usually expressed in terms of the logarithmic decibel scale.

PSNR is used to measure the quality of reconstruction of lossy compression codes (for example, for image compression). The signal in this case is the original image, and the noise is the error introduced by compression. When comparing compression codec's, PSNR is an approximation to human perception of reconstruction quality. Although a higher PSNR generally indicates that the reconstruction is of higher quality, in some cases it may not. One has to be extremely careful with the range of validity of this metric. It is only conclusively valid when it is used to compare results from the same codec (or codec type) and same content.

PSNR is most easily defined via the Mean Squared Error (MSE). Given a noise-free $m \times n$ image I and its noisy approximation K , MSE is defined as:

$$MSE = \frac{1}{m.n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \quad (3.38)$$

PSNR is defined as:

$$PSNR = 10 \cdot \log_{10} \frac{MAX_I^2}{MSE} \quad (3.39)$$

$$PSNR = 20 \cdot \log_{10} \frac{MAX_I}{\sqrt{MSE}} \quad (3.40)$$

MAX_I is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255. For colour images with three RGB values per pixel, the definition of PSNR is the same except the MSE is the sum over all squared value differences divided by image size and by three. Alternately, for colour images the image is converted to a different colour space and PSNR is reported against each channel of that colour space.

3.9 Summary

This chapter discussed the proposed techniques in increasing the amount of data that can be hidden and the proposed secured multi-level steganography model difficult to detect, remove or modify data by an intruder. The next chapter will implement the experimental design explained in this chapter and show results.

CHAPTER FOUR

4.0 System Implementation

Implementation can be described as putting something into effect or action. This chapter discusses the implementation of the steganographic model described in Chapter Three. Result produced the models using a simulation program developed to test and support the theory of the model, and carry out analysis of the output. The chapter also uses various software tools uses for the implementation of the system.

4.1 System Architecture

Large systems are always decomposed into sub-systems and what constitutes these subsystem are some related set of services. The initial design process of identifying these sub-systems and establishing a framework for sub-system control and communication is called architectural design. Architectural design is concerned with establishing a basic framework which identifies the components of a system and the communications between these components. The architecture of the Steganography system consists of three main components; the Stego-Encoder, Stego-Decoder and an Internet connection or communication channel. Each component has an individual role within the system. The system is illustrated in Figure 4.1

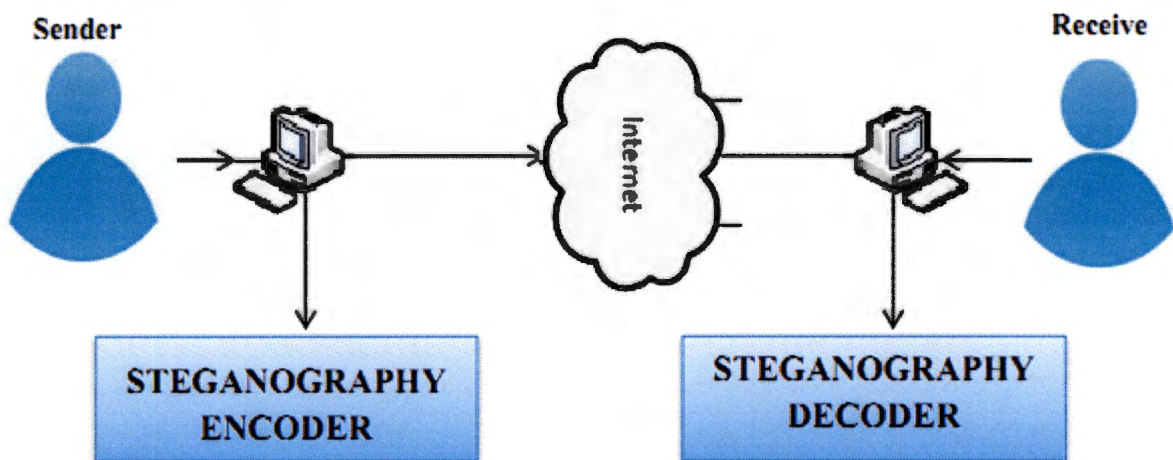


Figure 4-1 The Architecture of Steganographic System

4.2 Interface Design

Interface design is another important phase within system design. The interface of a system should be simple and easy to use by both less experienced and more experienced users. Figure 4.2 illustrates the main interface design for the Steganography System.

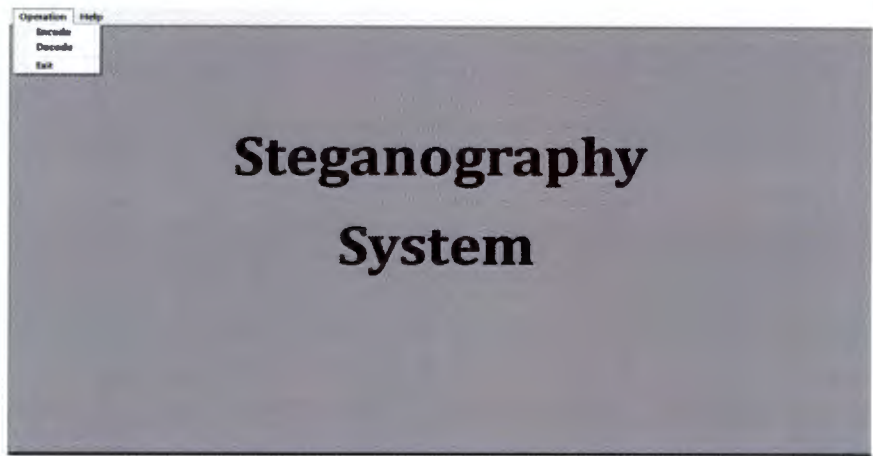


Figure 4-2 Interface design of the Steganography system

4.2.1 Message Processing Section

In the message processing section (as shown in Figure 4.3), the user selects the file that contains the message and then enters the secret key to be used for encrypting the message, and may compress the message file, depending on its size. The section processes the content of the message for encoding.

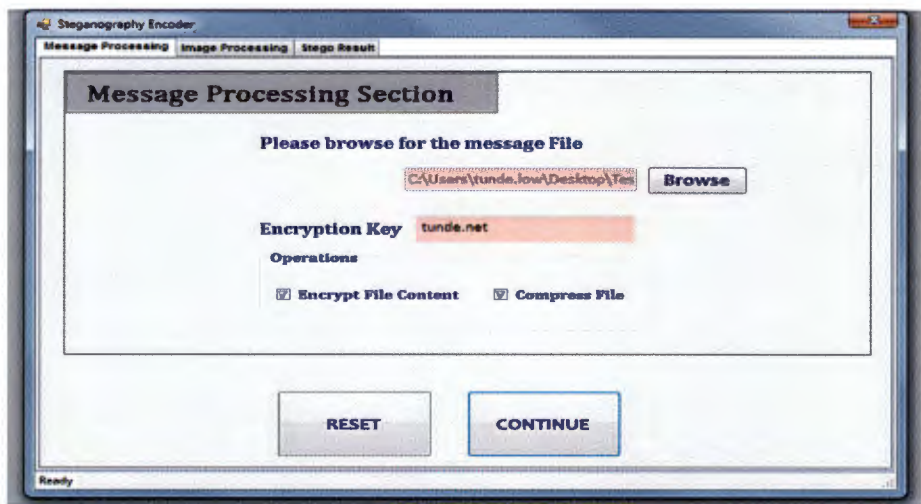


Figure 4-3 Message processing section of the Steganography system

4.2.2 Image Processing Section

This section deals with the processing of the cover image (as shown in Figure 4.4). The user selects the image file to be used for encoding; the size, height, width, and name of the image are displayed. After this the 'Finish' button, can be pressed to perform the steganography processes. On clicking the 'Finish' button the message file content is embedded under the image and the resulting stego-image is generated and displayed as shown in Figure 4.5



Figure 4-4 Image processing section of the Steganography System

Figures 4.6 and 4.7 show how the receiver can use the system to retrieve the embedded message in the stego image. Here, the user selects the stego-image as shown in Figure 4.6 and enters the secret key (the one provided by the sender) as shown in Figure 4.7. The content of the message will be retrieved if the key provided is correct. After the message has been retrieved, it is decrypted using the key provided. The system then indicates whether the retrieved file was compressed by the sender.



Figure 4-5 Stego Image section of the Steganography System



Figure 4-6 Stego Image processing section of the Steganography system



Figure 4-7 Message processing and retrieving section of the Steganography System

4.3 Experimental Setup and Results

a. Experimental setup

In this research, hydrangea.jpg, jellyfish.jpg, and lena.jpg were used as the cover images, although other JPEG images can also be used. These images were used to hide a range of secret text data which were stored in text files. To test the procedure, the secret text data was stored in 'stegomessage.txt'. The secret key used to encrypt the text was 'Stegonet', and it yielded a cipher-text 'cipher_stego message' file.

b. Experimental Results

The stego images remained unchanged after implementing the proposed approach and the amount of information that the proposed approach can hide is very high. This is a great advantage of using this technique for steganography. Table 4.1 describes the sizes of the text file and the stego images when tested with cover images such as hydrangea.jpg, jellyfish.jpg and lena.jpg. The initial sizes of the images were 28.43, 19.15 and, 69.14 kB respectively (as shown in Figure 4.8) while Table 4.2 shows their respective sizes when the data was compressed before hiding. Graphs depicting the information in Tables 4.1 and Table 4.2 are shown in Figures 4.9 and 4.10, respectively. The

horizontal axis represents the size of the text file while the vertical axis shows size of the stego images. The approach used in hiding information in the cover image makes it very secure.

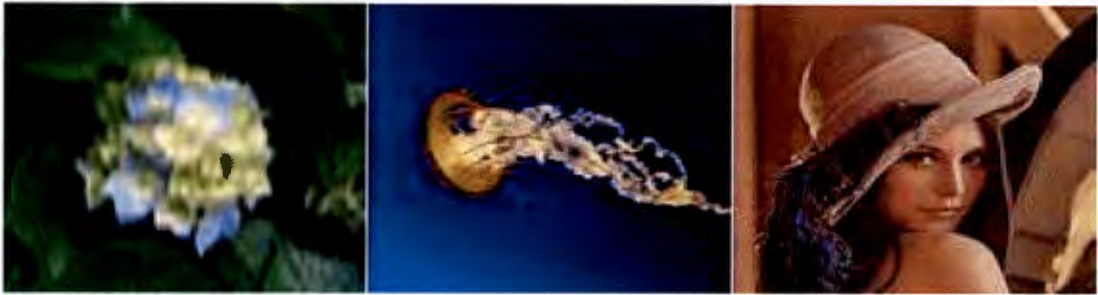


Figure 4-8 Images showing Hydrangea, Jellyfish and Lena

Table 4-1 Size of the text file and the stego images

Plaintext Size (kB)	Hydrangea (kB)	Jellyfish (kB)	Lena (kB)
6.20	29.29	19.75	39.76
12.4	29.29	19.75	39.76
20.60	29.2	19.79	39.75
28.7	29.30	19.77	39.40
41.30	29.30	19.81	39.75
57.40	29.31	19.80	39.80
70.4	29.30	19.80	39.80
82.7	29.37	19.84	39.76
93.8	29.36	19.87	39.76
113	29.45	20.01	39.77
134	29.57	20.19	39.72
153	29.91	20.47	39.64
165	30.14	20.78	39.69
186	30.24	20.98	39.81
207	31.80	23.35	39.92
240	34.00	26.33	40.14
295	40.21	33.83	42.27

Table 4-2 Size of the text file, compressed file and the stego images

Plaintext Size (kB)	Compressed File Size	Hydrangea (KB)	Jellyfish (kB)	Lena (kB)
6.20	499 b	29.27	19.72	39.77
12.4	512 b	29.27	19.72	39.77
20.60	578 b	29.27	19.72	39.75
28.7	655 b	29.27	19.72	39.77
41.30	735 b	29.28	19.72	39.75
57.40	882 b	29.28	19.72	39.85
70.4	973 b	29.28	19.73	39.76
82.7	1.04 KB	29.28	19.73	39.76
93.8	1.11 KB	29.28	19.73	39.76
113	1.24 KB	29.28	19.73	39.77
134	1.36 KB	29.29	19.73	39.77
153	1.52 KB	29.29	19.73	39.75
165	1.60 KB	29.29	19.71	39.76
186	1.76 KB	29.28	19.73	39.76
207	1.91 KB	29.28	19.75	39.95
240	2.16 KB	29.28	19.75	39.76
295	2.45 KB	29.28	19.75	39.75

Table 4.3 shows the PSNR values comparison between the applications developed from the proposed approach (multi-level) and existing steganography applications (P2P and QuickStego). There are three test images for each application, namely hydrangea (H), jellyfish (J), and lena (L) image. The results show that the application developed using the proposed method has higher PSNR than both P2P and Quickstego. Hence, the proposed method provides a better stego image quality.

In addition, when the size of the plaintext file reaches 113kB as shown in Table 4.3, the Quickstego application can no longer embed the text into the image as it reports large text (LT) errors. This shows that the application has a limited file size. At this file size the quality of the stego image produced by the P2P application drops as indicated by its PSNR values. The difference between the stego image and the original is noticeable to the human eye.

Table 4-3 PSNR comparisons between multi-level application, P2P application, and Quickstego application.

	Multi-Level			P2P			Quick Stego		
Plain	H	J	L	H	J	L	H	J	L
6.20	67.784	74.003	65.492	64.197	66.387	65.005	68.580	68.389	67.852
12.4	66.762	73.772	67.247	62.111	63.698	62.402	65.602	65.368	64.879
20.6	66.572	73.680	66.907	60.938	61.923	61.436	63.404	63.264	62.684
28.70	65.544	70.791	65.948	59.784	60.310	60.384	61.863	61.767	61.009
41.3	65.291	70.508	65.519	58.441	58.839	58.952	60.304	60.169	59.517
57.4	64.817	69.912	64.818	57.191	57.520	57.671	58.913	58.802	58.127
70.4	65.302	69.920	65.664	56.275	56.642	56.652	57.996	56.586	57.237
82.7	65.365	69.520	65.961	56.092	56.340	56.166	57.293	57.194	56.509
93.8	65.584	51.323	65.934	56.090	56.239	54.608	56.734	56.628	LT
113	65.385	51.321	65.664	53.475	53.936	52.442	LT	LT	
134	65.141	70.447	65.350	51.323	51.707	50.927			
153	65.604	69.249	66.472	50.082	50.443	49.894			
165	64.853	69.947	64.994	49.323	49.776	49.332			
186	51.849	69.732	64.953	48.992	49.389	47.052			
207	65.306	68.592	66.237	47.359	47.847	45.306			
240	65.075	68.792	65.821	44.224	44.306	43.464			
295	64.872	68.534	65.544	42.346	42.639	38.708			

Note: H = hydrangea
J = jellyfish
L = lena

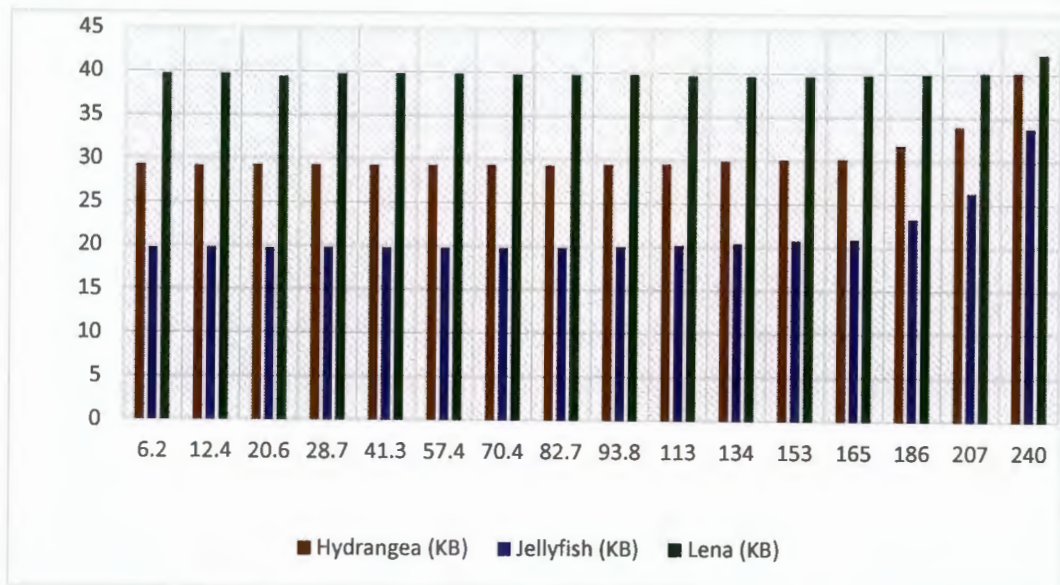


Figure 4-9 shows the size of the text file and size the of stego images

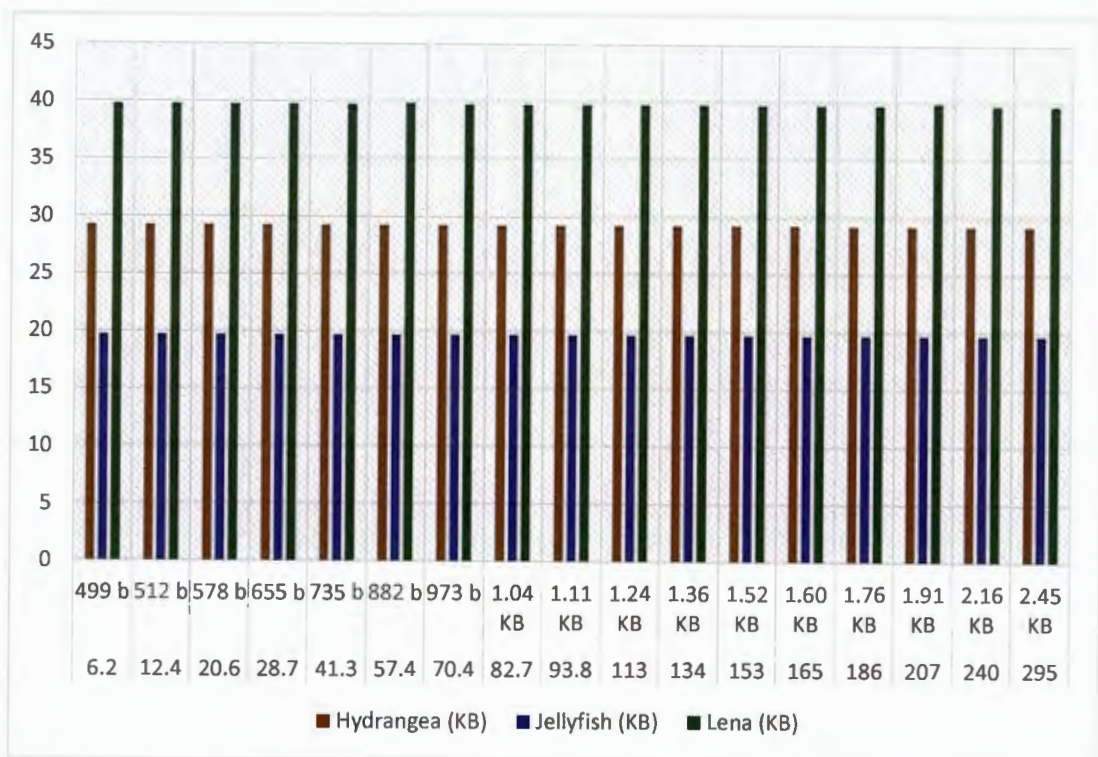


Figure 4-10 shows the size of the text file, compressed text file and size the of stego images

4.4 Chapter Summary

This chapter discussed the implementation of the steganographic model described in Chapter Three, detailed the findings of the models using a simulation program to test and support the theory and carried out analysis of the output. The chapter also presented various software tools used for the system implementation.

CHAPTER FIVE

5.0 Summary, Conclusion and Recommendation

5.1 Summary

A major advantage of this information technology age is the possibility of automating virtually all aspects of human intelligence. However, every new trend in information technology comes with its new threats, including data transmission and sharing on the Internet. The threats posed by hackers or intruders on information transmitted over the Internet demand serious attention, since the channel is an unrestricted one, jointly employed by individuals, government and corporate organizations in information dissemination. At times, the accuracy and integrity of crucial information being exchanged on the Internet is not achieved by the beneficiary due to interruption by hackers because the Internet as a whole does not use secure links, thus information in transit is vulnerable to interception. Therefore, the need to secure messages to be transmitted on the Internet is very important to reduce, if not totally eliminate the chance of the information being detected, removed or modified during transmission, with a view to changing or deleting its contents.

Cryptography and Steganography are both technologies used to protect information over the Internet. Cryptography has been widely used to encrypt plaintext data, transfer the ciphertext over the Internet and decrypt the ciphertext to extract the plaintext at the receiver side. The ciphertext is not really secured however, because hackers or intruders can easily observe that the information being sent on the channel has been encrypted and is not the plaintext. However, a steganography technique proves more prudent and secure when compared with cryptography techniques. This improvement is achieved by embedding the confidential information as part of a cover media such as an image, audio or video to hide the existence of the communication; this way, only the intended recipient realizes its true content.

As communication increases day by day, the value of security over networks to circumvent hackers also increases. Also, as the modern world gradually becomes "paperless", with a huge amount of critical information stored and exchanged over the Internet, cryptography and steganography techniques combined together can provide a sophisticated secure, multi-level steganographic system to accommodate more hidden data. This makes it difficult to detect, remove or modify data transmitted over the Internet.

The system is categorized into two major phases: the sender phase and the receiver phase. In the sender phase, the system has the following stages: cryptography (encryption), compression, encoding, and embedding, the end product here is a stego-image. In the receiver phase, the system has the following activities: retrieve, decode, decompress, and decrypt and the end product here is the original (plaintext) message.

In the compression phase, the message that the sender intends to send is stored inside a file and then compressed using an appropriate compression algorithm to generate compressed information. This is done to reduce the size of the file being sent and also to increase the speed (rate) at which the message is delivered to the destination.

5.2 Conclusion

No combination of proofs has ever been sufficiently potent to permanently stop burglars from successful burglary attempts. The strength of a security installation is measured by the amount of time, effort, money and risks needed by hackers to circumvent or break into a secured system.

In this work, a Secured Multi-Level Steganographic System with improved data hiding capacity is presented. This system combines the technologies in both cryptography and steganography into one, and uses it as a security measure to safeguard data transmission and sharing over the Internet. A key which serves as a crypto-key that is used for message encryption and decryption and also as a stego-key has been applied to the system during embedment of the message into the cover-image.

The system is better when the secret message is compressed, as it allows more data to be embedded. Also, the quality of the reconstructed (stego) image is better compared to that of P2P and QuickStego applications. As emphasis is placed on the area of security and privacy protection, it is believed that this research will help by improving on the security and the hiding capacity of existing technologies.

5.3 Recommendation

There are many important companies who rely on the Internet for over 90 percent of their income, they need to ward off all intrusions and still make a substantial profit. What is required is the relevant expertise, the determination and the will to make it happen.

As we know, the internet is not secure, and the information in transit is vulnerable to attack or interception. The method developed can here be used to secure secret messages intended for transmission. Therefore, this system is highly recommended for organizations that consider information security as of utmost importance.

REFERENCES

- [1] <http://www.internetworldstats.com/stats.htm>.
- [2] D. Stinson, *Cryptography: Theory and Practice*, 2nd Edition, Chapman and Hall/ CRC, February 2002.
- [3] N. Christian, E. Bill, G. Jay, S. Morgan, and W. Karli, "Wrox Professional C#," ISBN 978-0-470-19137-8., 2008.
- [4] A. A. Shejul and U. L. Kulkarni, "A secure skin tone based steganography using wavelet transform," *International Journal of computer theory and Engineering*, vol. 3, pp. 16-22, 2011.
- [5] N. Meghanathan and L. Nayak, "Steganalysis algorithms for detecting the hidden information in image, audio and video cover media," *International Journal of Network Security & Its Application (IJNSA)*, vol. 2, pp. 43-55, 2010.
- [6] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM Systems Journal*, vol. 35, pp. 313-336, 1996.
- [7] K.C.A. Cheddad, J. Condell, P. Mc Kevi, "Digital image steganography: Survey and analysis of current methods," *Signal Processing*, vol. 90, pp. 727-752, 2010.
- [8] S. Changder, N. C. Debnath, and D. Ghosh, "A New approach to hindi text steganography by shifting matra," in *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom'09. International Conference on*, 2009, pp. 199-202.
- [9] S. Singh and G. Agarwal, "Use of image to secure text message with the help of LSB replacement," *International Journal of Applied Engineering Research*, vol. 1, 2010.
- [10] N. Raftari and A. M. E. Moghadam, "Digital Image Steganography Based on Integer Wavelet Transform and Assignment Algorithm," in *Modelling Symposium (AMS), 2012 Sixth Asia*, 2012, pp. 87-92.
- [11] S. Sakkara, D. Akkamahadevi, K. Somashekar, and K. Raghunath, "Integer wavelet based secret data hiding by selecting variable bit length," *International Journal of Computer Applications*, vol. 48, pp. 7-11, 2012.

- [12] D. Bret, "A detailed look at Steganographic Techniques and their use in an Open-Systems Environment," SANS Institution Reading Room Site, 2002.
- [13] C. Douglas, "Internet: Microsoft Encarta 2009," *Microsoft Cooperation*, 2008.
- [14] M. M. Amin, P.S. Ibrahim, P.M. Salleh, M.R. Katmin, "Information Hiding Using Steganography," *Department of Computer System & Communication, Faculty of Computer Science and Information system, University Teknologi Malaysia.*, 2003.
- [15] M. Natarajan and N. Lopamudra, "Steganalysis Algorithms for Detecting the Hidden Information in Image, Audio and Video Cover Media," *International Journal of Network Security & Its Application (IJNSA)*, vol. 2, 2010.
- [16] S. Brigitte, "Introduction to steganography," *Athabasca University COMP607 Project*, 2004.
- [17] B. Debrup, "Asymmetric Key Steganography," *International Conference on Information and Electronics Engineering IPCSIT IACSIT Press, Singapore*, vol. 6, 2011.
- [18] O. Donal, P. Micheal, and T. Hitesh, "Electronic Payment Systems for E-Commerce," *Artech House Inc*, 2001.
- [19] A. Debra and S. Daniel, "Data Compression," www.ics.uci.edu/%7Edan/pubs/DC-Sec1.html, 2011.
- [20] L. Jeffrey, "Data Compression Algorithms:," www.ccs.neu.edu/home/jnl22/oldsite/cshonor/jeff.html#Assumptions, 1996.
- [21] J. Ingemar, L. Mathew, A. Jeffrey, F. Jessica, and K. Ton, "Digital Watermarking and Steganography," ed: Morgan Kaufmann Publishers, 2008.
- [22] D. Artz, "Digital Steganography - Hiding Data within Data," http://www.cc.gatech.edu/classes/AY2003/cs6262_fall/digital_steganography.pdf, 2001.
- [23] H. Farid, "Detecting Steganographic Messages in Digital Images," <http://www.cs.dartmouth.edu/~farid/publications/tr01.pdf>, 2006.
- [24] E. Cole, "Hiding in Plain Sight," in *ISBN 416-236-4433* ed. UK: Wiley John & Sons, Incorporated, 2003.

- [25] Anon, "Audio Steganography: Overview,"
<http://www.snotmonkey.com/work/school/405/overview.html>, 2006.
- [26] D. Bret, "A detailed look at Steganographic Techniques and their use in an Open System Environment," *paper from the SANS Institute Reading Room site. SANS Institute*, 2002.
- [27] A. Calpe, "Steganography in Images,"
http://www.cs.ucf.edu/courses/cot4810/fall04/presentations/Steganography_in_Images.ppt, 2006.
- [28] F. Queirolo, "Steganography in Images,"
<http://www.cse.buffalo.edu/~peter/cse741/Presentations/Refs/Queirolo.pdf>, 2006.
- [29] T. Morkel, J. H. Eloff, and M. S. Olivier, "An overview of image steganography," ISSA, University of Pretoria, Pretoria, 2005.
- [30] A. Mangarae, "Steganography faq," *Zone-H. Org March 18th*, 2006.
- [31] D. Sellars, "An Introduction to Steganography,"
<http://www.totse.com/en/privacy/encryption/163947.html>, 2006.
- [32] K. Minakshi, "Digital Image Processing " *Photogrammetry and Remote Sensing Division, Indian Institute of Remote Sensing, Dehra Dun*, 2011.
- [33] S. Saurabh and A. Gaurav, "Use of image to secure text message with the help of LSB replacement," *Invertis Institute of Engineering and Technology Bareilly India*, 2010.
- [34] A. Westfeld A. and A. Pitzmann, "Attacks on Steganography Systems," *Department of Computer Science Dresden University of Technology*, 1999.
- [35] P. William, "Digital Image processing," 3rd ed: John Wiley & Sons Inc., 2001.
- [36] B. Debnath, D. Poulami, K. Samir, and K. Tai-hoon, "Text Steganography – A Novel Approach," *International Journal of Advanced Science and Technology*, vol. 3, 2009.
- [37] S. Khosravi, D. Abbasi, M. Yektaie, and M. Khosravi, "A New Method to Steganography Whit Processing Picture in Three Colors (RGB)," *Int. J. Comp. Tech. Appl*, vol. 2, 2010. 274

- [38] K. Rupinder, K. Mandeep, and M. and Rahul, "A New Efficient Approach towards Steganography," *International Journal of Computer Science and Information Technologies (IJCSIT)*, , vol. 2 2011.
- [39] Shiva-Kumar K., Raja K., and Sabyasachi P., "Hybrid Domain in LSB Steganography," *International Journal of Computer Applications*, vol. 19, 2011.
- [40] R. Tiwari and G. Sahoo, "Designing Some Imperceptible Data Hiding Methodologies Using Steganographic Techniques," *International Journal of Information Technology and Knowledge Management*, vol. 1, pp. 209-217, 2008.
- [41] T. Balfrishan and P. Amar, "Moderate Bit Insertion for Hiding Crypto-Data in Digital Image for Steganography: A special issue from IJCA " *www.ijcsonline.org*, 2011.
- [42] M. Atanu, "Steganography - Technique to hide information within image file," http://www.programmer2programmer.net/live_projects/project_7/steganography.aspx, 2011.
- [43] Cybernescence, "Hide text in images fast and easy," <http://www.quickcrypto.com/free-steganography-software.html>, 2010.
- [44] M. Joel, "Murach's C# 2010," *Mike Murach & Associates, Inc.*, 2010.
- [45] S. David, "Data Compression - The Complete Reference," *Springer Science and Business Media LLC*, 2006
- [46] E. Arnoud, "The DES encryption algorithm," www.iusmentis.com/technology/encryption/des/, 2005.
- [47] J. Orlin, "The DES Algorithm Illustrated," <http://www.orlingrabbe.com/homepage.html>, 2009.
- [48] P. Shor, "Compression algorithm using lempel ziv," http://www-math.mit.edu/~shor/PAM/lempel_ziv_note.pdf, 2010.
- [49] K. Amanpreet, D. Renu, and S. Geeta, "A new Image Steganography Based on First Component Alteration Technique," *International of Computer Science and Information Security*, vol. 6, 2009.

APPENDIX

CODE FOR ENCRYPTING MESSAGE

```
public class Encryption
{
    public Encryption()
    {
        // Do Nothing
    }

    public static string Start(string plainText, string key)
    {
        string finalText = ProcessMESSAGE(plainText, key);
        string hexFinalText = Conversion.FromBinaryToHexadecimal(finalText);
        string cipherText = Conversion.FromHexadecimalToText(hexFinalText);

        return cipherText;
    }

    private static string ProcessMESSAGE(string plainText, string key)
    {
        Keys keys = Processor.ProcessKEY(key);

        string hexText = Conversion.FromStringToHexadecimal(plainText);
        string binText = Conversion.FromHexadecimalToBinary(hexText);
        binText = Conversion.SetMultipleOf64Bits(binText);
        StringBuilder EncryptedTextBuilder = new StringBuilder(binText.Length);

        for (int i = 0; i < (binText.Length / 64); i++)
        {
            string PermutedText = Iteration.DoPermutation(binText.Substring(i * 64, 64),
                PermutationData.IP);

            string L0 = "", R0 = "";

            L0 = Iteration.GetLeftHalvesKey(PermutedText);
            R0 = Iteration.GetRightHalvesKey(PermutedText);

            string FinalText = Processor.EncryptMessageUsingKey(L0, R0, keys, false);
            EncryptedTextBuilder.Append(FinalText);
        }
        return EncryptedTextBuilder.ToString();
    }
}
```

CODE FOR ENCRYPTING MESSAGE

```
public class Decryption
{
    public static string Start(string cipherText, string key)
    {
        string finalText = ProcessMESSAGE(cipherText, key);
        string plainFinalText = Conversion.FromBinaryToHexadecimal(finalText);
        string plainText = Conversion.FromHexadecimalToText(plainFinalText);
        return plainText;
    }

    private static string ProcessMESSAGE(string cipherText, string key)
    {
        Keys keys = Processor.ProcessKEY(key);
        string hexText = Conversion.FromStringToHexadecimal(cipherText);
```

```

string binText = Conversion.FromHexadecimalToBinary(hexText);
    binText = Conversion.SetMultipleOf64Bits(binText);

StringBuilder DecryptedTextBuilder = newStringBuilder(binText.Length);

for (int i = 0; i < (binText.Length / 64); i++)
    {
        string PermutedText = Iteration.DoPermutation(binText.Substring(i * 64, 64),
        PermutationData.IP);

        string L0 = "", R0 = "";

        L0 = Iteration.GetLeftHalvesKey(PermutedText);
        R0 = Iteration.GetRightHalvesKey(PermutedText);

        string FinalText = Processor.EncryptMessageUsingKey(L0, R0, keys, true);

        if ((i * 64 + 64) == binText.Length)
            {
                StringBuilder last_text = newStringBuilder(FinalText.TrimEnd('0'));

                int count = FinalText.Length - last_text.Length;

                if ((count % 8) != 0)
                    {
                        count = 8 - (count % 8);
                    }

                string append_text = "";

                for (int k = 0; k < count; k++)
                    {
                        append_text += "0";
                    }

                DecryptedTextBuilder.Append(last_text.ToString() + append_text);
            }
        else
            {
                DecryptedTextBuilder.Append(FinalText);
            }
    }

return DecryptedTextBuilder.ToString();
    }
}

```

CODE FOR CREATING 16 SUBKEYS IN DES (EACH IS 48BITS LONG)

```

publicclassKeys
    {
        publicstring[] Cn = newstring[17]; // Left Halves
        publicstring[] Dn = newstring[17]; // Right Halves
        publicstring[] Kn = newstring[16];

        internalKeys SetHalveKeys(string C0, string D0)
            {
                Keys keys = newKeys();
                keys.Cn[0] = C0;
                keys.Dn[0] = D0;

                for (int i = 1; i < keys.Cn.Length; i++)
                    {
                        keys.Cn[i] = this.LeftShift(keys.Cn[i - 1],
                        PermutationData.numberOfShifts[i]);
                    }
            }
    }

```

```

        keys.Dn[i] = this.LeftShift(keys.Dn[i] - 1,
PermutationData.numberOfShifts[i]);
        keys.Kn[i - 1] = Iteration.DoPermutation(keys.Cn[i] + keys.Dn[i],
PermutationData.PC_2);
    }

```

```

return keys;
}

```

```

public string LeftShift(string text)
{
    return this.LeftShift(text, 1);
}

```

```

public string LeftShift(string text, int count)
{
    if (count < 1)
    {
        Console.WriteLine("The count of leftshift must be more than 1 time.");
        return null;
    }

```

```

    string temp = text.Substring(0, count);
    StringBuilder shifted = new StringBuilder(text.Length);
    shifted.Append(text.Substring(count) + temp);

```

```

    return shifted.ToString();
}
}

```

CODE FOR PERFORMING DES ALGORITHM ITERATION

```

public static class Iteration
{
    public static string DoPermutation(string text, int[] order)
    {
        StringBuilder PermutedText = new StringBuilder(order.Length);
        for (int i = 0; i < order.Length; i++)
        {
            PermutedText.Append(text[order[i] - 1]);
        }
        return PermutedText.ToString();
    }
}

```

//For SBoxes Transformation

```

public static string DoPermutation(string text, int[,] order)
{
    string PermutedText = "";

    int rowIndex = Convert.ToInt32(text[0].ToString() + text[text.Length - 1].ToString(), 2);
    int colIndex = Convert.ToInt32(text.Substring(1, 4).ToString(), 2);
    PermutedText = Conversion.FromDecimalToBinary(order[rowIndex, colIndex]);
    return PermutedText;
}

```

```

public static string GetRightHalvesKey(string text)
{
    return GetHalveKey(false, text);
}

```

```

public static string GetLeftHalvesKey(string text)
{
    return GetHalveKey(true, text);
}

```

```

    }

    private static string GetHalveKey(bool isLeft, string text)
    {
        if ((text.Length % 8) != 0)
        {
            Console.WriteLine("The key is not multiple of 8bit.");
            return null;
        }

        int midindex = (text.Length / 2) - 1;
        string result = "";

        if (isLeft)
        {
            result = text.Substring(0, midindex + 1);
        }
        else
        {
            result = text.Substring(midindex + 1);
        }

        return result;
    }

```

CODE FOR PERFORMING CRYPTOGRAPHY OPERATION

```

public class Cryptography
{
    static string cipher_text_bin = "";

    public static string Encrypt(string msg, string key)
    {
        string cipherText = DoEncryption(msg, key);

        return cipherText;
    }

    public static string Decrypt(string msg, string key)
    {
        string plainText = DoDecryption(msg, key);

        return plainText;
    }

    private static string DoDecryption(string cipherText, string key)
    {
        string plainText = Algorithm.DecryptMessage(cipherText, key);
        return plainText;
    }

    private static string DoEncryption(string msg, string key)
    {
        string cipherText = Algorithm.EncryptMessage(msg, key);
        return cipherText;
    }

    internal static void EncryptFile(string filePath, string fOut, byte[] desKey, byte[] desIV)
    {
        using (FileStream fin = new FileStream(filePath, FileMode.Open, FileAccess.Read))
        {

```

```

using (FileStream fout = new FileStream(fout, FileMode.OpenOrCreate, FileAccess.Write,
    FileShare.ReadWrite))
    {
        fout.SetLength(0);

byte[] bin = new byte[filePath.Length];
long rdlen = 0;
long totlen = fin.Length
int len;

DES des = new DESCryptoServiceProvider();
CryptoStream encStream = new CryptoStream(fout, des.CreateEncryptor(desKey, desIV),
    CryptoStreamMode.Write);

Console.WriteLine("Encrypting...");

while (rdlen < totlen)
    {
        len = fin.Read(bin, 0, filePath.Length);
        encStream.Write(bin, 0, len);
        rdlen = rdlen + len;
Console.WriteLine("{0} bytes processed", rdlen);
    }
    encStream.Close();
    fout.Close();
    fin.Close();
}

}

public static void DecryptFile(string sInputFilename, string sOutputFilename, string sKey)
{
    DESCryptoServiceProvider DES = new DESCryptoServiceProvider();
    DES.Key = ASCIIEncoding.ASCII.GetBytes(sKey);
    DES.IV = ASCIIEncoding.ASCII.GetBytes(sKey);
    using (FileStream fsread = new FileStream(sInputFilename, FileMode.Open, FileAccess.Read))
    {
        ICryptoTransform desdecrypt = DES.CreateDecryptor();
        CryptoStream cryptostreamDecr = new CryptoStream(fsread,
            desdecrypt,
            CryptoStreamMode.Read);
        StreamWriter fsDecrypted = new StreamWriter(sOutputFilename);
        fsDecrypted.Write(new StreamReader(cryptostreamDecr).ReadToEnd());
        fsDecrypted.Flush();
        fsDecrypted.Close();
    }
}

CODE FOR COMPRESSING MESSAGE FILE
public static string CompressFile(ref string path)
{
    try
    {
        FileStream sourceFile = File.OpenRead(path);
        path = path + ".gz";
        FileStream destinationFile = File.Create(path);

        byte[] buffer = new byte[sourceFile.Length];
        sourceFile.Read(buffer, 0, buffer.Length);

        using (GZipStream output = new GZipStream(destinationFile, CompressionMode.Compress))

```

```

        {
            output.Write(buffer, 0, buffer.Length);
        }
        sourceFile.Close();
        destinationFile.Close();

return "File successfully compressed at the \r\n same destination as the source File";
    }
    catch (Exception ex)
    {
        return ex.Message;
    }
}

```

```

public static string UncompressFile(string path)
{
    FileStream sourceFile = File.OpenRead(path);

    int gz = path.LastIndexOf('.');
    path = path.Remove(gz);
    FileStream destinationFile = File.Create(path);

    byte[] buffer = new byte[4096];
    int n;

    using (GZipStream input = new GZipStream(sourceFile,
        CompressionMode.Decompress, false))
    {
        n = input.Read(buffer, 0, buffer.Length);
        destinationFile.Write(buffer, 0, n);
    }
    sourceFile.Close();
    destinationFile.Close();

    return path;
}
}

```

CODE FOR PERFORMING STEGANOGRAPHY ENCODING AND DECODING

```

public class StegoHelper
{
    public bool isEncryption { get; set; }
    public long FileSize { get; set; }
    public string SourceImagePath { get; set; }

    public Bitmap SourceBitmap {
        get
        {
            if (isEncryption)
                return new Bitmap(Image.FromFile(ConvertToBitmap()));
            else
                return new Bitmap(Image.FromFile(SourceImagePath));
        }
    }

    public int Height { get; set; }
    public int Width { get; set; }
    public long FileNameSize { get; set; }
    string _stegoImagePath;
    public string StegoImagePath
    {
        get
        {
            return _stegoImagePath;
        }
    }
}

```

```

    }
}

public string FilePath { get; set; }
public byte[] FileByte {
    get
    {
        return File.ReadAllBytes(FilePath);
    }
}

public void EncryptLayer()
{
    long FSize = FileSize;
    Bitmap changedBitmap = EncryptLayer(8, SourceBitmap, 0, (Height * (Width / 3) * 3) / 3 -
    FileNameSize - 1, true);
    FSize -= (Height * (Width / 3) * 3) / 3 - FileNameSize - 1;
    if (FSize > 0)
    {
        for (int i = 7; i >= 0 && FSize > 0; i--)
        {
            changedBitmap = EncryptLayer(i, changedBitmap, (((8 - i) * Height *
            (Width / 3) * 3) / 3 -
            FileNameSize - (8 - i)), (((9 - i) * Height * (Width / 3) * 3) / 3 -
            FileNameSize - (9 - i)), false);
            FSize -= (Height * (Width / 3) * 3) / 3 - 1;
        }
    }

    int lastSlash = FilePath.LastIndexOf('\\');
    string fileDir = FilePath.Remove(lastSlash + 1);
    _stegoImagePath = String.Format("{0}StegoImagePath ({1}).bmp", fileDir,
    DateTime.Now.ToString().Replace('/', '-').Replace(':', '-'));

    changedBitmap.Save(StegoImagePath);
}

private Bitmap EncryptLayer(int layer, Bitmap inputBitmap, long startPosition, long
endPosition, bool writeFileName)
{
    Bitmap outputBitmap = inputBitmap;
    layer--;
    int i = 0, j = 0;
    long FNSize = 0;
    bool[] t = new bool[8];
    bool[] rb = new bool[8];
    bool[] gb = new bool[8];
    bool[] bb = new bool[8];
    Color pixel = new Color();
    byte r, g, b;

    if (writeFileName)
    {
        FNSize = FileNameSize;
        string fileName = JustFileName(FilePath);

        for (i = 0; i < Height && i * (Height / 3) < FileNameSize; i++)
        for (j = 0; j < (Width / 3) * 3 && i * (Height / 3) + (j / 3) < FileNameSize; j++)
        {
            byte2bool((byte)fileName[i * (Height / 3) + j / 3], ref t);
            pixel = inputBitmap.GetPixel(j, i);
            r = pixel.R;

```

```

        g = pixel.G;
        b = pixel.B;
        byte2bool(r, ref rb);
        byte2bool(g, ref gb);
        byte2bool(b, ref bb);

if (j % 3 == 0)
    {
        rb[7] = t[0];
        gb[7] = t[1];
        bb[7] = t[2];
    }

elseif (j % 3 == 1)
    {
        rb[7] = t[3];
        gb[7] = t[4];
        bb[7] = t[5];
    }

else
    {
        rb[7] = t[6];
        gb[7] = t[7];
    }

Color result = Color.FromArgb((int)bool2byte(rb), (int)bool2byte(gb), (int)bool2byte(bb));
    outputBitmap.SetPixel(j, i, result);
    }
    i--;
}

int tempj = j;
for (; i < Height && i * (Height / 3) < endPosition - startPosition + FNSize && startPosition
+ i * (Height / 3) < FileSize + FNSize; i++)
for (j = 0; j < (Width / 3) * 3 && i * (Height / 3) + (j / 3) < endPosition -
    startPosition + FNSize && startPosition + i * (Height / 3) + (j / 3) <
    FileSize + FNSize; j++)
    {
        if (tempj != 0)
            {
                j = tempj;
                tempj = 0;
            }
        byte2bool((byte)FileByte[startPosition + i * (Height / 3) + j / 3 -
FNSize], ref t);
        pixel = inputBitmap.GetPixel(j, i);
        r = pixel.R;
        g = pixel.G;
        b = pixel.B;
        byte2bool(r, ref rb);
        byte2bool(g, ref gb);
        byte2bool(b, ref bb);

if (j % 3 == 0)
    {
        rb[layer] = t[0];
        gb[layer] = t[1];
        bb[layer] = t[2];
    }

elseif (j % 3 == 1)
    {
        rb[layer] = t[3];
        gb[layer] = t[4];
        bb[layer] = t[5];
    }

else

```

```

        {
            rb[layer] = t[6];
            gb[layer] = t[7];
            Color result = Color.FromArgb((int)bool2byte(rb), (int)bool2byte(gb), (int)bool2byte(bb));
            outputBitmap.SetPixel(j, i, result);
        }

    long tempFS = FileSize, tempFNS = FileNameSize;
    r = (byte)(tempFS % 100);
    tempFS /= 100;
    g = (byte)(tempFS % 100);
    tempFS /= 100;
    b = (byte)(tempFS % 100);
    Color flenColor = Color.FromArgb(r, g, b);
    outputBitmap.SetPixel(Width - 1, Height - 1, flenColor);

    r = (byte)(tempFNS % 100);
    tempFNS /= 100;
    g = (byte)(tempFNS % 100);
    tempFNS /= 100;
    b = (byte)(tempFNS % 100);
    Color fnlenColor = Color.FromArgb(r, g, b);
    outputBitmap.SetPixel(Width - 2, Height - 1, fnlenColor);

    return outputBitmap;
}

string PadBinary(string bin)
{
    int l = bin.Length;

    for (int i = 0; i < (8 - l); i++)
        bin = string.Format("0{0}", bin);

    return bin;
}

public string JustFileName(string path)
{
    int lastSlash = path.LastIndexOf('\\');
    string justFName = path.Substring(lastSlash + 1);

    return justFName;
}

private void byte2bool(byte inp, ref bool[] outp)
{
    if (inp >= 0 && inp <= 255)
        for (short i = 7; i >= 0; i--)
        {
            if (inp % 2 == 1)
                outp[i] = true;
            else
                outp[i] = false;
            inp /= 2;
        }
    else
        throw new Exception("Input number is illegal.");
}

private byte bool2byte(bool[] inp)

```

```

        {
            byte outp = 0;
            for (short i = 7; i >= 0; i--)
            {
                if (inp[i])
                    outp += (byte)Math.Pow(2.0, (double)(7 - i));
            }
            return outp;
        }

        string bitmapImagePath;
        private string ConvertToBitmap()
        {
            string copyImagePath = SourceImagePath;
            Image img = new Bitmap(copyImagePath);
            try
            {
                Bitmap bmp = new Bitmap(img);

                int iExt = copyImagePath.LastIndexOf('.');
                string SourceImagePathWithoutExt = copyImagePath.Remove(iExt);
                bitmapImagePath = SourceImagePathWithoutExt + ".bmp";

                img.Dispose();

                bmp.Save(bitmapImagePath, ImageFormat.MemoryBmp);
                bmp.Dispose();

            }
            return bitmapImagePath;
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }

    public string ConvertToJPEG(string bitmapPath)
    {
        MemoryStream ms = new MemoryStream(File.ReadAllBytes(bitmapPath));
        Image img = Image.FromStream(ms);
        try
        {
            Bitmap bmp = new Bitmap(img);

            int iExt = bitmapPath.LastIndexOf('.');
            string SourceImagePathWithoutExt = bitmapPath.Remove(iExt);
            string JPEGImagePath = SourceImagePathWithoutExt + ".jpg";
            img.Dispose();
            bmp.Save(JPEGImagePath, ImageFormat.Jpeg);
            bmp.Dispose();

        }
        return JPEGImagePath;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}

public string DecryptLayer()

```

```

    {
int i, j = 0;
bool[] t = newbool[8];
bool[] rb = newbool[8];
bool[] gb = newbool[8];
bool[] bb = newbool[8];
Color pixel = newColor();
byte r, g, b;
        pixel = SourceBitmap.GetPixel(Width - 1, Height - 1);
long fSize = pixel.R + pixel.G * 100 + pixel.B * 10000;
        pixel = SourceBitmap.GetPixel(Width - 2, Height - 1);
long fNameSize = pixel.R + pixel.G * 100 + pixel.B * 10000;
byte[] res = newbyte[fSize];
string resFName = "";
byte temp;

//Read file name:
for (i = 0; i < Height && i * (Height / 3) < fNameSize; i++)
for (j = 0; j < (Width / 3) * 3 && i * (Height / 3) + (j / 3) < fNameSize; j++)
    {
        pixel = SourceBitmap.GetPixel(j, i);
        r = pixel.R;
        g = pixel.G;
        b = pixel.B;
        byte2bool(r, ref rb);
        byte2bool(g, ref gb);
        byte2bool(b, ref bb);

if (j % 3 == 0)
    {
        t[0] = rb[7];
        t[1] = gb[7];
        t[2] = bb[7];
    }
elseif (j % 3 == 1)
    {
        t[3] = rb[7];
        t[4] = gb[7];
        t[5] = bb[7];
    }
else
    {
        t[6] = rb[7];
        t[7] = gb[7];
        temp = bool2byte(t);
        resFName += (char)temp;
    }
    }

//Read file on layer 8 (after file name):
int tempj = j;
    i--;

for (; i < Height && i * (Height / 3) < fSize + fNameSize; i++)
for (j = 0; j < (Width / 3) * 3 && i * (Height / 3) + (j / 3) < (Height * (Width / 3) * 3) /
3 - 1 && i *
    (Height / 3) + (j / 3) < fSize + fNameSize; j++)
    {
if (tempj != 0)
    {
        j = tempj;
        tempj = 0;
    }
    }

```

```

        pixel = SourceBitmap.GetPixel(j, i);
        r = pixel.R;
        g = pixel.G;
        b = pixel.B;
        byte2bool(r, ref rb);
        byte2bool(g, ref gb);
        byte2bool(b, ref bb);

if (j % 3 == 0)
    {
        t[0] = rb[7];
        t[1] = gb[7];
        t[2] = bb[7];
    }
elseif (j % 3 == 1)
    {
        t[3] = rb[7];
        t[4] = gb[7];
        t[5] = bb[7];
    }
else
    {
        t[6] = rb[7];
        t[7] = gb[7];
        temp = bool2byte(t);
        res[i * (Height / 3) + j / 3 - fNameSize] = temp;
    }
}

```

//Read file on other layers:

```
long readedOnL8 = (Height * (Width / 3) * 3) / 3 - fNameSize - 1;
```

```

for (int layer = 6; layer >= 0 && readedOnL8 + (6 - layer) * ((Height * (Width / 3) * 3) / 3 - 1) < fNameSize; layer--)
for (i = 0; i < Height && i * (Height / 3) + readedOnL8 + (6 - layer) * ((Height * (Width / 3) * 3) / 3 - 1) < fNameSize; i++)
for (j = 0; j < (Width / 3) * 3 && i * (Height / 3) + (j / 3) + readedOnL8 + (6 - layer) * ((Height * (Width / 3) * 3) / 3 - 1) < fNameSize; j++)
    {
        pixel = SourceBitmap.GetPixel(j, i);
        r = pixel.R;
        g = pixel.G;
        b = pixel.B;
        byte2bool(r, ref rb);
        byte2bool(g, ref gb);
        byte2bool(b, ref bb);

if (j % 3 == 0)
    {
        t[0] = rb[layer];
        t[1] = gb[layer];
        t[2] = bb[layer];
    }
elseif (j % 3 == 1)
    {
        t[3] = rb[layer];
        t[4] = gb[layer];
        t[5] = bb[layer];
    }
else
    {
        t[6] = rb[layer];
        t[7] = gb[layer];
        temp = bool2byte(t);

```

```

        res[i * (Height / 3) + j / 3 + (6 - layer) * ((Height * (Width /
3) * 3) / 3 - 1) + readedOnL8] = temp;
    }
}

string retrievedFilePath = FilePath + "\\\" + resFName;
File.WriteAllBytes(retrievedFilePath, res);

return retrievedFilePath;
}
}

```