Women University Press
Jayoti Publication Desk
ISBN No. : 978-81-950200-1-0

Stegnography In cryptography

- JV'n Nishu Sharma
- JV'n Indu

Original Image

Steganography Image

JAYOTI VIDYAPEETH WOMEN'S UNIVERSITY, JAIPUR

UGC Approved Under 2(f) & 12(b) | NAAC Accredited | Recognized by Statutory Councils

Printed by :
JAYOTI PUBLICATION DESK

Published by :
Women University Press
Jayoti Vidyapeeth Women's University, Jaipur

Faculty of Education & Methodology

Stegnography in Cryptography

Author Name: JV'n Ms. Nishu Sharma, JV'n Ms. Indu

Published By: Women University Press

Publisher's Address: Jayoti Vidyapeeth Women's University, Jaipur
Vedaant Gyan Valley,
Village-Jharna, Mahala Jobner Link Road, NH-8
Jaipur Ajmer Express Way,
Jaipur-303122, Rajasthan (INDIA)

Printer's Detail: Jayoti Publication Desk

Edition Detail: I

ISBN: 978-81-950200-1-0

Index

Chapter - 1

INTRODUCTION

1.1 Introduction

In the current scenario of the world, the technologies have advanced so much that most of the people prefer working with the internet as the main medium to transfer data or messages from one location to another in the world. There are various possible ways to transmit data through internet: via e-mails, chats, etc. The data transition is made very fast, simple, efficient and accurate via the internet.

However, the main problems while sending the data over the internet is security threat it faces i.e. the personal or confidential data can be stolen or hacked in many ways. Therefore it becomes important to consider the data security, as it is one of the most important factors that need attention in the process of data transferring.

Data security means protecting the data from unauthorized users or hackers and providing high security to stop or prevent data modification. This area of data security has gained much more attention over the last few years of period of time due to the heavy increase in data transfer rate through the internet.

In order to improve the features of data security during data transfers via the internet, many techniques have been developed like: Cryptography, digital watermarking and Steganography. While Cryptography is a simply technique to conceal information by encrypting the data to cipher texts and transmitting it to the receiver using an specific and unknown key,

Steganography provides further more security by hiding the special or cipher text into an invisible image or any other formats.

Cryptography and steganography are general, well known and widely used techniques that change the information (messages) in order to hide or cipher its existence. These techniques are

being in use in a lots of applications in computer science and other related fields: they are used to protect various type of messages like e-mail, credit card information, corporate data, etc.

More specifically, steganography is the art and science of data communication in a way which hides the presence of the communication. A steganographic system thus embeds hidden content inside a media so as not to present an eavesdropper's suspicion. As an example, it is possible to hide a text behind an image or an audio file.

On the other hand, cryptography is the study of techniques using mathematics related to issues of information security like confidentiality, data security as integrity, authentication, and data origin authentication.

Cryptography protects data by converting it into an unreadable format. It is important to achieve confidential and secure transmission over a network which is public. The actual text, or plaintext, is manipulated into a coded text equivalent called ciphertext via an encryption algorithm.

Only those receipiantes who have a secret key can decipher/convert (decrypt) the ciphertext into plaintext or actual text. Cryptography systems can be broadly categorized into symmetric-key systems which use a single key (i.e., a password) that both the sender and the receiver have, and public-key systems that use two separate keys, a public key known to everyone and a private key that only a particular recipient of messages uses[12].

1.2 Proposed Study

- To hide the message or a secret data into an image which acts as a cover medium.
- The primary motivation of my current work is to increase efficiency and accuracy of the stego image.
- To develop a software system(Java Based) for stegnography of images. The system will be able to hide the message within image and will be able to retrieve the message back from the image.

- The Software system used LSB method for stegnography and encryption using Java APIs.

We investigated the methods of stegnography with the special emphasis of method of LSB. Also we developed a Java based system to hide the message inside an image and to retrieve the information back from the information.

1.3 Outline of The chapter

In chapter 1, the general overview is given with proposed study and outline of thesis.

Chapter 2 introduces detailed literature review.

Chapter 3 discussees about the cryptography and stegnography with its methods. Chapter 4 describes efficiency considerations and its difference with proposed method. The chapter also describes details about technology used in proposed tool.

In Chapter 5, screenshots with details of implementation is given. In chapter 6 conclusion and future work is given. At last references are shown.

Chapter- 2

LITERATURE REVIEW

The word cryptography is made by two Greek words which mean “secret writing”. Cryptography is the technique of scrambling the actual text by rearranging, altering or substituting the original text, arranging it in a different unreadable format which is not visible.

Cryptography is an effective way to protect the information that is transmitting through the network communication paths (**Bishop, 2005).**

Steganography in Greek means "covered writing". Steganography is the process of hiding the one information into other sources of information like text, image or audio file, so that it is not visible to the natural view.

There are varieties of steganographic techniques available to hide the data depending upon the carriers we use. Steganography and cryptography both are used for the purpose of sending the data securely. The same approach is followed in Steganography as in cryptography like encryption, decryption and secret key. In steganography the message is kept secret without any changes but in cryptography the original content of the message is differed in different stages like encryption and decryption. Steganography supports different types of digital formats that are used for hiding the data. These files are known as carriers. Depending upon the redundancy of the object the suitable formats are used. Redundancy is the process of providing better accuracy for the object that is used for display by the bits of object. The main file formats that are used for steganography are Text, images, audio, video, protocol **(Morkel, 2005).**

Cryptology is the science that deals about cryptography and cryptanalysis. Cryptography is the approach of sending the messages secretly and securely to the destination. Cryptanalysis is the method of obtaining the embedded messages into original texts**(Whitman, 2007).**

C.P.Sumathiet. al (2013) describes, While steganography can be achieved using any cover media, we are concerned with hiding data in digital images. The features expected of a stego-medium are imperceptibility and robustness, so that the secret message is known only to the intended receiver and also the stego-medium being able to withstand attacks from intruders. The

amount of secret message embedded should be such that it doesn't reduce the quality of the stego image. The goal of steganography is to embed secret data into a cover in such a way that no one apart from the sender and intended recipients even realizes there is secret data. A few key properties that must be considered when creating a digital data hiding system are

- Imperceptibility: Imperceptibility is the property in which a person should be unable to distinguish the original and the stego-image.
- Embedding Capacity: Refers to the amount of secret information that can be embedded without degradation of the quality of the image.
- Robustness: Refers to the degree of difficulty required to destroy embedded information without destroying the cover image.

MamtaJuneja, and Dr. Parvinder S. Sandhu(2013), proposed an improved LSB(least Significant bit) based Steganography technique for images imparting better information security . They presents an embedding algorithm for hiding encrypted messages in nonadjacent and random pixel locations in edges and smooth areas of images. It first encrypts the secret message, and detects edges in the cover-image using improved edge detection filter. Message bits are then, embedded in the least significant byte of randomly selected edge area pixels and 1-3-4 LSBs of red, green, blue components respectively across randomly selected pixels across smooth area of image.

M.Rajkamal And B.S.E.Zoraida(2014), developed a new technique of image steganography inside the embedding the encrypted Data file or message using Hash-LSB with RSA algorithm for providing more security to data as well as our data hiding method. The developed technique uses a hash function to generate a pattern for hiding data bits into LSB of RGB pixel values of the carry image. This technique makes sure that the data has been encrypted before embedding it intoa carry image. Embedded-text in images usually carries important messages about the content.

LSB based technique changes pixel value by ± 1 or leave them unchanged. The goal of a steganalyst is to estimate if I has hidden data. (I - Index set that denote the mean subtracted cover

image) Substitution Based Steganographic Methods listed in chronological order starting from latest. Table 2.1

S.No.	Author	Year	Method Used
1.	Mamta Juneja et. al.[4]	2013	Two component based LSB
2.	P.Thiyagarajan et. al [5]	2013	Scheme using 3D geometric models.
3.	Shamim Ahmed Laskar et. al[6]	2013	Data embedding in the red plane of the image selected using PRNG
4.	S.Shanmuga Priya et. al [7]	2012	Embedding done in the sharper edge regions using a threshold
5.	B.Sharmila et. al. [8]	2012	Edge regions selected for embedding using LSBMR (LSB Matching Revisited)
6.	Shweta Singhal et. al [9]	2011	1 byte of blue factor of pixels are replaced with secret bits.
7.	Fahim Irfan Alam et al [10]	2011	Noise filtering before embedding combined with encryption.
8.	Rajkumar Yadav et al. [11]	2011	A novel approach for image steganography In spatial domain using last two bits of pixel value
9.	M.B.Ould MEDENI et. al. [12]	2010	A novel steganographic method based on Pixel Value Differencing (PVD)
10.	Weiqi Luo et. al. [13]	2010	An Edge adaptive scheme for Region selection and LSBMR for data
11.	C.H.Yang et. al. [14]	2010	Improving histogram based reversible data hiding by

			interleaving predictions (512 x 512 Image)
12.	Venkata Abhiram.M et. al. [15]	2009	Pixel Intensity based steganography with improved randomness.
13.	G.Sahoo et. al. [16]	2009	Data embedded in static & dynamic portions after place analysis
14.	Jasvinder Kaur et. al. [17]	2009	Embedding using digital operations are compared
15.	Hao-Tian et. al. [18]	2009	Steganography in 3D geometrics & images using adjacent Bin Mapping (LSB^+ algorithm)
16.	Bawankar Chetan.D et. al. [19]	2009	Steganography Algorithm using Pattern Matching with External Hardware
17.	Tanmay Bhattacharya et. al. [20]	2009	A hiding technique using bit level cross fold transposition and genetic algorithm
18.	Chin-Chen Chang et. al. [21]	2004	Code word grouping - palette generation algorithm Encoded codeword is modified to hide secret message
19.	R.Chandramouli et. al. [22]	2001	Adaptive Steganography

In Hemalatha.S et.al's [23] paper, the authors propose a method that uses two gray scale images of size 128 x 128 that are used as secret images and embedding is done in RGB and YCbCr domains. The quality of stego images are good in RGB domain by comparing the PSNR values. The authors have used Integer Wavelet Transform (IWT) to hide secret images in the color cover

image. The authors have compared the PSNR values and image quality when embedding is done in the RGB and YCbCr domains.

In another article by Hemalatha .S et. al. [24] Integer Wavelet Transform (IWT) have been suggested to hide multiple secret images and keys in a color cover image which is more efficient. The cover image is represented in the YCbCr color space. Two keys are obtained, encrypted and hidden in the cover image using IWT.

In Keith.L. Haynes 's article [25] the author studies the use of image steganography to breach an organization's physical and cyber defences. The proposed method utilizes computer vision and machine learning techniques to produce messages that are undetectable and if intercepted cannot be decrypted without key compromise. To avoid detection DWT (Discrete Wavelet Transform) is used.

In S.Arivazhagan et. al.'s work [26] the authors propose a method that works in the transform domain and attempts to extract the secret almost as same as the embedded one, maintaining minimal changes to cover image by using techniques like median maintenance, offset & quantization.

A modified approach for embedding colour images within colour images is proposed and it overcomes the limitations in embedding. Arnold Transform is applied on the secret image to increase robustness. This transformed image is then split into the three colour planes R, G, B and are subjected to DWT individually, converted to bit stream and then concatenated to be embedded in the cover image which is also subjected to DWT.

In Anindya Sarkar et. al.'s paper [27] the authors propose a Matrix Embedding with Repeat Accumulate (ME-RA) based steganography in which the host coefficients are minimally perturbed such that the transmitted bits fall in a coset of a linear code, with the syndrome conveying the hidden bits.

In Prosanta Gope et. al.'s article [28], the authors introduce an enhanced JPEG steganography along with a suitable encryption methodology using a symmetric key cryptographic algorithm.

In Po-Chyi et.al.'s article [29] the authors compare the advantage of embedding in JPEG 2000 images with the previous approach of embedding in JPEG images. Most of the steganographic methods are based on JPEG because as a block DCT codec JPEG lends itself a good candidate for information hiding due to its fixed block structure.

In Hideki Noda et.al.'s paper [30] the authors propose a method that is based on a seamless integration of JPEG2000 lossy compression scheme and bit-plane complexity segmentation (BPCS) steganography. In bit-plane decomposition an n bit image is decomposed into a set of n binary images by bit slicing operations, combined with replacing binary data in LSB bit planes with secret data.

In Tomas Filler et. al.'s work [31], the authors propose a practical methodology for minimizing additive distortion in steganography with general embedding operation which is more flexible and easy.

In Jessica Fridrich et.al.'s research paper [32] the authors propose a reversible embedding scheme for VQ-compressed images that is based on side matching and relocation. The new method achieves reversibility without using the location map. Even a tiny distortion of the original content is not applicable in some sensitive applications such as military, medical / fine art data.

Therefore the value of reversible methods of steganography is increasing. VQ (Vector Quantization) is a popular compression technique because of its simple encoding and decoding procedures. To achieve better imperceptibility the codebook is partitioned into several clusters before embedding. The input needed will be a VQ compressed image, a stream of secret bits, a super codebook SC, clusters of the super codebook SC and multiple hit maps.

The output will be a VQ stego image. Block X in the cover image will fall into one of the three following cases. If X is equal to the ith codeword of Go, the embedding process is invoked. If X is equal to the ith codeword of G1, no secret bit can be embedded and a compensation procedure is needed to avoid conflicting with case 1.

If X does not belong to G0 U G1, no secret bit can be embedded and X is skipped. Secret bits can be embedded only in case 1.

In Chin-Chen Chang et.al.'s article [33] a new approach to wet paper codes using random linear codes of small co-dimension is used which improves embedding efficiency is proposed.

To prevent from attack, the selection channel should not be publicly available even in any partial form. A possible remedy is to select it according to some side information that is in principle unavailable to the attacker (e.g.) random or that cannot be well estimated from the stego image.

Steganography with non shared selection channels requires codes for memories with defective cells also called wet paper codes. This paper provides a new tool for steganography a coding method that empowers the steganographer with the ability to use arbitrary selection channels while substantially decreasing the number of embedding changes.

The algorithm combines wet paper codes with matrix embedding arbitrary selection channels and improved embedding efficiency using random linear codes of small co-dimension.

In Zhicheng Ni et.al.'s article [34] the authors present a lossless data hiding which is robust against JPEG / JPEG 2000 compression. The image is split into 8 x 8 blocks and each block is split into two subsets (A, B).

For each block the difference value _ is calculated where _ is the arithmetic average of differences of pixel pairs within the block. This _ is selected as a robust quantity for embedding the information bit. Each bit of the secret message is associated with a group of pixels eg. A block in an image.

The bit embedding strategy used is as follows, If _ is located within a threshold & to embed bit 1, shift _ to right/left beyond a threshold by adding/subtracting a fixed number from each pixel value within one subset.

To embed 0, the block is intact. If _ is located outside the threshold, always embed 1 thus shifting the value _ away beyond a threshold. Then error correction code is applied.

Table 2.2 Statistical Steganographic Methods in chronological order starting from latest

S.No.	**Author**	**Year**	**Method Used**
1.	Tomas Filler et. al. [31]	2010	Additive distortion function in Steganography using Syndrome Trellis codes
2.	Jessica Fridrich et. al. [32]	2006	Matrix embedding with wet paper codes
3.	Chin-Chen Chang et. al. [33]	2006	Reversible embedding scheme for VQ compressed images based on side matching and relocation uses location map.
4.	Zhicheng Ni et. al.[34]	2004	Lossless Data Hiding

In M.B.Ould MEDENI et.al.'s article [35], the authors use error correcting codes in steganographic protocols. An optimal code is one that makes most of the maximum embeddable (MLE). The method referred to as matrix encoding requires the sender and recipient to agree in advance on a parity check matrix H.

The cover medium is processed to extract a sequence of symbols _, which is modified into s to embed the message m, s is sometimes called the stegodata, and modifications on s are translated on the cover-medium to obtain the stego-medium. Relation between steganographic algorithms and error correcting codes are discussed.

In D.P.Gaikwad et. al.'s paper [36]the authors propose image restoration technique in steganography. The image is blurred before hiding the message image using special point spread function and randomly generated key. Sequential LSB embedding in the R plane is done in this project. The number of rows and columns of the message image is encrypted in the first row of the cover image.

Before inserting, the original message image is blurred using the specific PSF (Point Spread Function). The parameters used for blurring with PSF are used as keys during deblurring.
The secret key values are sent through a secure channel (Tunnelling). The secret image is recovered using the two keys and a third key, which is randomly generated and depends on the content of the hiding message.

Table 2.3 Distortion Steganographic Methods in chronological order starting from latest

S.No.	**Author**	**Year**	**Method Used**
1.	M.B.Ould MEDENI et. al. [35]	2010	Use of error correcting codes in steganography
2.	D.P.Gaikwad et. al. [36]	2010	Image blurring with sequential LSB embedding

Chapter – 3

STEGANOGRAPHY AND LSB Method

3.1 Introduction

The word steganography is derived from the Greek words "stegos" meaning "cover" and "grafia" meaning "writing" defining it as "covered writing". Steganography is one such pro-security innovation in which secret data is embedded in a cover. The notion of data hiding or steganography was first introduced with the example of prisoners' secret message by Simmons in 1983.

Three techniques are interlinked, steganography, watermarking and cryptography. Following figure and table the difference among all these can be explained.

Figure 3.1 Steganography

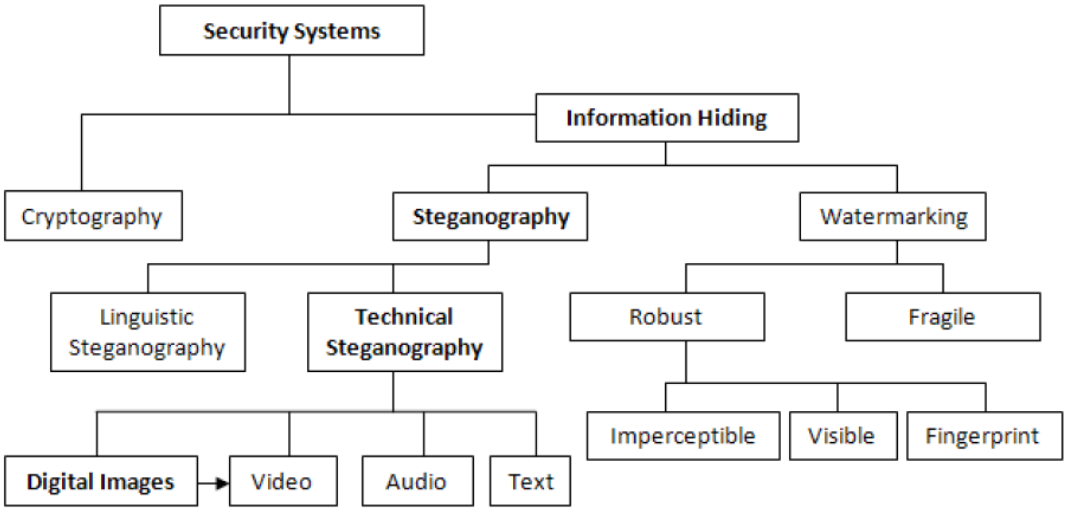


Table 3.1 Comparison of steganography, watermarking and encryption

Criterion/Method	**Steganography**	**Watermarking**	**Encryption**
Carrier	Any digital media	Mostly image/audio files	Usually text based, with some extensions to image files
Secret data	Payload	Watermark	Plain text
Key	Optional	Optional	Necessary
Input files	At least two unless in self embedding	At least two unless in self embedding	One
Detection	Blind	Usually informative (i.e., original cover or watermark is needed for recovery)	Blind
Authentication	Full retrieval of data	Usually achieved by cross correlation	Full retrieval of data
Objective	Secrete communication	Copyright preserving	Data protection
Result	Stego-file	Watermarked file	Cipher text
Concern	Delectability/capacity	Robustness	Robustness
Type of attacks	Steganalysis	Image processing	Cryptanalysis
Visibility	Never	Sometimes	Always
Fails when	It is detected	It is removed/replaced	De-ciphered
Relation to cover	Not necessarily related to the cover. The message is more	Usually becomes an attribute of the cover image. The	N/A

	important that the cover.	cover is more important than the message.	
Flexibility	Free to choose any suitable cover	Cover choice is restricted	N/A
History	Very ancient except its digital version	Modern era	Modern era

There exist two types of materials in steganography: message and carrier. Message is the secret data that should be hidden and carrier is the material that takes the message in it. There are many types of steganography methods. Fig below shows the different categories of the formats that can be used for steganography techniques.

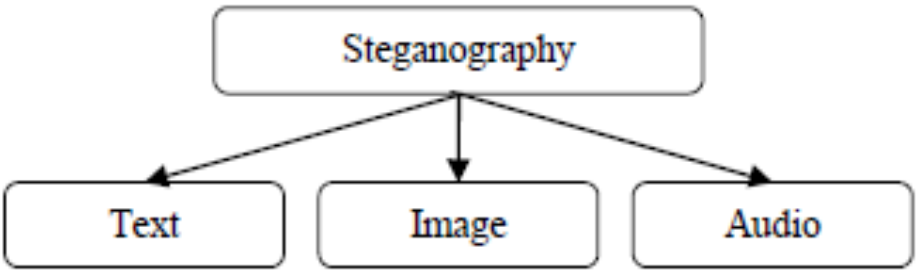


Figure 3.2 Steganography types diagram

I. Text Steganography

Text steganography can be achieved by altering the text formatting, or by altering certain characteristics of textual elements (e.g., characters). The goal in the design of coding methods is to develop alternations that are reliably decidable (even in the presence of noise) yet largely indiscernible to the reader.

II. Image Steganography

Hiding information inside images is a popular technique nowadays. An image with a secret message inside can easily be spread over the World Wide Web or in newsgroups.

The use of steganography in newsgroups has been researched by German steganographic expert NielsProvos, who created a scanning cluster which detects the presence of hidden messages inside images that were posted on the net. However, after checking one million images, no hidden messages were found, so the practical use of steganography still seems to be limited.

To hide a message inside an image without changing its visible properties, the cover source can be altered in ”noisy” areas with many color variations, so less attention will be drawn to the modifications.

The most common methods to make these alterations involve the usage of the least-significant bit or LSB, masking, filtering and transformations on the cover image. These techniques can be used with varying degrees of success on different types of image files.

III. Audio Steganography

In audio steganography, secret message is embedded into digitized audio signal which result slight altering of binary sequence of the corresponding audio file.

There are several methods are available for audio steganography. We are going to have a brief introduction on some of them.

LSB Coding

Sampling technique followed by Quantization converts analog audio signal to digital binary sequence.

In this technique LSB of binary sequence of each sample of digitized audio file is replaced with binary equivalent of secret message.

3.2 Methods of Image Steganography

There are various methods of steganography:

1 .Least significant bit (LSB) method

2. Transform domain techniques

3. Statistical methods

4. Distortion techniques

Least significant bit (LSB) method

Least significant bit (LSB) method of steganography is a simple method to embed information behind an image file. In this method the LSB(Least significant bit) of a byte is exchanged with bit of message. This technique produces good results for media steganography like image, audio and video.

To the human eye, the resultant image will look same to the original object. For example, if we use image steganography then a letter A might be hidden in three pixels (we are assuming no compression). The original data for 3 pixels (9 bytes) may be

(00100111 11101001 11001000)

(00100111 11001000 11101001)

(11001000 00100111 11101001)

The binary equivalent for A is 10000001. If we insert the binary equivalent for A in the three pixels, it will result as

(00100111 11101000 11001000)

(00100110 11001000 11101000)

(11001000 00100111 11101001)

There are only three actually changed in the 8 bytes of 3 pixels. On average, LSB requires that only one half of the bytes in an image needs be changed. We can also hide data in the least and second least significant bits as per our choice and still the general human eye will not be able to recognize the changes.

Encryption and Decryption

Encryption is the process of converting plain data (plaintext) into some other format that appears to be random and meaningless (ciphertext). Decryption is the process of translating ciphertext back to original format of text (plaintext). To encrypt more than a small amount of data, symmetric encryption is used

Chapter- 4

PROPOSED WORK

4.1 Pitfalls and Improvements

In the whole study till now, we studied the concept of LSB. We found three main issues which need to be cover in our study for efficiency and security of message which is hidden in the image.

- Storing length of message: In standard LSB method, the length of messages is stored in first 31 pixels and data is stored in next each pixel. This is the common idea given in the LSB method. So a malicious user can retrieve the length of message as first step by reading first 31 pixels. So it is essential to store length of message somewhere else.

- Storing message: After the length, the data is stored with last bit each pixel which can be easily identified and retrieved by a malicious user. We changed the algo so as hiding the message not in continuous pixel, this might not be stored in continuous pixels but to store even pixels or odd pixels or any arbitrary defined order like every 10^{th} pixel. It's also to be decided on the basis of size of message to be hidden with the image. Also the pixel number in which data is to be stored can be retrieved dynomically by retieving size of images and size of messages.

- Encoding messages: The first concen of study was to make message more and more secure. Any smart user or hacker can retrieve such data easily if the basic structure of storage of message is known. So one more implementation was given for security of data/message. We found that data should be stored in LSB after encryption so that this will be next layer of security.

For implementation of all above, we have implemented following strategies for defining the message in more secure way.

- For better image after embedding of message, we used alternative pixel for storing the

message. If we store message bit in each continous pixel, the image quality degrades so we better if we store the message in even number of pixel. Like first bit will be stored in 32nd pixel and then next bit to 34th pixel and so on.

- For security of data, length is stored in any random number of pixel in the image. As per given in most of the standards, length is being stored in first 31 pixels which can easily be retrieved by any of the person for misuse. So we have used concepts that length is to be stored in any of the arbitrary defined location which is known only to sender and receiver. Here both send and receiver works on agreed opon protocol and embed/decode the message accordingly

- Also for security of data, message is encrypted using java encryption/decryption APIs. So even if a malicious user get the message, he/she will not be able to use it without decryption. We planned to use encryption methodology by using javax.crypto APIs which is using more secure base 64 methods.

4.2 Differences of Standard LSB method and Proposed Method

Standard LSB Method	**Our LSB Method**
Uses continuous pixels of image for storing messages	Do not use continous pixels of image for storing message.
Message length is stored on specified pixels like first 31 pixels	Where the message length is, this will be known only to sender and receiver
Stegno image is almost same as the original image.	Stegno image is better then the image of original LSB method.
The location of storing message in pixels is common and static.	The location of message is not static and decided on the basis size of image,

	message and their ratio.
Original message is stored with image.	Message is stored in image after encryption.

4.3 Java

The proposed tool is developed in Java programming language. Java is an object-oriented programming language developed by James Gosling and colleagues at Sun Microsystems in the early 1990s. Unlike conventional languages which are generally designed either to be compiled to native (machine) code, or to be interpreted from source code at runtime, Java is intended to be compiled to a bytecode, which is then run (generally using JIT compilation) by a Java Virtual Machine.

There were five primary goals in the creation of the Java language:
1. It should use the object-oriented programming methodology.

2. It should allow the same program to be executed on multiple operating systems.

3. It should contain built-in support for using computer networks.

4. It should be designed to execute code from remote sources securely.

5. It should be easy to use by selecting what was considered the good parts of other object-oriented languages.

Other than a lots of java language features, we used javax.swing and javax.cypto packages in the proposed tool. So here we will discuss only these two packages and respective classes which are being in use in our tool.

4.4 Javax.swing Package

The Swing package is part of the JavaTM Foundation Classes (JFC) in the Java platform. The JFC encompasses a group of features to help in building GUIs (graphical user interfaces). Swing provides components such as panels, buttons, selection boxes, etc.

Previous versions of Java (jdk 1.0, 1.1) have used AWT package for these purposes, which provides similar features. Although the Java 2 Platform still supports the AWT components, Swing components are easier to use and provide far more functionality. You can identify Swing components because their names start with J. The AWT button class, for example, is named Button, whereas the Swing button class is named JButton. In addition, the AWT components are in the java.awt package, whereas the Swing components are in the javax.swing package.

The most commonly used components are those used for control (s.a. JButton, JRadioButton, JCheckBox), menus (s.a. JMenuBar, JPopupMenu, menus also may be constructed out of buttons), text areas (JTextArea, JTextField, JPasswordField, etc.), tables, and many others. Some components are themselves containers, so you can add other components to them. Those include JPanel (a general-purpose container, most commonly used to implement nesting of components), JScrollPane, which rovides a scrollable view of components, JSplitPane, which displays two components, either side by side or one on top of the other, with a divider that one can drag to specify how much of the split pane's area goes to each component.

Although swing has thousands of classes and a lots of packages but we have used a very small number of classes in our proposed tool. In our proposed tool, we have used following swing classes/components

- **JFrame:** The JFrame class is slightly incompatible with Frame. Like all other JFC/Swing top-level containers, a JFrame contains a JRootPane as its only child. The content pane provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame. Unlike a Frame, a JFrame has some notion of how to respond when the user attempts to close the window. The default behavior is to simply hide the JFrame when the user closes the window.
- **JButton**: An implementation of a "push" button. Buttons can be configured, and to some degree controlled, by Actions. Using an Action with a button has many benefits beyond directly configuring a button.
- **JSplitPane:** JSplitPane is used to divide two (and only two) Components. The two Components are graphically divided based on the look and feel implementation, and the two Components can then be interactively resized by the user.

- **JPanel:** JPanel is a generic lightweight container.
- **JFileChooser**: JFileChooser provides a simple mechanism for the user to choose a file.
- **JLabel:** A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it. A JLabel object can display either text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.
- **JOptionPane:** JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something. All dialogs are modal. Each showXxxDialog method blocks the caller until the user's interaction is complete.
- **JTextArea:** A JTextArea is a multi-line area that displays plain text. It is intended to be a lightweight component that provides source compatibility with the java.awt.TextArea class where it can reasonably do so.

4.5 Javax.crypto Package: The javax.crypto package defines classes and interfaces for various cryptographic operations. The central class is Cipher, which is used to encrypt and decrypt data. CipherInputStream and CipherOutputStream are utility classes that use a Cipher object to encrypt or decrypt streaming data. SealedObject is another important utility class that uses a Cipher object to encrypt an arbitrary serializable Java object. The KeyGenerator class creates the SecretKey objects used by Cipher for encryption and decryption. SecretKeyFactory encodes and decodes SecretKey objects. TheKeyAgreement class enables two or more parties to agree on a SecretKey in such a way that an eavesdropper cannot determine the key. The Mac class computes a message authentication code (MAC) that can ensure the integrity of a transmission between two parties who share a SecretKey. A MAC is akin to a digital signature, except that it is based on a secret key instead of a public/private key pair.

Although javax.crypto has a lots of classes and packages but we have used a very small number of classes in our proposed tool. In our proposed tool, we have used following classes of crypto package:

Cipher: This class provides the functionality of a cryptographic cipher for encryption and decryption. It forms the core of the Java Cryptographic Extension (JCE) framework. In order to create a Cipher object, the application calls the Cipher's getInstance method, and passes the name of the requested transformation to it. Optionally, the name of a provider may be specified.

A transformation is a string that describes the operation (or set of operations) to be performed on the given input, to produce some output. A transformation always includes the name of a cryptographic algorithm (e.g., DES), and may be followed by a feedback mode and padding scheme.

A transformation is of the form:

- "algorithm/mode/padding" or
- "algorithm"

(in the latter case, provider-specific default values for the mode and padding scheme are used). For example, the following is a valid transformation:

```
Cipher c = Cipher.getInstance("DES/CBC/PKCS5Padding");
```

Using modes such as CFB and OFB, block ciphers can encrypt data in units smaller than the cipher's actual block size. When requesting such a mode, you may optionally specify the number of bits to be processed at a time by appending this number to the mode name as shown in the "DES/CFB8/NoPadding" and "DES/OFB32/PKCS5Padding" transformations. If no such number is specified, a provider-specific default is used. (For example, the SunJCE provider uses a default of 64 bits for DES.) Thus, block ciphers can be turned into byte-oriented stream ciphers by using an 8 bit mode such as CFB8 or OFB8.

Modes such as Authenticated Encryption with Associated Data (AEAD) provide authenticity assurances for both confidential data and Additional Associated Data (AAD) that is not encrypted

KeyGenerator: This class provides the functionality of a secret (symmetric) key generator.

Key generators are constructed using one of the `getInstance` class methods of this class. KeyGenerator objects are reusable, i.e., after a key has been generated, the same KeyGenerator object can be re-used to generate further keys. There are two ways to generate a key: in an algorithm-independent manner, and in an algorithm-specific manner. The only difference between the two is the initialization of the object:

- **Algorithm-Independent Initialization**

  All key generators share the concepts of a keysize and a source of randomness. There is an `init` method in this KeyGenerator class that takes these two universally shared types of arguments. There is also one that takes just a `keysize`argument, and uses the SecureRandom implementation of the highest-priority installed provider as the source of randomness (or a system-provided source of randomness if none of the installed providers supply a SecureRandom implementation), and one that takes just a source of randomness.

  Since no other parameters are specified when you call the above algorithm-independent `init` methods, it is up to the provider what to do about the algorithm-specific parameters (if any) to be associated with each of the keys.

- **Algorithm-Specific Initialization**

  For situations where a set of algorithm-specific parameters already exists, there are two `init` methods that have an `AlgorithmParameterSpec` argument. One also has a `SecureRandom` argument, while the other uses the SecureRandom implementation of the highest-priority installed provider as the source of randomness (or a system-provided source of randomness if none of the installed providers supply a SecureRandom implementation).

In case the client does not explicitly initialize the KeyGenerator (via a call to an `init` method), each provider must supply (and document) a default initialization. Every implementation of the Java platform is required to support the following standard `KeyGenerator` algorithms with the keysizes in parentheses:

- AES (128)
- DES (56)
- DESede (168)
- HmacSHA1
- HmacSHA256

SecretKey: This interface contains no methods or constants. Its only purpose is to group (and provide type safety for) secret keys. Provider implementations of this interface must overwrite the equals and hashCode methods inherited from java.lang.Object, so that secret keys are compared based on their underlying key material and not based on reference.

Keys that implement this interface return the string RAW as their encoding format (see getFormat), and return the raw key bytes as the result of a getEncoded method call. (The getFormat and getEncoded methods are inherited from thejava.security.Key parent interface.)

SecretKeySpec: This class specifies a secret key in a provider-independent fashion.It can be used to construct a SecretKey from a byte array, without having to go through a (provider based) SecretKeyFactory. This class is only useful for raw secret keys that can be represented as a byte array and have no key parameters associated with them, e.g., DES or Triple DES keys.

Chapter- 5

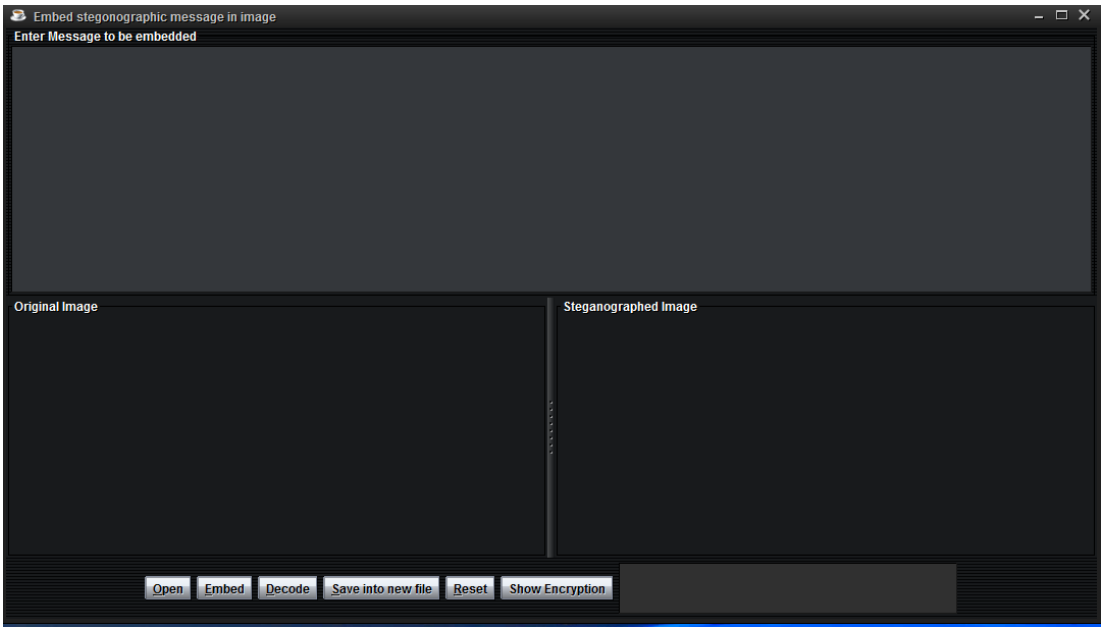
IMPLEMENTATION DETAILS

5.1 Screenshots

MessageAdd.java

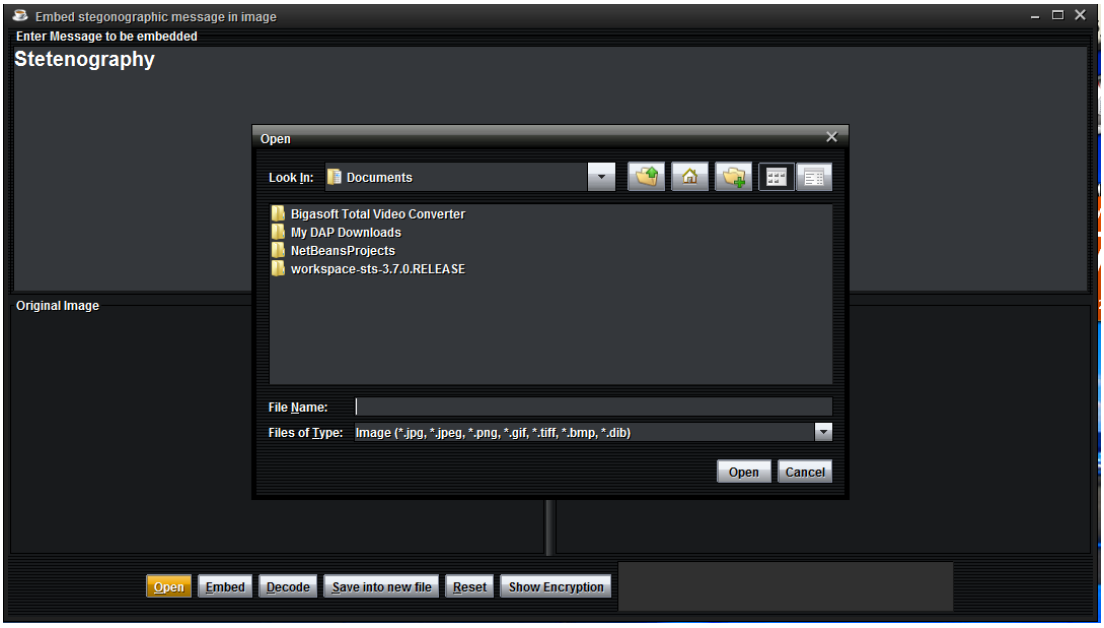
//MainScreen

figure 5.1



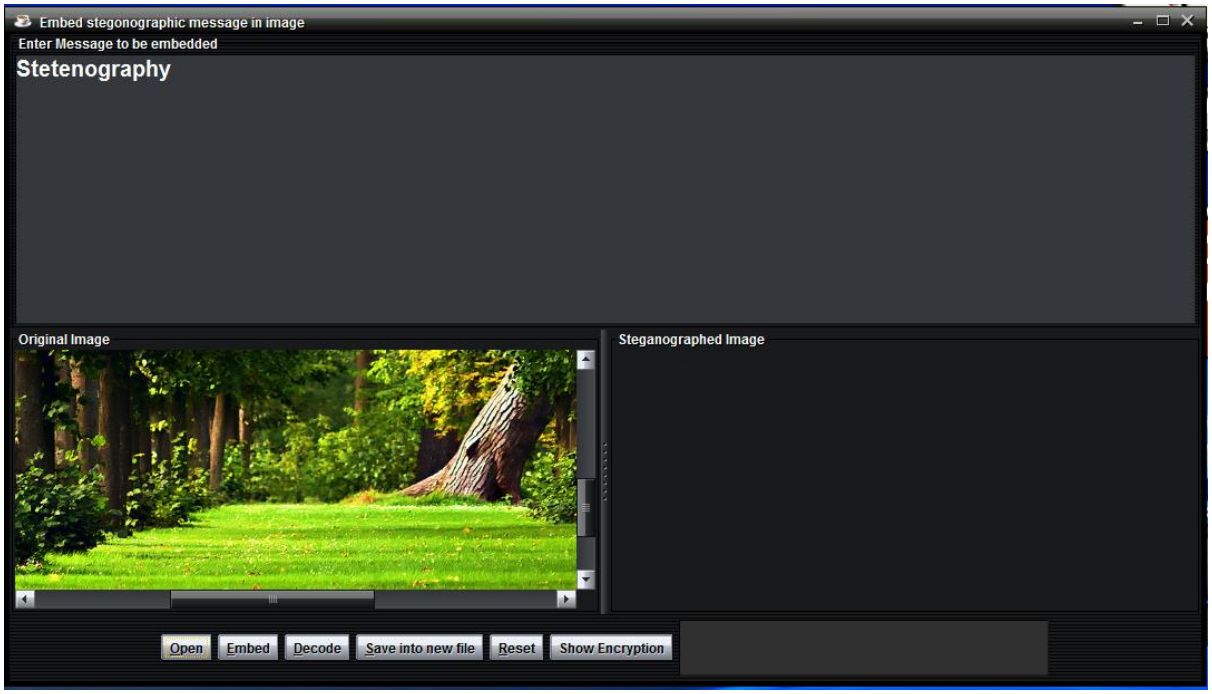
This is the main screen. Message can be given in the text area given on upper portion. In the lower panel, original image can be taken. In the lower panel, right panel the image with message are shown.

// open image Figure 5.2



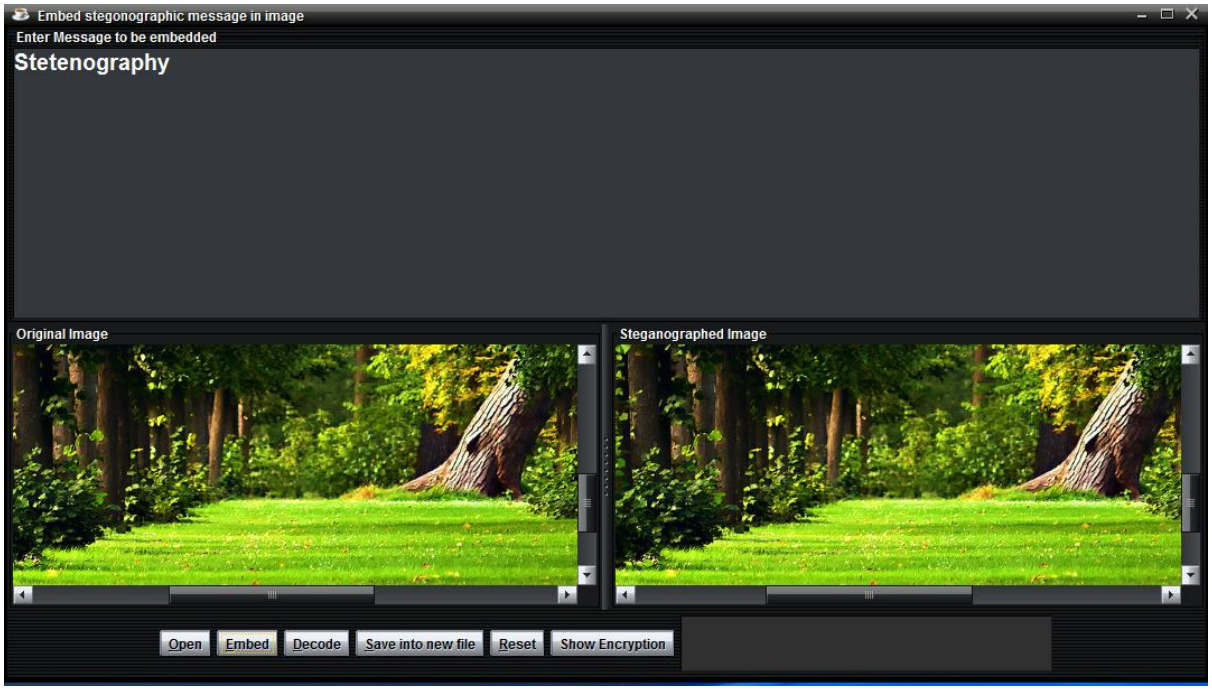
On the main screen, if open button is clicked, a open file dialog box is shown. By this screen original image can be select.

// after image open



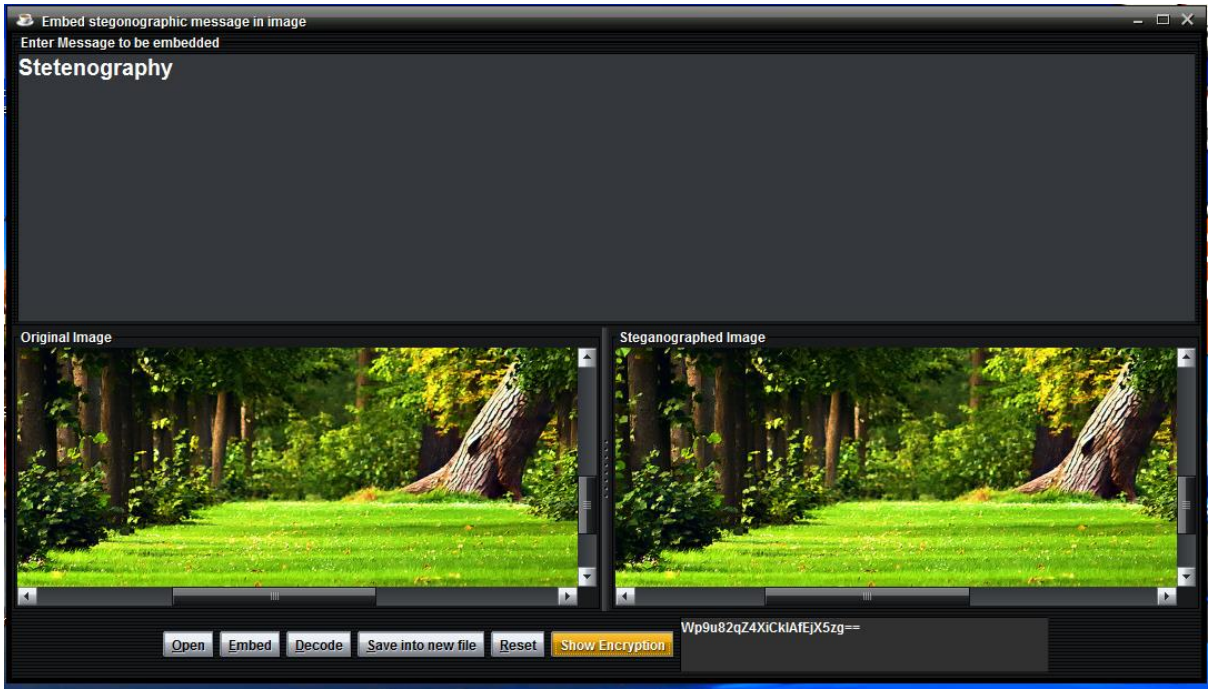
After selection of image, it is shown in left side of lower panel. If the image is large then automatically scroll bar is added in image.

// embed message in image



After giving the message in text area and image in left panel, steged image is shown on lower right panel.

// show encrypt message

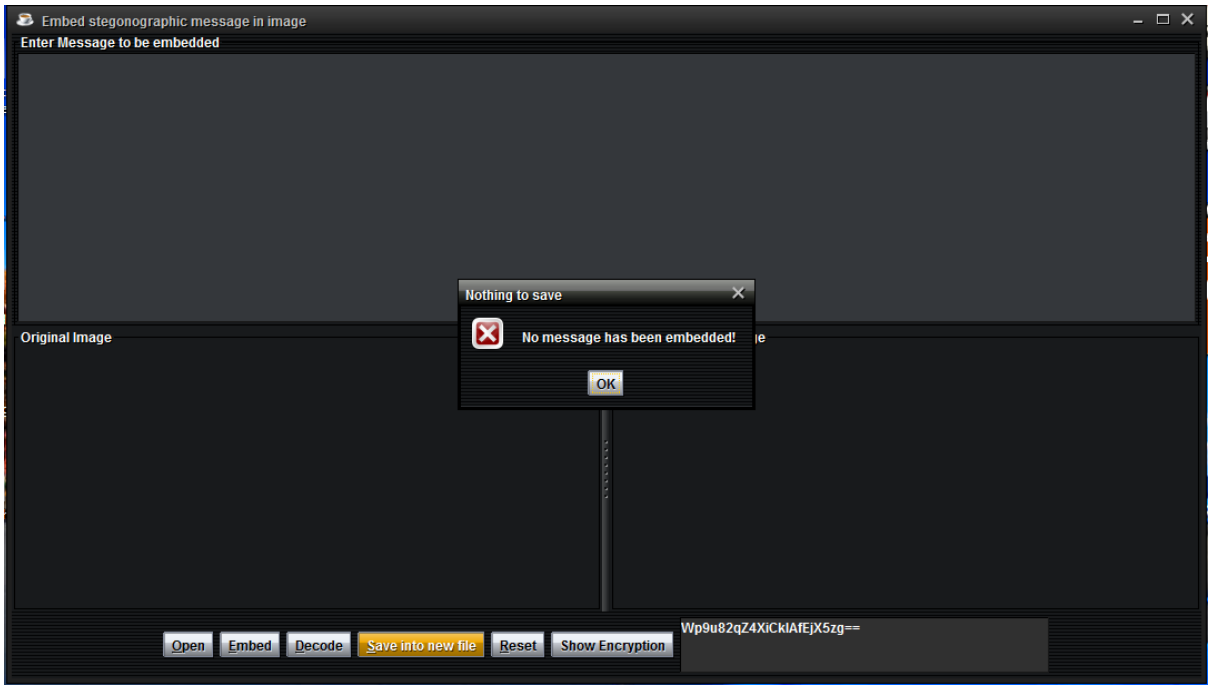


On clicking on show encryption, encrypted message is shown on lower panel.

// save stego image

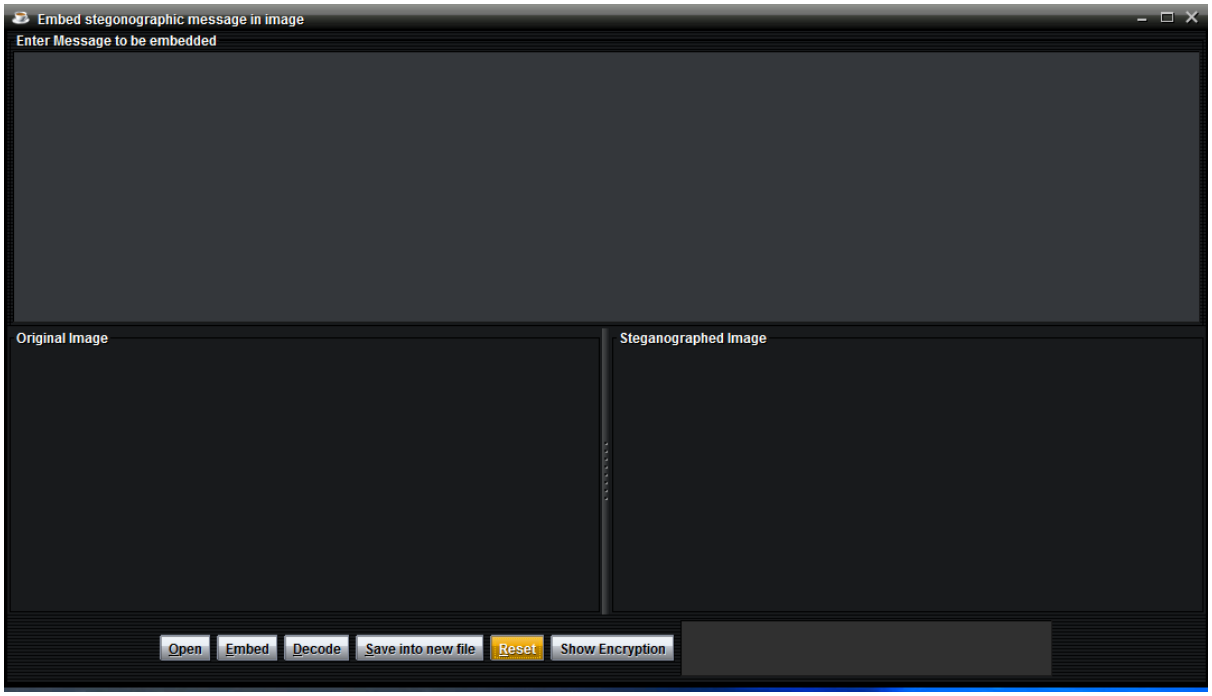
This screen is for saving the stegged image on a given location.

// error



Error message if no message is embedded.

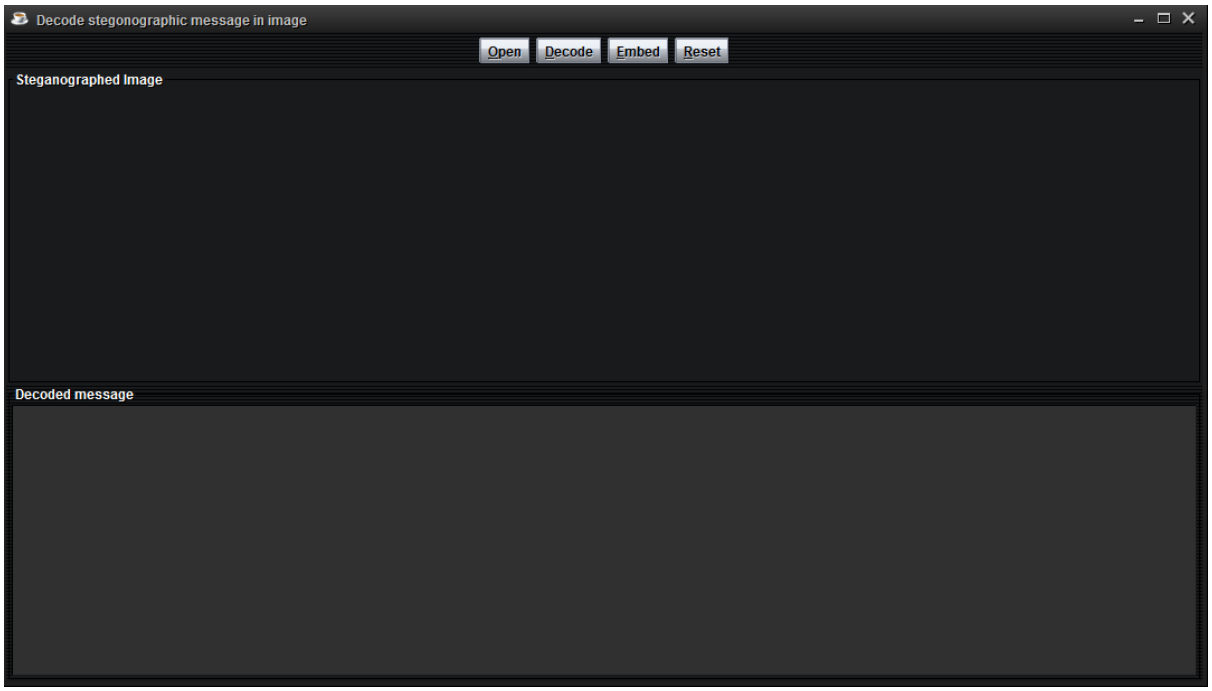
// Reset click



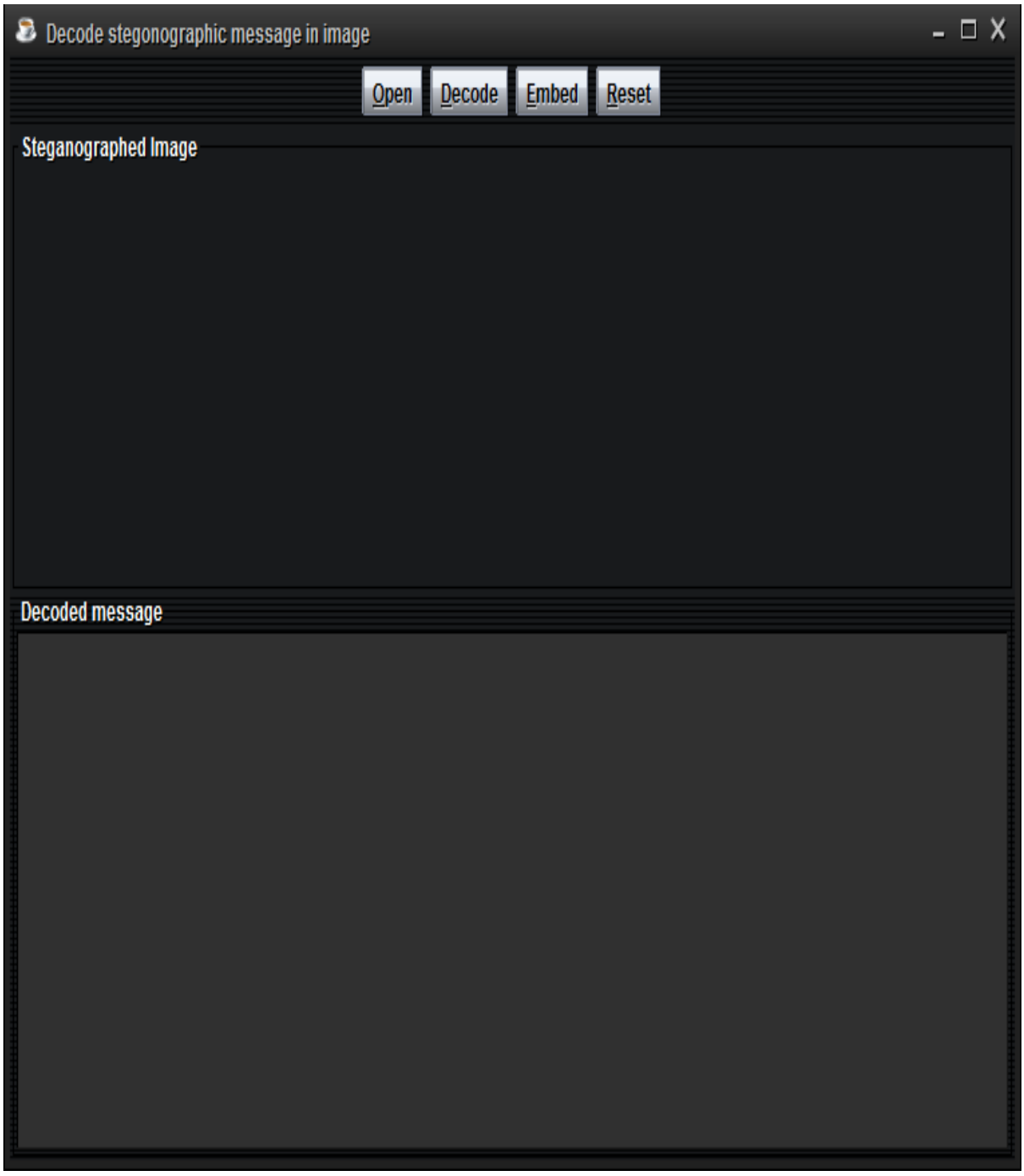
On clicking on Reset button the whole screen get clear.

MessageExtract

// MainScreen

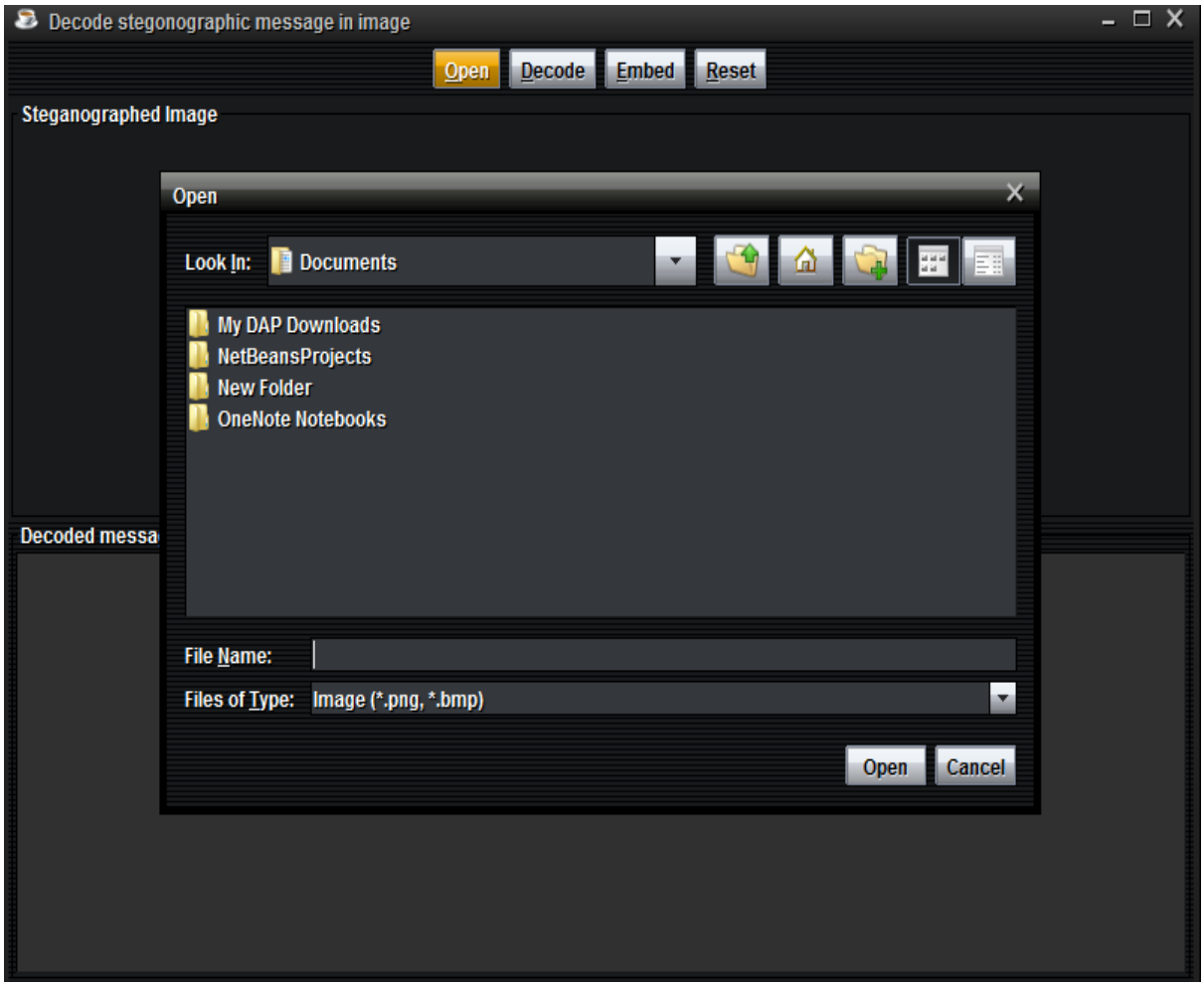


DecodeMessage.java figure 5.3



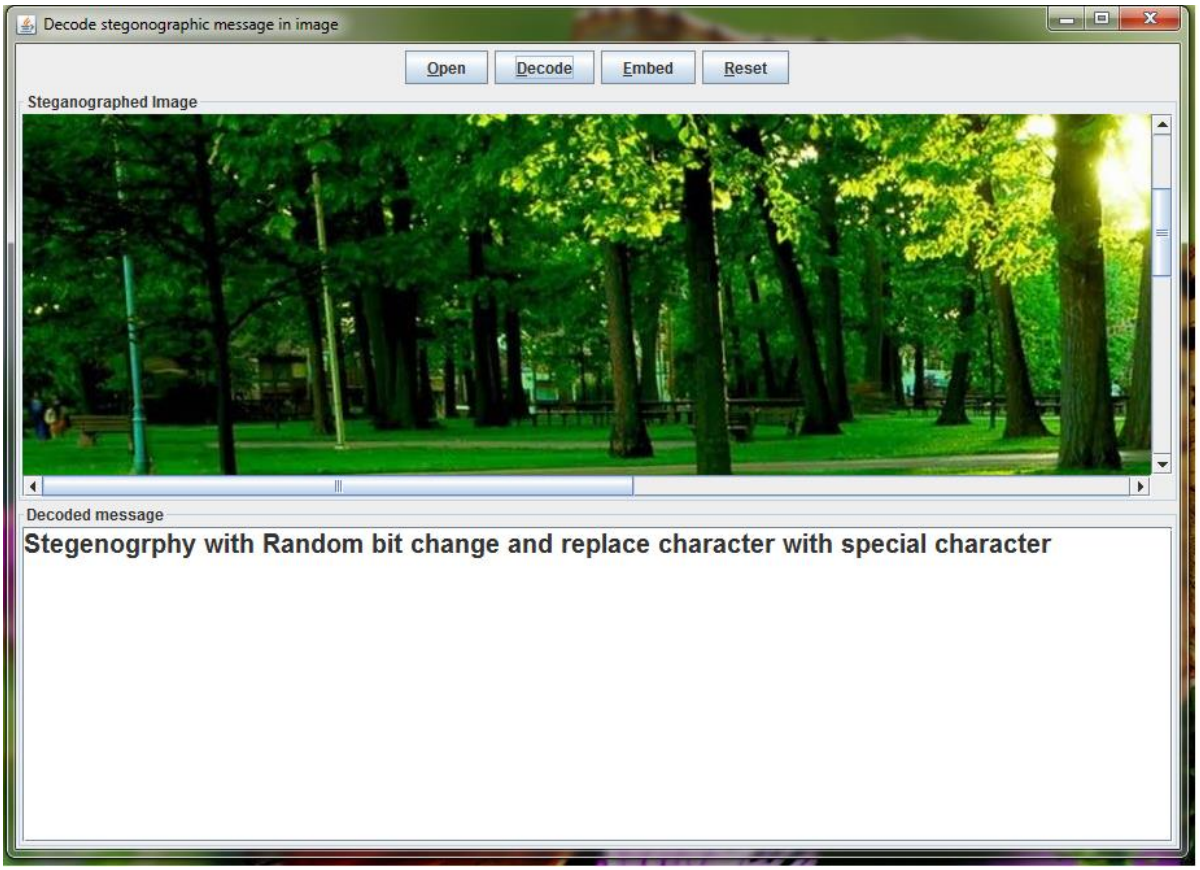
For decoding the above screen can be shown. After selection of steged image, decoded message is shown in the text area which is given on lower panel (Text area).

Figure 5.4



For decoding message from image, open button is clicked by which steged image can be selected.

Figure 5.5



After selection of a image, decoded message is shown on lower panel.

5.2 Class, method details and Algorithms

MessageAdd class Methods

Modifier and Type	Method and Description
private void	assembleInterface() This method used for add components and containers on Frame And setting their layout. Which provide user interface to user. Algorithm 1. Create a panel and set FlowLayout of it. 2. Add buttons open, embed, decode , save, reset on it. 3. Set listener and set Mnemonic on these buttons. 4. Create another Panel, set GridLayout of it. 5. It add scrollpane on panel and split it.
private void	embedByte(java.awt.image.BufferedImage img, byte b, int start, int storageBit) This method take 4 argument 1. Image to embed msg. 2. Byte to be store 3. Start position of store bit. 4. storageBit assume it and take value 0. This Method add encrypted message in Image. This method calculate height and width of image. Algorithm 1. First calculate max height and width of image. 2. Select random pixel of image repeate step 3 to 6 Encrypted message length. 3. Calculate RGB of current pixel.

	4. Calculate key value using getBitValue method. 5. Calculate value which store in image using setBitValue method 6. Set calculated bit in current pixel of image using setRGB method.
private void	embedInteger(java.awt.image.BufferedImage img, int n, int start, int storageBit) This method take 4 argument 1. Image to embed length of Encrypted Message. 2. Length of Encrypted Message. 3. Start position of store bit. 4. storageBit assume it and take value 0. This Method add length of Encrypted Message in the Image in random pixel of image. Algorithm 1. First calculate max height and width of image. 2. Select pixel of image,step 3 to 6 repeated first 31 pixels of image. 3. Calculate RGB of current pixel. 4. Calculate key value using getBitValue method. 5. Calculate value which store in image using setBitValue method 6. Set calculated bit in current pixel of image using setRGB method.
static byte[]	encrypt(java.lang.String Data) This method take String to be encrypt. It used AES Algorithm to encrypt Message and Base64 to encode message.

	Algorithm 1. Take string to be Encrypt. 2. Calculate key for encryption 3. Create Cipher's object basis on key. 4. It encrypts string data and Encodes using Base64Encoder. 5. And return encrypted message.
private static java.security.Key	generateKey() This Method Generate Key to encrypt Message basis on given bytes. Algorithm 1. Use SecretKeySpec. 2. SecretKeySpec's constructor take user define bytes array to generate Key.
private int	getBitValue(int n, int location) This method take 2 argument 1. Byte value which is store in image. 2. Location of bit. This Method calculates the Key value for Image Basis on Message character and Pixel's RGB value. Return key value. Algorithm 1. Take byte to generate key value. 2. It return 0 or 1 bit for current pixel.
private void	MessageAdd() use for hidden message in image. Algorithm

	1. First Extract text from TextArea. 2. Calculate subimage from actual image. 3. call MessageAdd Method. 4. Create Label. 5. Set icon on Label.
private void	MessageAdd(java.awt.image.BufferedImage img, java.lang.String mess) it Method take Image and Message argument.This Method first Encrypt the message and add encrypted message on random pixel in image. Algorithm 1. First calculate message length. 2. Calculate height and width of image. 3. Get Key value. 4. Encrypt Message using encrypt Method. 5. Set Length of Encrypted message using embedInteger method. 6. Set Encrypted message in image using embedByte Method. 7. Repeate embedByte method encryptedBytes's length.
private void	openImage() openImage() Method Open the Image which are selected by File Chooser. This Image is used for Hide Message and it's length. Algorithm 1. Get File from FileDialog. 2. Read image from File.

	3. Create Label and add image on it. 4. Add label on originalPane.
private void	resetInterface() This Method Reset all component and set in starting position. Algorithm 1. Reset TextArea 2. Remove all component from ScrollPane. 3. Remove all images.
private void	saveImage() This Method save the Stego image which have hidden encrypted Message and it's length at specific location. Algorithm 1. Get File to be save. 2. Get Name of File. 3. Set extension of File(png). 4. And Write Image at particular location.
private int	setBitValue(int n, int location, int bit) This method take 3 argument 1. RGB of current pixel. 2. Location assumes it and takes value 0. 3. Key value which is calculated. This Method calculates a value using LSB Algorithm witch adds in image. Algorithm 1. Calculate value of this location.

	2. Get value to be stored this position. 3. Match both values using LSB algorithm. 4. And return int value.
private java.io.File	showFileDialog(boolean open) This Method use for show File Dialog and Filter File like jpg,png,gif etc. Algorithm 1. Create FileChooser's object. 2. Using Filter for Filter Image. 3. Select File. 4. Return Selected File.
void	actionPerformed(java.awt.event.ActionEvent ae) It take 1 argument of ActionEvent class generated by JVM when event on button.This Method performed on clicked any Button. Algorithm 1. Get source of object. 2. Check which button clicked. 3. Call particular method.
static void	main(java.lang.String[] arg) This is Main Method of Class, where program is Started. Algorithm 1. Set Look and Feel of Frame. 2. Create object of this class. 3. Visible of created frame.

MessageExtract class Methods

Modifier and Type	Method and Description
void	actionPerformed(java.awt.event.ActionEvent ae) This Method performed on click on any Button. It take 1 argument of ActionEvent class generated by JVM when event on button.This Method performed on clicked any Button. Algorithm 1. Get source of object. 2. Check which button clicked. 3. Call particular method.
private void	assembleInterface() This method used for add components and containers on Frame And setting their layout. Which provide user interface to user. Algorithm 1. Create a panel and set FlowLayout of it. 2. Add buttons open, embed, decode , save, reset on it. 3. Set listener and set Mnemonic on these buttons. 4. Create another Panel, set GridLayout of it. 5. It add scrollpane on panel and split it.
static byte[]	decrypt(java.lang.String encryptedData) This Method take String which decrypt. This Method Decrypt message from return by image using AES

	algorithm and Decode64 to decoding message. Algorithm 1. Take string to be Decrypt. 2. Get key for decryption 3. Create Cipher's object basis on key. 4. It decrypts string data and decodes using Base64Encoder. 5. And return Decrypted message.
private byte	extractByte(java.awt.image.BufferedImage img, int start, int storageBit) This method take 3 argument 1. Stego Image whitch has hidden msg. 2. Start position of store bit. 3. storageBit assume it and take value 0. This Method extract encrypted message from Image. And return byte from stego image.
private int	extractInteger(java.awt.image.BufferedImage img, int start, int storageBit) This method take 3 argument 1. Stego Image whitch has hidden length of Encrypted msg. 2. Start position of store bit. 3. storageBit assume it and take value 0. This Method return length of encrypted message from stego image.
private int	getBitValue(int n, int location) This method take 2 argument 1. RGB value of current pixel. 2. storageBit assume it and take value 0.

	This Method calculates key value and return key it. Algorithm 1. Take byte to generate key value. 2. It return 0 or 1 bit for current pixel.
private void	MessageExtract() use for extract encrypted Message from stego image and decrypt it.
private void	openImage() openImage() Method Open the Image which are selected by File Chooser. And using Filter for filte image like png,jpeg,bmp,jpg. openImage() Method Open the Image which are selected by File Chooser. This Image is used for Hide Message and it's length. Algorithm 1. Get File from FileDialog. 2. Read image from File. 3. Create Label and add image on it. 4. Add label on originalPane.
private void	resetInterface() This Method Reset all component in starting position. Algorithm 1. Reset TextArea 2. Remove all component from ScrollPane.

	3. Remove all images.
private int	setBitValue(int n, int location, int bit) This method take 3 argument 1. Assume Default value of stored bit is 0. 2. Position of bit value. 3. Key value which is calculated. This Method calculates bit value, which stored in stego image. It return byte to stored and set actual RGB value in image. Algorithm 1. Calculate value of this location. 2. Get value to be stored this position. 3. Match both values using LSB algorithm. 4. And return int value.
private java.io.File	showFileDialog(boolean open) This Method use for show File Dialog and Filter File like jpg,png,gif etc. Algorithm 1. Create FileChooser's object. 2. Using Filter for Filter Image. 3. Select File. 4. Return Selected File.
private void	assembleInterface() This method used for add components and containers on Frame And setting their layout. Which provide user interface to user.

	Algorithm 6. Create a panel and set FlowLayout of it. 7. Add buttons open, embed, decode , save, reset on it. 8. Set listener and set Mnemonic on these buttons. 9. Create another Panel, set GridLayout of it. 10. It add scrollpane on panel and split it.

5.3 Code

File MessageAdd.java

```
import java.awt.image.*;
 import javax.swing.*;
 import java.awt.*;
 import java.awt.event.*;
 import javax.imageio.*;

 public class MessageAdd extends JFrame implements ActionListener
 {
 JButton open = new JButton("Open"), embed = new JButton("Embed"),
   save = new JButton("Save into new file"),decode = new JButton("Decode"), reset =
new JButton("Reset");

 JTextArea message = new JTextArea(10,3);

 BufferedImage sourceImage = null, embeddedImage = null;

 JSplitPane sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
 JScrollPane originalPane = new JScrollPane(),
   embeddedPane = new JScrollPane();
```

```
public MessageAdd() {
  super("Embed stegonographic message in image");
  assembleInterface();

  this.setDefaultCloseOperation(EXIT_ON_CLOSE);
  this.setBounds(GraphicsEnvironment.getLocalGraphicsEnvironment().
    getMaximumWindowBounds());
  this.setVisible(true);
  sp.setDividerLocation(0.5);
  this.validate();
  }

private void assembleInterface() {
  JPanel p = new JPanel(new FlowLayout());
  p.add(open);
  p.add(embed);
  p.add(decode);
  p.add(save);
  p.add(reset);
  this.getContentPane().add(p, BorderLayout.SOUTH);
  open.addActionListener(this);
  embed.addActionListener(this);
  save.addActionListener(this);
  reset.addActionListener(this);
  decode.addActionListener(this);
  open.setMnemonic('O');
  embed.setMnemonic('E');
  save.setMnemonic('S');
  reset.setMnemonic('R');
  decode.setMnemonic('D');
```

```
    p = new JPanel(new GridLayout(1,1));
    p.add(new JScrollPane(message));
    message.setFont(new Font("Arial",Font.BOLD,20));
    p.setBorder(BorderFactory.createTitledBorder("Message to be embedded"));
    this.getContentPane().add(p, BorderLayout.NORTH);

    sp.setLeftComponent(originalPane);
    sp.setRightComponent(embeddedPane);
    originalPane.setBorder(BorderFactory.createTitledBorder("Original Image"));
    embeddedPane.setBorder(BorderFactory.createTitledBorder("Steganographed
Image"));
    this.getContentPane().add(sp, BorderLayout.CENTER);
    }

 public void actionPerformed(ActionEvent ae) {
    Object o = ae.getSource();
    if(o == open)
      openImage();
    else if(o == embed)
      MessageAdd();
    else if(o == save)
      saveImage();
    else if(o == reset)
      resetInterface();
     else if(o == decode){
     dispose();
          new MessageExtract();
     }

    }
```

```
private java.io.File showFileDialog(final boolean open) {
  JFileChooser fc = new JFileChooser("Open an image");
  javax.swing.filechooser.FileFilter ff = new javax.swing.filechooser.FileFilter() {

    public boolean accept(java.io.File f) {
      String name = f.getName().toLowerCase();

      if(open) // if true
        return f.isDirectory() || name.endsWith(".jpg") || name.endsWith(".jpeg") ||
          name.endsWith(".png") || name.endsWith(".gif") || name.endsWith(".tiff") ||
          name.endsWith(".bmp") || name.endsWith(".dib");

      return f.isDirectory() || name.endsWith(".png") ||    name.endsWith(".bmp");
      }

    public String getDescription() {

      if(open)  // true
        return "Image (*.jpg, *.jpeg, *.png, *.gif, *.tiff, *.bmp, *.dib)";

      return "Image (*.png, *.bmp)";

      }
    };
  fc.setAcceptAllFileFilterUsed(false);
  fc.addChoosableFileFilter(ff);

  java.io.File f = null;   // Get File as Image
  if(open && fc.showOpenDialog(this) == fc.APPROVE_OPTION)
    f = fc.getSelectedFile();
```

```
    else if(!open && fc.showSaveDialog(this) == fc.APPROVE_OPTION)
      f = fc.getSelectedFile();
    return f;
    }

  private void openImage() {
    java.io.File f = showFileDialog(true);
    try {
      sourceImage = ImageIO.read(f);
      JLabel l = new JLabel(new ImageIcon(sourceImage));
      originalPane.getViewport().add(l);
      this.validate();
      } catch(Exception ex) { ex.printStackTrace(); }
    }

  private void MessageAdd() {
    String mess = message.getText();
    embeddedImage = sourceImage.getSubimage(0,0,
      sourceImage.getWidth(),sourceImage.getHeight());

   MessageAdd(embeddedImage, mess);      // call MessageAdd      and initialize
"embeddedImage" varible

    JLabel l = new JLabel(new ImageIcon(embeddedImage));
    embeddedPane.getViewport().add(l);
    this.validate();
    }

  private void MessageAdd(BufferedImage img, String mess) {
    int messageLength = mess.length();
```

```
    int imageWidth = img.getWidth(), imageHeight = img.getHeight(),
      imageSize = imageWidth * imageHeight;

    if(messageLength * 16 + 32 > imageSize) {
      JOptionPane.showMessageDialog(this, "Message is too long for the chosen image",
        "Message too long!", JOptionPane.ERROR_MESSAGE);
      return;
      }

      System.out.println("Message Length "+messageLength);  // see info

    embedInteger(img, messageLength, 0, 0); // set Length

    byte b[] = mess.getBytes();

    for(int i=0; i<b.length; i++)
      embedByte(img, b[i], i*16+32, 0);  // set Message
    }

  private void embedInteger(BufferedImage img, int n, int start, int storageBit) {
      int maxX = img.getWidth(), maxY = img.getHeight();
      int startX = start/maxY, startY = start - startX*maxY;
      int count=0;

      System.out.println("width  "+maxX+"  height  "+maxY+"  startX  "+startX+"  startY
"+startY);

    for(int i=startX; i<maxX && count<32; i++) {
      for(int j=startY; j<maxY && count<32; j++) {

        int rgb = img.getRGB(i, j);
```

```
        System.out.println("i "+i+" j "+j+" count "+count+" RGB "+rgb );   // rgb calulate

        int bit = getBitValue(n, count);
        rgb = setBitValue(rgb, storageBit, bit);
        img.setRGB(i, j, rgb);



        count++;
        }
      }
    }
    int even = 0;

 private void embedByte(BufferedImage img, byte b, int start, int storageBit) {
          System.out.println("Byte b "+b+"\n \n");
    int maxX = img.getWidth(), maxY = img.getHeight();

    int startX = start/maxY, startY = start - startX*maxY, count=0;

      System.out.print("width  "+maxX+"  height  "+maxY+"  startX  "+startX+"  startY
"+startY);
     System.out.println(" startX "+startX+" startY "+startY);
          int bb = b;
          System.out.println("bbbbbb "+bb);

          if(bb%2 == 0)
        bb+=2;
     else
        bb-=2;
```

```
            System.out.println("after even odd  "+bb);

    for(int i=startX; i<maxX && count<8; i++) {
      for(int j=startY; j<maxY && count<8; j++) {




            if(even%2 == 0)
            {
                    int rgb = img.getRGB(i, j);

                     System.out.println("i "+i+" j "+j+" count "+count+" RGB "+rgb );    // rgb
calulate


                    int bit = getBitValue(bb, count);
                    rgb = setBitValue(rgb, storageBit, bit);
                    img.setRGB(i, j, rgb);
                    count++;
                    even = 1;
            } else {
                    even = 0;
            }

            even = 0;
          }
       }
    }

 private void saveImage() {
```

```
  if(embeddedImage == null) {
    JOptionPane.showMessageDialog(this, "No message has been embedded!",
      "Nothing to save", JOptionPane.ERROR_MESSAGE);
    return;
    }
  java.io.File f = showFileDialog(false);
  String name = f.getName();
  String ext = name.substring(name.lastIndexOf(".")+1).toLowerCase();
  if(!ext.equals("png") && !ext.equals("bmp") &&   !ext.equals("dib")) {
      ext = "png";
      f = new java.io.File(f.getAbsolutePath()+".png");
      }

  try {
    if(f.exists()) f.delete();
    ImageIO.write(embeddedImage, ext.toUpperCase(), f);
    } catch(Exception ex) { ex.printStackTrace(); }
  }

private void resetInterface() {
  message.setText("");
  originalPane.getViewport().removeAll();
  embeddedPane.getViewport().removeAll();
 sourceImage = null;
  embeddedImage = null;
  sp.setDividerLocation(0.5);
  this.validate();
  }

private int getBitValue(int n, int location) {    //  getBit
  int v = n & (int) Math.round(Math.pow(2, location));
```

```
    System.out.println("get n "+n+" Location "+location+" V "+v);
    return v==0?0:1;
    }

  private int setBitValue(int n, int location, int bit) {   // setBit
    int toggle = (int) Math.pow(2, location);

    System.out.println("set n: "+n+" loc "+location);

    int bv = getBitValue(n, location);
     System.out.println("set bv: "+bv+" bit "+bit);

    if(bv == bit)
    {
          System.out.println("Set n after condtion "+n+"\n");
          return n;

    }

    if(bv == 0 && bit == 1)
      n |= toggle;
    else if(bv == 1 && bit == 0)
      n ^= toggle;

     System.out.println("Set n after condtion "+n+"\n");  // Value decrease if GRB in minus
value
    return n;
    }

  public static void main(String arg[]) {
```

```
    new MessageAdd();
    }
}
```

File MessageExtract.java

```
import java.awt.image.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.imageio.*;

public class MessageExtract extends JFrame implements ActionListener
{
JButton open = new JButton("Open"), decode = new JButton("Decode"),
    reset = new JButton("Reset"),embed = new JButton("Embed");
JTextArea message = new JTextArea(10,3);
BufferedImage image = null;
JScrollPane imagePane = new JScrollPane();

public MessageExtract() {
    super("Decode stegonographic message in image");
    assembleInterface();

    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setBounds(GraphicsEnvironment.getLocalGraphicsEnvironment().
        getMaximumWindowBounds());
    this.setVisible(true);
    }

private void assembleInterface() {
```

```
    JPanel p = new JPanel(new FlowLayout());
    p.add(open);
    p.add(decode);
    p.add(embed);
   p.add(reset);
    this.getContentPane().add(p, BorderLayout.NORTH);
    open.addActionListener(this);
    decode.addActionListener(this);
    reset.addActionListener(this);
    embed.addActionListener(this);
    open.setMnemonic('O');
    decode.setMnemonic('D');
    embed.setMnemonic('E');
    reset.setMnemonic('R');

    p = new JPanel(new GridLayout(1,1));
    p.add(new JScrollPane(message));
    message.setFont(new Font("Arial",Font.BOLD,20));
    p.setBorder(BorderFactory.createTitledBorder("Decoded message"));
    message.setEditable(false);
    this.getContentPane().add(p, BorderLayout.SOUTH);

    imagePane.setBorder(BorderFactory.createTitledBorder("Steganographed Image"));
    this.getContentPane().add(imagePane, BorderLayout.CENTER);
    }

  public void actionPerformed(ActionEvent ae) {
    Object o = ae.getSource();
    if(o == open)
      openImage();
    else if(o == decode)
```

```
    MessageExtract();
  else if(o == reset)
    resetInterface();
  else if(o == embed){
        dispose();
        new MessageAdd();
  }



  }

private java.io.File showFileDialog(boolean open) {
  JFileChooser fc = new JFileChooser("Open an image");
  javax.swing.filechooser.FileFilter ff = new javax.swing.filechooser.FileFilter() {
    public boolean accept(java.io.File f) {
      String name = f.getName().toLowerCase();
      return f.isDirectory() ||   name.endsWith(".png") || name.endsWith(".bmp");
      }
    public String getDescription() {
      return "Image (*.png, *.bmp)";
      }
    };
  fc.setAcceptAllFileFilterUsed(false);
  fc.addChoosableFileFilter(ff);

  java.io.File f = null;
  if(open && fc.showOpenDialog(this) == fc.APPROVE_OPTION)
    f = fc.getSelectedFile();
  else if(!open && fc.showSaveDialog(this) == fc.APPROVE_OPTION)
    f = fc.getSelectedFile();
  return f;
```

```
  }

private void openImage() {
  java.io.File f = showFileDialog(true);
  try {
    image = ImageIO.read(f);
    JLabel l = new JLabel(new ImageIcon(image));
    imagePane.getViewport().add(l);
    this.validate();
    } catch(Exception ex) { ex.printStackTrace(); }
  }

private void MessageExtract() {

  int len = extractInteger(image, 0, 0);

  byte b[] = new byte[len];

  for(int i=0; i<len; i++){
        b[i] = extractByte(image, i*16+32, 0);
        System.out.println(b[i]);
  }

  message.setText(new String(b));
  }

private int extractInteger(BufferedImage img, int start, int storageBit) {

  int maxX = img.getWidth(), maxY = img.getHeight(),
    startX = start/maxY, startY = start - startX*maxY, count=0;
```

```
  int length = 0;

   for(int i=startX; i<maxX && count<32; i++) {
     for(int j=startY; j<maxY && count<32; j++) {
       int rgb = img.getRGB(i, j), bit = getBitValue(rgb, storageBit);
       length = setBitValue(length, count, bit);
       count++;
       }
     }
   return length;
   }
int even = 0;
 private byte extractByte(BufferedImage img, int start, int storageBit) {

   int maxX = img.getWidth(), maxY = img.getHeight(),
     startX = start/maxY, startY = start - startX*maxY, count=0;

   byte b = 0;

   for(int i=startX; i<maxX && count<8; i++) {
     for(int j=startY; j<maxY && count<8; j++) {

         if(even%2 == 0)
         {
                 int rgb = img.getRGB(i, j), bit = getBitValue(rgb, storageBit);
                 b = (byte)setBitValue(b, count, bit);
                  System.out.println("Binary "+b);
                  count++;
                  even = 1;
         } else {
                 even = 0;
```

```
          }
          even = 0;

        }
      }

   int bb = b;

   if(bb%2 == 0)
       bb-=2;
    else
       bb+=2;

   return (byte)bb;
   }
private void resetInterface() {
   message.setText("");
   imagePane.getViewport().removeAll();
   image = null;
   this.validate();
}

private int getBitValue(int n, int location) {   // Get Bit
   int v = n & (int) Math.round(Math.pow(2, location));

   System.out.println("n "+n+" Location "+location+" V "+v);
   return v==0?0:1;
   }

private int setBitValue(int n, int location, int bit) {  // Set Bit
   int toggle = (int) Math.pow(2, location), bv = getBitValue(n, location);
```

```
    if(bv == bit)
      return n;
    if(bv == 0 && bit == 1)
     n |= toggle;
    else if(bv == 1 && bit == 0)
      n ^= toggle;
    return n;

    }
}
```

Chapter - 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion and Future Work

The proposed method is combination of one of the popular method of stagnography which is LSB and beauty of cryptography APIs of Java. Now the latest versions of java are using base 64 methods for encryption and decryption. In the proposed tool we have used least significant bit of image for hiding the message after encoding it in some encrypted format.

The proposed system is using Java1.8 for coding. The user interface is designed in Swing. Java has provided very efficient APIs for image handing. The tool is a good demonstration of LSB method and encoding scheme which can later be used in several applications.

A lots of researchers have worked on concept of LSB. Some of them have designed good algorithm. But a very least number of researchers have taken it to implementation level. Our focus of the work was to implement one of the best models, in which the data can be moved in secure and safe way. We tried to study various language specifications. Java has implemented one of the best security model so we used java security for designing the tool.

For security of encrypted data, a lots of complex algorithms were suggested by researchers. Most of these algorithms were very effective but these algorithms are a bit complex in understanding and implementation. So we used our own algorithm for storing length of message, for encoding message location etc.

Overall the proposed tool is very nice, compact and effective tool for implementation and understanding of LSB method of stegnography. Also the tool is developed by using complete object oriented methodology which can later be extended as per need.

The work of stagnography and encryption can never ends. This is the race condition in

between hackers and these methods. In the present tool more image processing like resizing, more brightness etc can be added. Although java is using Base64 method which is one of the best methods as on date but still it is not destination. As and when new methodologies are evaluated, it is essential to update the tool as per requirement

Our ability to discover hidden information during our investigations is vital, especially as new and innovative methods continue to evolve. During the past decade, data hiding technologies have advanced from limited use to ubiquitous deployment. With the rapid advancement of smart mobile devices, the need to protect valuable proprietary information has generated a plethora of new methods and technologies for both good and evil. Most dangerous among these are those that employ hiding methods along with cryptography, thus providing a way to both conceal the existence of hidden information while strongly protecting the information even if the channel is discovered.

Many vendors provide excellent technologies for protecting the privacy of information for the desktop. In addition, many of the latest smart mobile platforms (Android and iPhone) include built-in cryptographic capabilities. What is more dangerous and difficult to discover/decipher are data hiding methods that exploit multimedia and protocol weaknesses to both hide and communicate covertly. These new techniques provide hybrid solutions that combine the best of cryptography with the best of steganography. The interest, innovation, and advancement of these threats continue to go unchecked for the most part.

The present study can be extended for the use of different mobile technologies like window mobile, android, iphone etc. Also the study can be extended for audio and vedio type of media. In the present system, we are using encryption methods which are given by java only. Later on, the tool and such applications can be developed in almost all the available technologies. As java is open source and source code of encryption/decryption methods are available, these methods/classes can be re-written to extend their algorithm and our new ideas can be included in these methods.

REFERENCES

[1] CHIN-CHEN CHANG, , H.W .TSENG. ,"A steganographic method for digital images using side match. Pattern Recognition Letters, 2004,vol. 25, p.1431-1437.

[2] Vijay kumarsharma, Vishal Shrivastava, "A Steganog-raphy algorithm for hiding image in image by improved LSB substitution by minimize technique", Journal of Theoretical and Applied Information Technology, Vol. 36 No.1, 15th February 2012.

[3] Mehdi Kharrazi, Husrev T. Sencar, and NasirMemon,, Image Steganography and: Concepts and Practice", Depart-ment of Electrical and Computer Engineering Department of Computer and Information Science Polytechnic Universi-ty,Brooklyn, NY 11201, USA.

[4] R. Amirtharajan, R. Akila, P. Deepikachowdavarapu "A Comparative Analysis of Image Steganogra-phy",International Journal of computer Applications,Vol2- No3, May 2010.

[5] SaeedMahmoudpour, SattarMirzakuchaki,"Hardware Architecture for a Message Hiding Algorithm with Novel Randomizers", International Journal of Computer Applica-tions (0975 – 8887) Volume 37– No.7, January 2012.

[6] Mrs. Kavitha, KavitaKadam, AshwiniKoshti, PriyaDunghav, "Steganography Using Least Significant Bit Algo-rithm", International Journal of Engineering Research and applications, vol.2, issue 3, pp. 338-341May-June2012.

[7] BassamJamilMohd, Saed Abed and Thaier Al- Hayajneh, Computer Engineering Department Hashemite University, Zarqa, Jordan Sahel Alouneh,ComputerEngi-neering Department, German-Jordan University, Amman, Jordan, "FPGA Hardware of the LSB Steganography Meth-od" IEEE 2012.

[8] Atallah M. Al-Shatnawi, "A New Method in Image ste-ganography with improved image quality", Applied mathe-matical science, Vol. 6, no79, 2012.

[9] Nagham Hamid, AbidYahya, R. Badlishah Ahmad, Osamah M, "Image Steganography Techniques: An Over-view", International Journal of computer science and securi-ty, vol (6), Issue (3), 2012.

[10] Aneesh Jain, IndranilSen Gupta, ―A JPEG Compression Resistant Steganography Scheme for Raster Graphics Images‖, TENCON 2007 - 2007 IEEE Region 10 Conference, vol.2

[11] Jessica Fridrich, MiroslavGoljan, and Rui Du, ―Detecting LSB Steganography in Color and Gray-Scale Images‖, Magazine of IEEE Multimedia, Special Issue on Multimedia and Security, pp.22-28, October-December 2001.

[12] MohesenAshourian, R.C. Jain and Yo-Sung Ho, "Dithered Quantization for Image Data Hiding in the DCT domain", in proceeding of IST2003, pp.171-175, 16-18 August, 2003 Isfahan Iran.

[13]Y. R.PARK, H.H.KANG, S.U.SHIN, K.R.KWON,"A steganographic scheme in digital images using information of neighboring pixels. In Proc. International

[14] MamtaJuneja, Parvinder S. Sandhu, and EktaWalia,"Application of LSB Based Steganographic Technique for 8-bit Color Images" World Academy of Science Engineering, and Technology 50 2009.

[15] R. Chandramouli and N. Memon, "Analysis of lsb based image steganography techniques," in ImageProcessing, 2001.Proceedings. 2001 International Conference on, vol. 3, pp. 1019–1022, IEEE,2001.

[16] V. Lokeswara Reddy, Dr.A.Subramanyam, Dr.P. Chenna Reddy, "Implementation of LSB Steganography and its Evaluation for Various File Formats", Int. J. Advanced Networking and Applications 868 Volume: 02, Issue: 05, Pages: 868-872 (2011).

[17] A. Daneshkhah, H. Aghaeinia, and S. H. Seyedi, “A more secure steganography method in spatial domain,” in Intelligent Systems, Modelling and Simulation (ISMS), 2011 Second International Conference on, pp. 189–194, IEEE, 2011.

[18] Petitcolas, Fabien A.P.; Katzenbeisser, Stefan (2000). Information Hiding Techniques for Steganography and Digital Watermarking.Artech House Publishers.ISBN 1-580-53035-4.

[19] Johnson, Neil; Duric, Zoran; Jajodia, Sushil (2001). Information hiding: steganography and watermarking: attacks and countermeasures. Springer.ISBN 978-0-792-37204-2.

[20] N.F. Johnson and S. Jajodia, Exploring steganography: Seeing the unseen, IEEE Computer,31(2) (1998) 26 - 34.

[21] J.C. Judge, Steganography: Past, present, future. SANS Institute publication, http://www.sans.org/ reading_room/whitepapers/stenganography/552.php, 2001.

[22] N. Provos and P. Honeyman, Hide and seek: An introduction to steganography, IEEE Security and Privacy, 01 (3) (2003) 32-44.

[23] P. Moulin and R. Koetter, Data-hiding codes, Proceedings of the IEEE, 93 (12) (2005) 2083 - 2126.

[24] S.B. Sadkhan, Cryptography: Current status and future trends, in: roceedings of IEEE International Conference on Information & Communication Technologies: From Theory to Applications, Damascus. Syria, April 19-23,2004, pp. 417-418.

[25] G.J. Simmons, The prisoners’ problem and the subliminal channel, in: Proceedings of International conference on Advances in Cryptology, RYPTO8 3, August 22-24, 1984, pp. 51-67.

[26] C. Kurak and J. McHugh, A cautionary note on image downgrading, in: roceedings of the IEEE 8th Annual Computer Security Applications Conference, 30 Nov-4 Dec 1992, pp. 153-159.

[27] cf. section 1-133, "Color/Graphics Adapter",page 143 of ibm_techref_v202_1.pdf

[28] R.J Andersen and F.A.P Petitcolas.On the limits of steganography. IEEE Journal of Selected Areas in Communications,Special Issue on Copyright and Privacy Protection, 16(4):474–481, 1998.

[29] Jiri Fridrich. A new steganographic method for palettebased images. In Proceedings of the IS&T PICS conference, pages 285–289, Savannah, Georgia, April 1998.

[30] Jiri Fridrich and Du Rui. Secure steganographic methods for palette images. In Inter'l Workshop on Information Hiding, pages 47–60, 1999.

[31] N.F. Johnson and S. Jajodia. Exploring steganography: seeing the unseen. In IEEE Comput., pages 26–34, February 1998.

[32] K.B.Raja, C.R.Chowdary, Venugopal K R, and L.M.Patnaik," A Secure Image Steganography using LSB, DCT and Compression Techniques on Raw Images" Department of Computer Science Engineering, Bangalore 2005 IEEE

[33] N.F. Johnson and S. Jajodia. Steganalysis of images created using current steganography software. In Proc. the Second Inform. Hiding Workshop LNCS, volume 1525, pages 273–289.Springer-Verlag, 1998.

[34] D. Kahn. The history of steganography. In R. Anderson, editor, 1st Information Hiding Workshop, Lecture Notes in Computer Science, volume 1174, pages 1–5. Springer -Verlag, 1996.

[35] Me hdi Kharrazi, Husrev T. Sencar, and NasirMemon. Image steganography: Concepts and practice. 2004.

[36] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In Advances in Cryptology: Proceedings of Crypto 83 (David Chaum, ed.), pages 51 – 67. Plenum Press,1984.

[37] J. Fridrich and M. Long, "Steganalysis of lsb encoding in color images," in Multimedia and Expo,2000. ICME 2000.2000 IEEE International Conference on, vol. 3, pp. 1279–1282, IEEE, 2000.

[38] Fridrich, J., Goljan, M. and Du, R.: Reliable detection of LSB steganography in color and grayscale images. Proc. ACM Workshop on Multimedia and Security, Ottawa, ON, Canada, Oct. 5, 2001, pp. 27-30.

[39] Sharp, T.: An implementation of key-based digital signal steganography. Proc. 4th International Workshop on Information Hiding.Springer LNCS, vol. 2137, pp.13-26, 2001.

[40] Kawaguchi, E. and Eason, R.: Principle and applications of BPCS-Steganography. Proc. Multimedia Systems and Applications Conference, Boston, MA, USA, November 2, 1998. SPIE series, vol. 3528, pp. 464-473.

[41] Moskowitz, I., Longdon G. and Chang, L.: A New Paradigm Hidden in Steganography. Proc. 2000 Workshop on new security paradigms, Ballycotton, Country Cork, Ireland, 2000. ACM Press, New York, pp. 41-50.

[42] Kurak, C. and McHugh, J.: A Cautionary Note on Image Downgrading. Proc. IEEE 8th Annual Computer Security Applications Conference. San Antonio, USA, Nov./Dec. 1992, pp. 153-155.

[43] ZHANG, J., COX, I. J., DOERR, G. Steganalysis for LSB matching in images with high-frequency noise. In Proc. IEEE Ninth Workshop on Multimedia Signal Processing. Chania (Greece), 2007, p. 385-388.

[44] KER, A. D. Steganalysis of LSB matching in grayscaleimages.IEEE Signal Processing Letters, 2005, vol. 12, no. 6, p. 441-444.

[45] V. Lokeswara Reddy, Dr.A.Subramanyam, Dr.P. Chenna Reddy, "Implementation of LSB Steganography and its Evaluation for Various File Formats", Int. J. Advanced Networking and Applications 868 Volume: 02, Issue: 05, Pages: 868-872 (2011).

[46] Moskowitz, I., Longdon G. and Chang, L.: A New Paradigm Hidden in Steganography. Proc. 2000 Workshop on new security paradigms, Ballycotton, Country Cork, Ireland, 2000. ACM Press, New York, pp. 41-50.

[47] Sharp, T.: An implementation of key-based digital signal steganography. Proc. 4th International Workshop on Information Hiding, Pittsburgh, USA, April 25, 2001. Springer LNCS, vol. 2137, pp. 13-26.

[48] Kawaguchi, E. and Eason, R.: Principle and applications of BPCS-Steganography. Proc. Multimedia Systems and Applications Conference, Boston, MA, USA, November 2, 1998. SPIE series, vol. 3528, pp. 464-473.

[49] S. Venkatraman, A. Abraham, M. Paprzycki, "Significance of Steganography on Data Security", International Conference on Information Technology: Coding and Computing (ITCC'04), Las Vegas, 5-7 April 2004.

[50] CHI-KWONG CHAN, L. M. CHENG," Hiding data in images by simple LSB substitution", Pattern Recognition, 2004, vol. 37, p.469-474.

[51] XIAOLONG LI, BIN YANG, DAOFANG CHENG, TIEYONG ZENG ,"A generalization of LSB matching", IEEE Signal Processing Letters,2009, vol. 16, no. 2, p. 69-72.

[52] WEN-NUNG LIE, LI-CHUN CHANG,“ Data hiding in images with adaptive numbers of least significant bits based on the human visual system”, In Proc. IEEE Int. Conf. Image Processing. Kobe (Japan), October 24-28, 1999, p. 286-290.

[53] Y. K. LEE, L. H. CHEN,” High capacity image steganographicmodel”,IEE Proc., Vis. Image Signal Process, 2000, vol. 147, no. 3, p. 288-294.

[54] SHAO-HUI LIU, TIAN-HANG CHEN, HONG-XUN YAO,WENGAO,” A variable depth LSB data hiding technique in images”, In Proc. 2004 International Conference on Machine Learning and Cybernetics. Shanghai (China), Aug. 26-29, 2004, vol. 7, p. 3990-3994.

[55] http://docs.oracle.com/

[56] http://docstore.mik.ua/orelly/java-ent/jfc

[57] http://cs.wellesley.edu/~ecom/lecture/swing.html

[58] http://www.cs.ait.ac.th/

[59] http://docstore.mik.ua/orelly/java-ent/jnut/ch26_01.htm