Higher Institute of Computer Science and Information Systems
6th of October

IMAGE Steganography

Graduation Project
2019

DATA HIDING

Supervisors

Dr. Noha Ramadan

Eng. Aya Adel

ACKNOWLEDGEMENT

We would like to express our deepest appreciation to all those provided us the possibility to complete this project. A special gratitude we give to our Final year graduation project supervisor Dr. Noha Ramadan and Eng. Aya Adel, whose contribution in simulating suggestions and encouragement, helped us to coordinate our project especially in writing this report.

Last but not lease, many thanks to Dr. Sami Al - Daleel, the head Higher Institute of Computer Science and Information System and all other supervisors and who supported us completing this project, improve our presentations and execute this report professionally.

ABSTRACT

Steganography is the art of hiding the fact that communication is taking place, by hiding information in other information. Many different carriers le formats can be used, but digital images are the most popular because of their frequency on the Internet. For hiding secret information in images, there exists a large variety of steganography techniques some are more complex than others and all of them have respective strong and weak points. Different applications have different requirements of the steganography technique used.

For example, some applications may require absolute invisibility of the secret information, while others require a larger secret message to be hidden. This project hides the message with in the image. For a more secure approach, the project it allows user to choose the bits for replacement instead of LSB replacement from the image. sender select the cover image with the secret text or text le and hide it in to the image with the bit replacement choice, it helps to generate the secure stego image. the stego image is sent to the destination with the help of private or public communication network. on the other side i.e. receiver. receiver download the stego image and using the software retrieve the secret text hidden in the stego image.

Table of Contents

Chapter 01: Introduction 5
1.1. Introduction 6
1.2. Image Steganography 7
1.3. Image Files 7
1.4. General Concepts 8
1.5. Problem Background 8
1.6. Problem Statement 9
1.7. Objectives and Achievement 10
Chapter 02: HISTORY OF STEGANOGRAPHY 11
2.1. What is Steganography? 12
2.2. Importance of Steganography 16
Chapter 03: Technical History 17
3.1. Image Steganography Techniques 18
3.1.1. Spatial Domain 19
3.1.2. Transform Domain 22
3.2. Steganography Tools 24
3.3. F5 Technique Implementation [21] 33
3.3.1. JEPG Image 33
3.3.2. JPEG Image Compression 33
3.3.3. Discrete Cosine Transform (DCT) 35
3.3.4. Quantization 37
3.3.5. Zig Zag ordering 39
3.3.6. Huffman Coding 40
3.3.7. Entropy coding 40
3.3.8. Baseline Decoding 41
3.3.9. PNSR Block Computes 41
3.3.10. F5 Technique Execution by Java 43
Chapter 04: Results 44
4.1. Application's User interface for LSB Techniques 45
4.1.1. LSB-Steganography Method Steps: 46

4.2. Application's User interface for F5 Techniques 51
4.3. Usage Guide 53
4.3.1. Embedding text to image 53
4.3.2. Extracting text from image 53
4.3.3. Calculating PSNR 53
4.4. Source Code F5 and LSB on GitHub 53
4.5. Some Coding of program F5 55
4.5.1. Embedded Image with Text 55
4.5.2. Extract Text of Image 56
4.5.3. PSNR: 59
4.5.4. Histogram by MATLAB For F5 Techniques: 60
Chapter 05: Conclusion and Future Work 62
5.1. Conclusion 63
5.2. The Future of Image Steganography 64
References 65
Appendix: 67

CH01

Introduction

1.1. Introduction

Steganography is the idea of hiding private or sensitive data or information within something that appears to be nothing out of the normal. Steganography and cryptology are similar in the way that they both are used to protect important information. The difference between the two is that Steganography involves hiding information so it appears that no information is hidden at all. Nowadays the term "Information Hiding" relates to both watermarking and steganography. Watermarking is the technique use to hides information in a digital object (video, audio or image) so that information is robust to adjustments or alterations. By watermarking, the mark itself is invisible or unnoticeable for the human vision system. In addition, it should be impossible to remove a watermark without degrading the quality of the data of the digital object. The important application of watermarking is to copyright protection systems, which are intended to prevent unauthorized copying of digital media (pirating). For example, if the digital signal (audio, pictures or video) is copied, then the information is also carried in the copy. On the other hand, the main goal of steganography is to hide secret information in the other cover media (video, audio or image) so that other persons will not notice the presence of the information. This is a major distinction between this method and the other methods of covert exchange of information because, for example, in cryptography, the individuals notice the information by seeing the coded information but they will not be able to comprehend the information. However, in steganography, the existence of the information in the sources will not be noticed at all. Although steganography is separate and different from cryptography, but they are related in the way that they both are used to protect valuable information. From here emerged the urgent need to find new techniques alternative organization to overcome these weaknesses, giving rise to conceal information technology (Information Hiding), which are based on a different principle to the idea of organization, where they are buried information (Information Embedding) within another media carrier, and

making them aware (Imperceptible) by hackers and attackers, and so are the public domain of information to users of the network, while the content monopoly "on the relevant agencies, which alone knows how to extract content.

1.2. Image Steganography

As stated previously, images are considered as the most popular file formats used in steganography. They are known for constituting a non-causal medium, due to the possibility to access any pixel of the image at random. In addition, the hidden information could remain invisible to the eye. However, the image steganography techniques will exploit "holes" in the Human Visual System (HVS).

1.3. Image Files

An image is defined as an arrangement of numbers and such numbers usually stand for different light intensities in different parts of the image. The numeric description takes the form of a lattice where the individual points given the name 'pixels'. Pixels are displayed horizontally, row by row. In a color scheme, the number of bits is known as the bit depth and this basically refers to the number of bits assigned to each pixel. Moreover, the smallest bit depth in the color scheme is 8, i.e., 8 bits are utilized to represent the color of each pixel. Both Monochrome and gray scale images usually utilize 8 bits for each pixel and such bits are capable of displaying up to 256 different colors or shades of gray. One more point to add is that digital color images are known for being saved in 24-bit files and for utilizing the RGB color model. Almost all the color variations for the pixels of a 24-bit image are derived from three basic color terms: red, green, and blue, and each of these colors is represented by 8 bits. Thus, in any given pixel, the number of different shades of red, green, and blue can reach 256 that adding up to more than 16 million combinations that finally result in more than 16 million colors. The most prominent image formats, exclusively on the internet, are the graphics interchange format (GIF), joint photographic experts'

group (JPEG) format, and to a lesser degree, the portable network graphics (PNG) format. The important issue to touch here is that most of the steganographic techniques attempt to exploit the structure of these formats. However, some literary contributions use the bitmap format (BMP) simply because of its simple and uncomplicated data structure.

1.4. General Concepts

Lossless compression

is known for being preferable when the original data should stay in its entirety. In this manner, the original image information will never be removed, and this makes it possible the reconstruction of the original data from the compressed data. This is typical of images in GIF and BMP.

Lossy compression

saves storage space by discarding the points the human eyes find difficult to identify. In this case the resulting image is expected to be something similar to the original image, but not the same as the original. JPEG compression uses this technique. A cover image is the image designated to carry the embedded bits or secret information.

- A stego image refers to the image carrying the hidden message.
- A stegokey is secret information necessary to get the hidden message from the stego image.

1.5. Problem Background

Depend feature is that the concealment of all samples covers the same percentage, and can thus determine the percentage of concealment soon to know the volume and type of cover technique used, as well as to this method achieve the highest rate at each hide application. In the field of Steganography, some terminology has

developed. The adjectives ' concealment ' defined at the information hiding workshop held in Cambridge, England. The term "concealment" refers to description of the original, innocent massage, data, audio, video.

1.6. Problem Statement

The problem in the hiding information or Steganography is the size of data that user want to embed inside the multimedia file, image is one of the multimedia file, the most commend method for hiding information in the image is LSB and f5, LSB is efficient instead of that it's not easy to analysis, however, it is not effective in term of the data hidden quantity, all researchers agreed the fact that the size of data hidden is a problem in that particular area, the other problem that faced there, in fact if we try to increase the quantity of data in the image there will be a suspect change which become clear to human eyes, for instance, this research will face a challenge that high rate data hidden without affecting the images quality, there are many trends that needs to be fallowed, initially; how can the new algorithm increase the amount of data, then what is the feature in the new image, how can the new algorithm deal with, all this items will be discuss in-depth in this research by suggest an enhancement to the work of hiding information in the image using the human vision system.
F5 is a steganography algo for hiding information in JPEG images. Unless other implementations it really hides it inside the image itself (not in metadata/comment fields or appended to the end of the file).

As a summery, the main problems in the Steganography fallow as:

1. The size of data hidden
2. Quality of image
3. Algorithms that apply should also cover the gray level image
4. Level of data protecting
5. The level of suspecting

1.7. Objectives and Achievement

The aims and scopes of researcher to complete this thesis can be summarized as follows:

1-Objectives:

1. To formulate an algorithm based on data hidden technique in the image to higher the quantity of hidden data without affecting the quality of images
2. To develop a system based on the algorithm established above.
3. To collect data and evaluate the accuracy of the algorithm by conducting a study on a survey for the images before and after hidden data
4. To act a new stego-analysis approach on the image to evaluate the new algorithm.

2-Research Scope

The scope of the study encompasses:

1. This research will focus on hidden data in image
2. Study the texture feature on image after increase the data hidden and its relation with the quality of image

CH02

HISTORY OF STEGANOGRAPHY

2.1. What is Steganography?

Abstract: Steganography is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity **[1].**

Steganographic technologies are a very important part of the future of Internet security and privacy on open systems such as the Internet. Steganography is a method of hiding a secret message inside of other data. Steganography, derived from Greek, literally means "covered writing." It includes a vast array of secret communications methods that conceal the message's very existence. It refers to the science of invisible communication. Unlike cryptography, where the goal is to secure communications from an eaves-dropper, Steganographic techniques strive to hide the very presence of the message itself from an observer. Steganography in the modern-day sense of the word usually refers to information or a file that has been concealed inside a digital Picture, Video or Audio file **[1].**

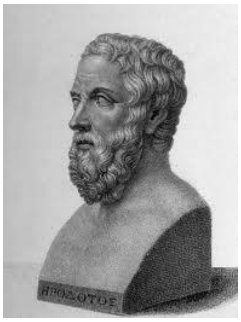


Fig. 2.1. Herodotus

(484 BC) was an ancient Greek historian who was born in Halicarnassus in the Persian Empire (Turkey). He is known for having written the book The Histories,

The first recorded uses of steganography can be traced back to 440 BC when Herodotus mentions two examples of steganography in The Histories of Herodotus. Demaratus sent a warning about a forthcoming attack to Greece by writing it directly on the wooden backing of a wax tablet before applying its beeswax surface. Wax

tablets were in common use then as reusable writing surfaces, sometimes used for shorthand **[2]**.

Another ancient example is that of Histiaeus, who shaved the head of his most trusted slave and tattooed a message on it. After his hair had grown the message was hidden. The purpose was to instigate a revolt against the Persians.

Steganography has been widely used, including in recent historical times and the present day. Possible permutations are endless and known examples include **[2]**:

1. Hidden messages within wax tablets in ancient Greece, people wrote messages on the wood, and then covered it with wax upon which an innocent covering message was written.
2. Hidden messages on messenger's body: also used in ancient Greece. Herodotus tells the story of a message tattooed on a slave's shaved head, hidden by the growth of his hair, and exposed by shaving his head again. The message allegedly carried a warning to Greece about Persian invasion plans. This method has obvious drawbacks such as delayed transmission while waiting for the slave's hair to grow, and its one-off use since additional messages requires additional slaves.

Fig. 2.2. Hidden messages

3. In WWII, the French Resistance sent some messages written on the backs of couriers using invisible ink.

Fig. 2.3. Hidden messages on messenger's body

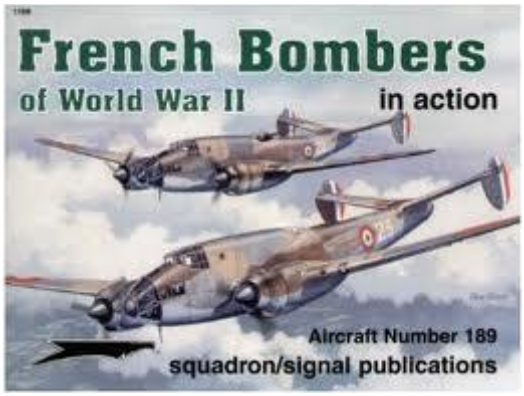


Fig. 3.4. messages written on the backs

4. Hidden messages on paper written in secret inks, under other messages or on the blank parts of other messages.
5. Messages written in Morse code on knitting yarn and then knitted into a piece of clothing worn by a courier.
6. Messages written on the back of postage stamps.

Fig. 3.5. Messages written on the back of postage stamps

7. During and after World War II, espionage agents used photographically produced microdots to send information back and forth. Microdots were typically minute, approximately less than the size of the period produced by a typewriter. WWII microdots needed to be embedded in the paper and covered with an adhesive (such as colliding). This was reflective and thus detectable by viewing against glancing light. Alternative techniques included inserting microdots into slits cut into the edge of post cards.
8. During World War II, a spy for Japan in New York City, Velvalee Dickinson, sent information to accommodation addresses in neutral South America. She was a dealer in dolls, and her letters discussed how many of this or that doll to ship. The stegotext was the doll orders, while the concealed "plaintext" was itself encoded and gave information about ship movements, etc. Her case became somewhat famous and she became known as the Doll Woman.
9. Cold War counter-propaganda. In 1968, crew members of the USS Pueblo (AGER-2) intelligence ship held as prisoners by North Korea, communicated in sign language during staged photo opportunities, informing the United States they were not defectors but rather were being held captive by the North Koreans. In other photos presented to the US, crew members gave "the finger"

to the unsuspecting North Koreans, in an attempt to discredit photos that showed them smiling and comfortable.

2.2. Importance of Steganography

Unlike information hiding and digital watermarking, the main goal of steganography is to communicate securely in a completely undetectable manner.

The majority of today's Steganographic systems use multimedia objects like image, audio, video etc. as cover media because people often transmit digital pictures over email and another Internet communication. In modern approach, depending on the nature of cover object, Steganography can be divided into five types:

1. Text Steganography
2. Image Steganography
3. Audio Steganography
4. Video Steganography
5. Protocol Steganography

So, in the modern age so many Steganographic techniques have been designed which works with the above concerned objects. More often in today's security advancement, we sometimes come across certain cases in which a combination of Cryptography and Steganography are used to achieve data privacy over secrecy.

CH03

Technical History

3.1. Image Steganography Techniques

Image steganography techniques can be divided into two groups: Image Domain – also known as Spatial Domain techniques embed messages in the intensity of the pixels directly. -, and Transform Domain – also known as Frequency Domain, images are first transformed and then the message is embedded in the image. Many carrier messages can be used in the recent technologies, such as Image, text, video and many others. The image file is the most popular used for this purpose because it easy to send during the communication between the sender and receiver. As show fig (3.1)

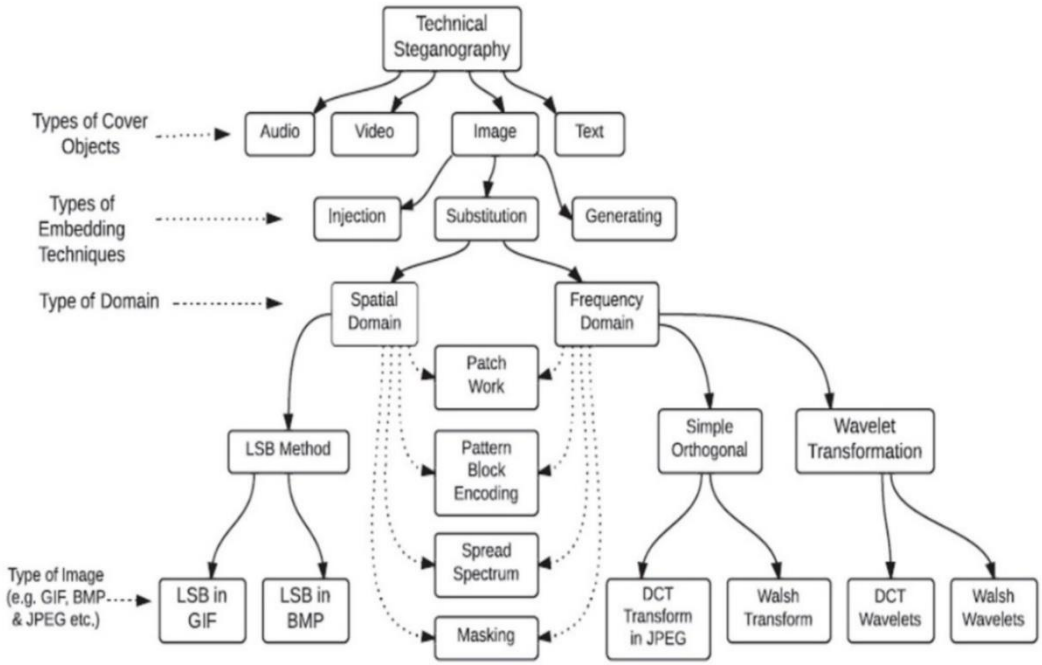


Fig. 3.1. Classification of various image steganography techniques

The images are divided into three types: Binary (Black-White), Gray-scale and Red-Green-Blue (RGB) images. The binary image has one-bit value per pixel represent by 0 for black and 1 for white pixels. While the gray scale image has 8-bits value per pixel represent from 00000000 for black and 11111111 for white pixels. The RGB image has 24-bits values per pixel represent by (00000000, 00000000 and 00000000) for black and (11111111, 1111111 and 11111111) for white pixels. The RGB image

is the most suitable because it contains a lot of information that help in hiding the secret information with a bit change in the image resolution which does not affect the image quality and make the message more secure **[3]**.

3.1.1. Spatial Domain

Least significant bit (LSB) methods

The Least Significant Bit (LSB) insertion method is a common, simple approach to embedding information in a graphical image file. In LSB insertion method the LSB of every pixel is replaced by every message bit. There is 50% chance that the message may match with the LSB's of the Cover image. Thus only 50% LSB's are likely to change. Also, the change occurs only in the bit which is least significant, thus keeping the other more significant bits unaltered. Therefore, this does not affect the original image perceptibility. Hence it is a very popular technique. However, it is extremely vulnerable to attacks. Any image manipulations such as cropping, intensity changes for any enhancements such as contrast stretching, histogram equalization, addition of noise… etc. will destroy the embedded message **[4]**.

The techniques other than LSB technique are complicated, although they are robust to most attacks. LSB technique can therefore be used wherever we want to store confidential information on a standalone PC or one which is shared among several users. LSB technique can be used to store personal data such as ATM, PIN, Credit card details, salary statement, income tax data, passport information... etc. in an imperceptible way. So, wherever this kind of information is to be preserved in a manner that only legitimate user should be able to retrieve it whenever needed, by simple ways, LSB is a better solution.

Least Significant Bit (LSB) encoding is the easiest of the techniques used for embedding secret or confidential information in digital images. For a gray scale bitmap (BMP), using the LSB of each byte (8 bits) in an image, a secret message

of size 1/8th of the Cover image can be stored. This can be easily done by directly substituting every bit of the secret message into every LSB.

For a 24-bit color image as the cover image, since there are 3 bytes for every pixel, 3 bits of data can be stored in each pixel, so the capacity to store increases by 3 times thus making it 3/8 of the cover image size. If the message to be embedded is a text message a secret message of size 1/7th of the grayscale cover image can be stored and in a 24-bit color image as cover a text message of size 3/7 can be embedded **[4]**.

The confidential information which is embedded in the Cover image can be an image (grayscale, binary or color image), text or even audio. As the type and size of confidential information varies, the embedding capacity varies for a particular type of Cover used. LSB technique can be used either by directly replacing the Cover LSB's by the Secret information bits, or to add one more level of security, it can be encrypted and then inserted into the LSB"s of the Cover.

The following Algorithm is the Concept of LSB in RGB Images:

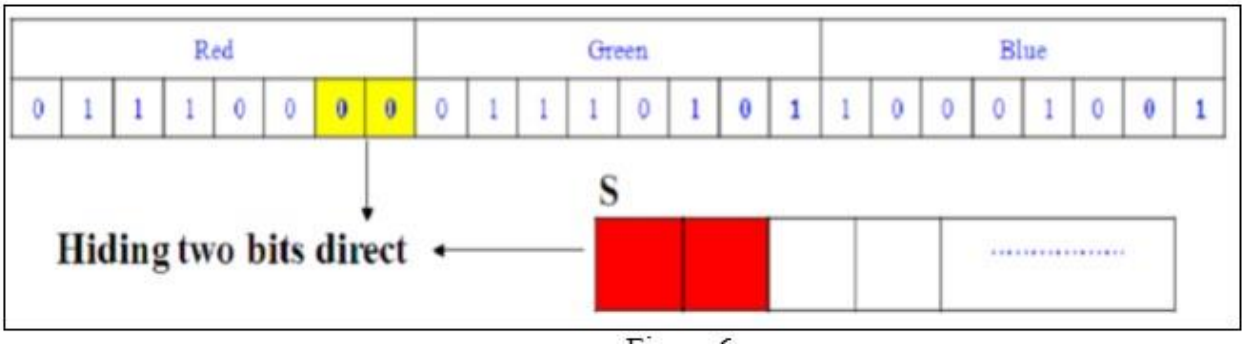


Fig. 3.2. (Algorithm): Least Significant Bit Hiding

Begin:
Scan the image row by row and encode it in binary. Encode the secret message in binary. check the size of the image and the size of the Secret message.
Start sub-iteration 1:
Divide the image into three parts (Red, Green and Blue parts) hide two by two bits of the secret message in each part of the pixel in the two least Significant bits.
End sub-iteration 1:
Set the image with the new values and save it.
End

The Proposed Hiding of Least Significant Bit Method:

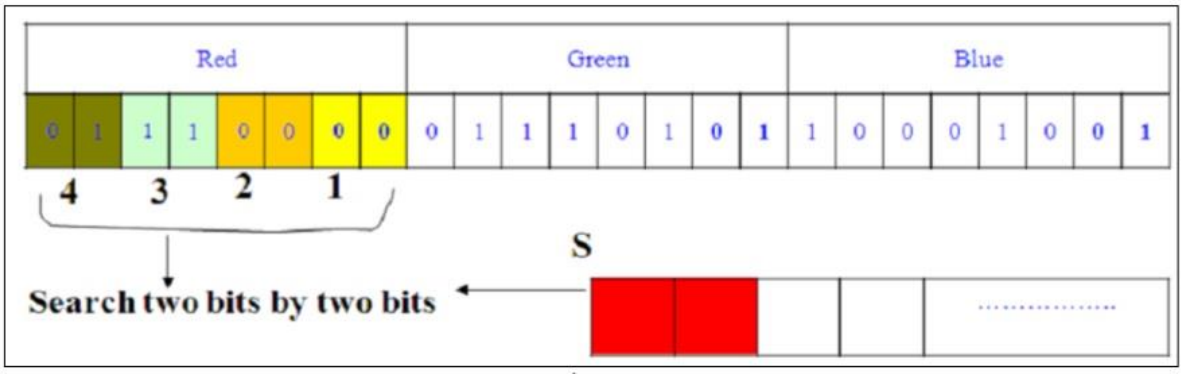


Fig. 3.3. (Algorithm): Proposed Hiding Method

```
Begin:
Scan the image row by row and encode it in binary. Encode
the secret message in binary. check the size of the image
and the size of the secret message.
Start sub-iteration 1:
Choose one pixel of the image randomly Divide the image
into three parts (Red, Green and Blue parts) hide two by
two bits of the secret message in each part of the pixel
by searching about the identical. If the identical is
satisfied then set the image with the new values.
otherwise hide in the two least significant bits and set
the image with the new values save the location of the
hiding bits in binary table.
End sub-iteration 1.
Set the image with the new values and save it.
End
```

3.1.2. Transform Domain

(also known as Frequency Domain, images are first transformed and then the message is embedded in the image).

JSteg:

The JSteg algorithm is acclaimed as the first commercially available steganographic tool for JPEG images. The algorithm applies Discrete Cosine Transform (DCT) to the image blocks and embeds the data in to LSBs of the DCT coefficients, sequentially. The sequential embedding and absence of any secret

key makes the algorithm susceptible to eavesdropping as only knowledge of the embedding procedure is sufficient to decode the hidden message. Moreover, JSteg is easily steg-analyzed using the χ2-attack. Also, as the algorithm uses the DCT, it is extremely necessary to treat the DCT coefficients with sensitive care and intelligence in order to prevent the algorithm from leaving significant statistical signatures **[5]**.

Outguess:

This algorithm was an improvement of the existing JSteg algorithm. The Outguess uses a PRNG (Pseudo Random Number Generator) to randomize the pixels in which the embedding is to be made. It also does not embed into DCT coefficients with values 0 and 1 as because they form a Pair of Value when their LSB changes and there are no ways of distinguishing between a zero DCT coefficient and a steganographic zero.

The algorithm, after embedding, modifies the unchanged DCT coefficients to preserve the histogram of the original image. Thus, OutGuess is immune to attacks like the visual attack, histogram attack and the χ2attack. The steganalyzing algorithm for OutGuess utilizes the fact that as Outguess uses LSB embedding of the DCT coefficients and that it makes random changes to the quantized coefficients, the spatial discontinuity at the border of each8×8block will increase.

Jsteg:

- Easily detected by statistical attacks.
- JPEG compression.
- Embedding function of Jsteg.

F4:

(improved embedding operation)

F5 Technique:

Benefits of F5:

- High steganographic capacity
- High efficiency via matrix encoding
- Prevents visual attacks
- Resistant to statistical attacks (chi square).
- Uses JPEG as carrier (common in e-mails).
- Source code publicly available.

F5 (advanced efficiency)

- Per mutative straddling
  - Matrix encoding

The F5 differs from most other steganographic algorithm in the fact that it **does not overwrite LSBs** of DCT coefficients/pixels rather it increments/decrement the value of the DC coefficients depending on need.

The algorithm takes into consideration that flipping the LSBs either at the pixel level or at the DC coefficient level alters the statistical properties of the image and can serve as a means to steg-analyze the algorithm.

F5 uses per mutative straddling and matrix encoding to scatter the embedding effect and to embed data, respectively.

3.2. Steganography Tools

There is many software available that offer steganography. Some offer normal steganography, but a few offer encryptions before hiding the data. Few can only hide

data behind image, but few can hide data behind any file. Look at these tools and see how they work. I am sure you will surely start using any of these tools for your daily communications in which you want to have some kind of security **[6] [7] [8].**

Xiao Steganography

Xiao Steganography is free software that can be used to hide secret files in BMP images or in WAV files. Use of the tool is easy. You can just open the software, load any BMP image or WAV file to its interface. Then add a file, which you want to hide. It also supports encryption. So, you can select from various algorithms like RC4, Triple DES, DES, Triple DES 112, RC2 and hashing SHA, MD4, MD2, MD5. Select any one from the lost and then save the target file.

To read the hidden message from this file, you will have to use this software again. This software will read the file and will decode the hidden file from it. You cannot extract the hidden file from any other software **[9]**.

Fig 3.4. xiao steganography

Image Steganography

Image Steganography is also a free software for hiding your information in image files. You can hide text message or files inside an image file. Just select the source file in which you want to hide the secret message, and then select the file to hide or write the text message to hide. Select the output image location and then click

on start button to start encoding the file. Encoded image will have the secret message inside the image. You can use the decode option of the same tool to decode the hidden file or message from the image **[10]**.

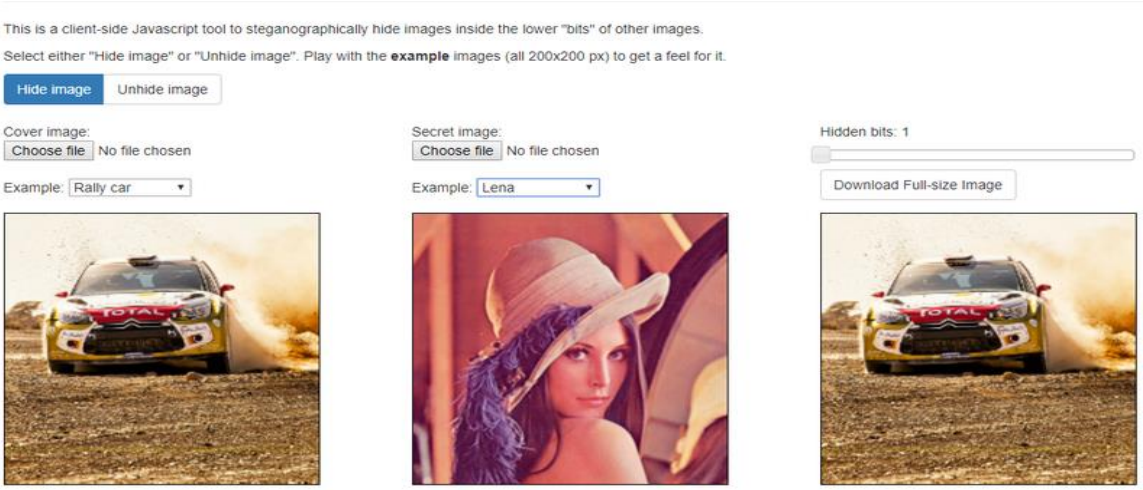


Fig 3.5. image steganography

Steghide

Steghide is an open source Steganography software that lets you hide your secret file in image or audio file. You will not notice any change in the image or audio file. However, your secret file will be inside the original image or audio file. It is command line software. Therefore, you need to learn the command to use the tool. Commands will be used to embed files in the image or audio file. In addition, to extract your file from image or audio file, you need to use other command **[11]**.

Fig. 3.6. Steghide

Crypture

Crypture is another command line tool that performs Steganography. You can use this tool to hide your sensitive data inside a BMP image file. But there is one requirement. BMP file should be eight times larger than the data file which you want to hide inside the BMP file. If you have a small amount of data to hide, you can use this tool. This tool is very small and is only 6KB in size. It does not need any kind of installation **[12]**.

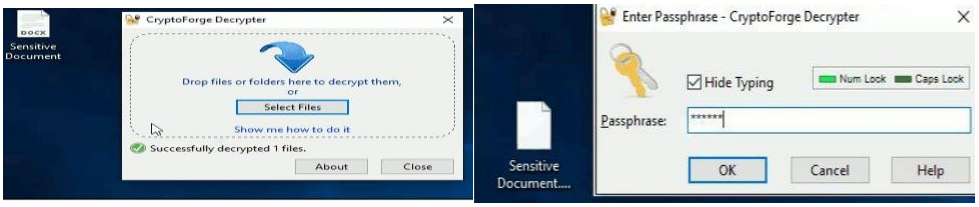


Fig. 3.7. Crypture

SteganographX Plus

SteganographX Plus is another small tool that lets you hide your confidential data inside a BMP image. It also does not need any installation and is of only 496 KB in size. it is simple and offers easy to use interface. You can use this tool to hide your sensitive data inside a BMP file and recover your sensitive data from that file **[13]**.

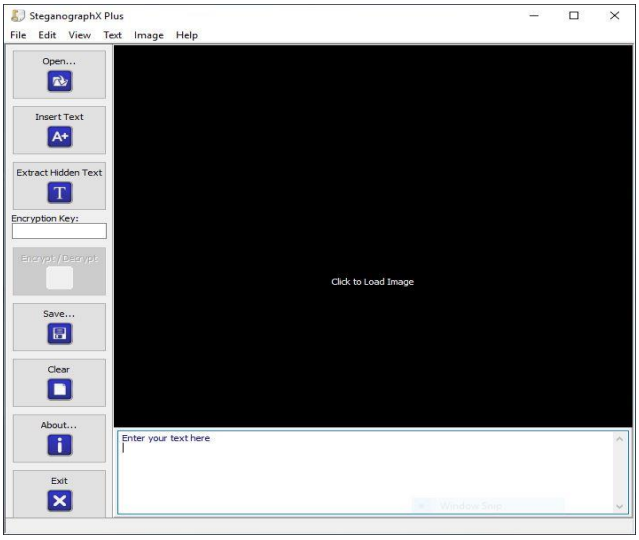


Fig. 3.8. SteganographX

rSteg

The rSteg is a Java based tool that lets you hide textual data inside an image. It has two buttons: one to encrypt and second to decrypt the text. Just select the image file, enter the PIN, and then enter the text which you want to hide in the image. It will generate a target image file with hidden text inside. If you want to read that text again, use this tool and select decrypt option **[14]**.

Fig. 3.9. rSteg

SSuite Picsel

SSuite Picsel is a free portable application to hide text inside image file. However, it has a different approach. It uses image file as a key to protect your hidden text inside an image. Don't be confused. Actually, this tool can hide text inside an image file. Nevertheless, to hide and unhide text inside an image, you need to enter another image as a key. To hide a text inside the image, select image in

which you want to hide the text and select another image for key. Now you can hide your text inside the first image. To unhide you text, you need to enter the key image **[15]**.

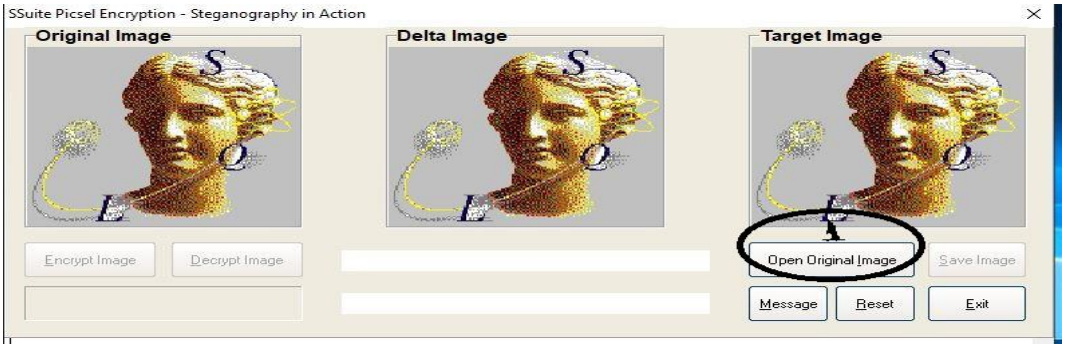


Fig. 3.10. SSuite Picsel

Our Secret

Our Secret is another similar kind of tool that is used to hide sensitive information in a file. Interface of the tool is divided into two parts. One part is to hide the data in a file and other part is to unhide. You need to select the carrier file in which you want to hide your data. Then select the data or file which you want to hide. Enter the password to encrypt your message and then hide the data in the file. Use the same tool again to unhide the data from the file you created with hidden data **[16]**.

Fig. 3.11. Our Secret

Camouflage

Camouflage is also a nice steganography tool that lets you hide any type of file inside of file. There is no kind of restriction in the software for hiding the file. Use of the tool is also simple and easy. You can just right click on any file and select the option of Camouflage. To extract your sensitive data from the file, right

click and select Uncamouflaged. You can also set password to encrypt the hidden data inside the file **[17]**.

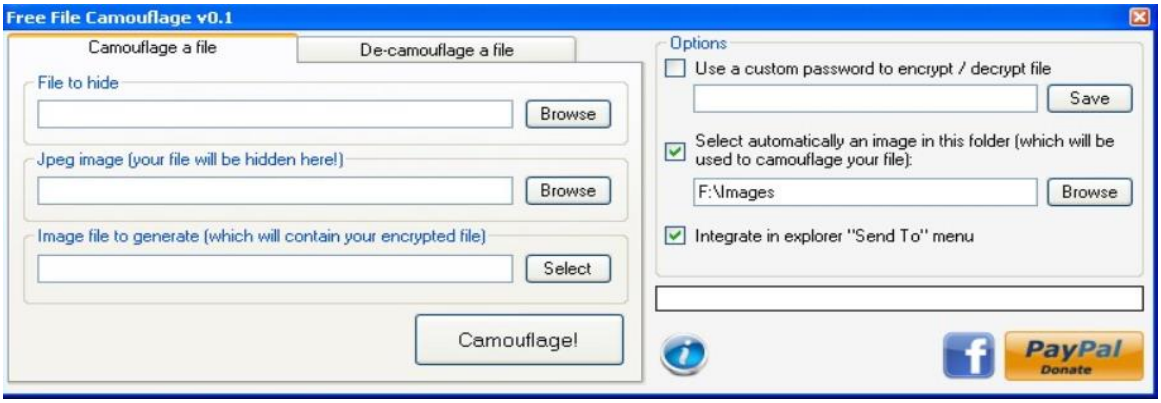


Fig. 3.12. Camouflage

OpenStego

OpenStego is another good option. You can attach any kind of secret message file in an image file. You can hide images in BMP, GIF, JPEG, JPG, PNG and WBMP. You can hide data in these files and take output as PNG file. The same software will be used to unhide data from the output file. It also uses password to encrypt your data along with hiding inside the image file. This tool is open source and developed on Java **[18]**.

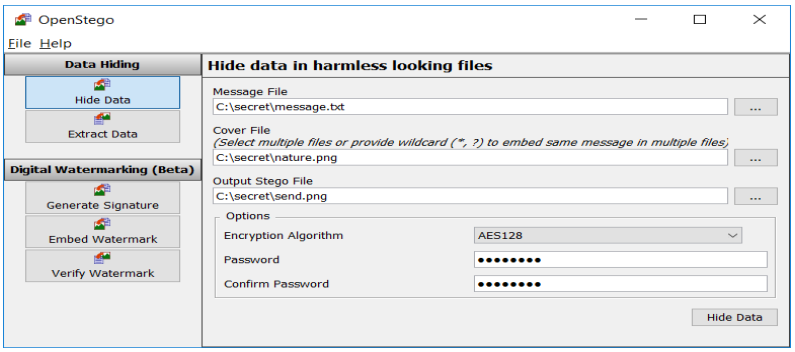


Fig. 3.13. OpenStego

SteganPEG

SteganPEG lets you hide any kind of file into a JPG image file. You can attach any file and give password to hide inside a JPG File. Only SteganPEG can extract your file from that output JPG image. Output file will act like ordinary image file

and one cannot tell that the image was modified just by looking into it. Interface of the software is simple with fewer options. So, one can easily use this tool to hide data into a JPG image file **[19]**.

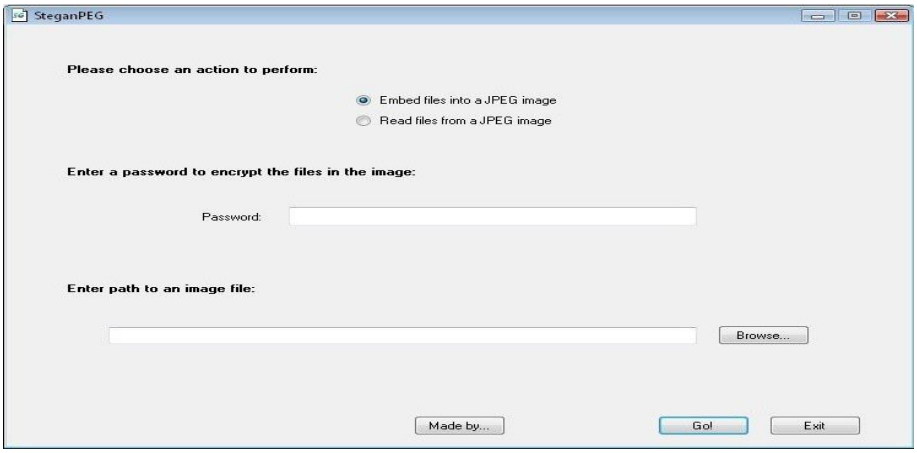


Fig. 3.14. SteganPEG

Hide'N'Send

Hide'N'Send is also a small utility which offers steganography feature. It lets you hide any kind of file behind a JPG image file. It supports hashing and encryption too. Therefore, you can hide your date by encrypting. This adds an extra layer of security. Interface of the tool is simple and offers two tabs –one to hide data and other to extract data. You can select the options accordingly. Just run the tool, select the image file, then select the file which you want to hide, select the encryption type and then hide the data the image. Use the same tool again to extract the hidden information in the image **[20]**.

Fig 3.15. hide'N'send

○ Compare Between Some Tools:

Types of Software Tools	License Model	Techniques of Encryption& Decryption	Capacity of Hiding size	Degree of Security
Xiao Steganography	Free Software	various algorithms like RC4, Triple DES, DES, Triple DES 112, RC2 and hashing SHA, MD4, MD2, MD5	limited by the size of image	Very Higher Secure
Image Steganography	Free Software	The data can be encrypted with AES	Capacity feature - displays the data capacity for an image, and shows whether the data can be encoded	Medium Secure
SSuite Picsel	Free Software	to encrypt any text document using groundbreaking steganography technique. Its also included is our latest security application called File shredder in new version		Higher Secure
Steghide	Open Source (Free)	encryption used is Rijndael in CBC mode with the 128-bit key, along with the CRC32 checksum.	128-bit key	Higher Secure

3.3. F5 Technique Implementation [21]

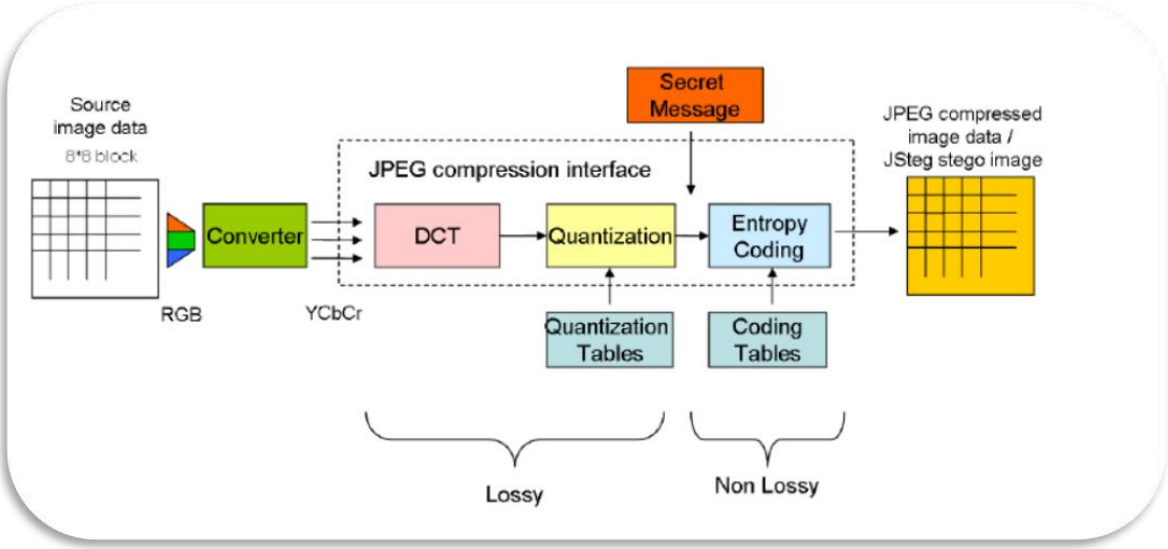


Fig. 3.16. F5 implementation

3.3.1. JEPG Image

A JPEG (stands for Joint Photographic Experts Group) is a compressed file format. It's a file format designed to balance image quality with file size and it's usually the right choice of format for pictures on the Web. A JPEG's file size can be a tenth or less of the file size of the original image. Advantages of JPEG are, it produces a small image size, perfect for most images and uses millions of colors. While the disadvantages are it leads to high compression loses quality and every time a JPEG is saved, it loses more and more pixel of the picture.

3.3.2. JPEG Image Compression

Lossless Image Compression

Lossless Image Compression is a class of image compression algorithms that allows the exact original image to be reconstructed from the compressed image.

Lossy Image Compression

Lossy encoding is based on the concept of compromising the accuracy of the reconstructed image in exchange for increased compression

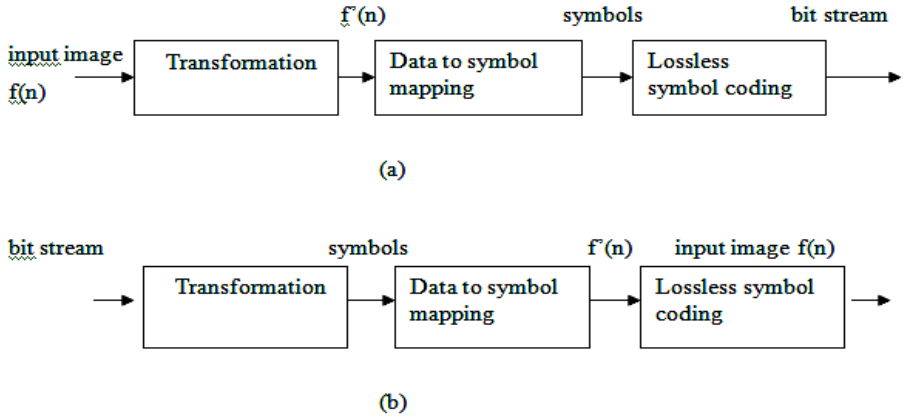


Fig. 3.17. General lossless coding system: (a) encoder (b) decoder.

the steganography technique that will be use is transformations. DCT (Direct cosine transformation) is one such method. DCT is used by the JPEG compression algorithm to transform successive 8 x 8-pixel blocks of the image, into 64 DCT coefficients each. Steganography tools can use the LSB of the quantized DCT coefficient can be used to hide information. In addition to DCT, images can be processed with fast Fourier transformation and wavelet transformation. This stage applies a reversible (one-to-one) transformation to the input image data. The purpose of this stage is to convert the input image data f(n) into a form f'(n) that can be compressed more efficiently. For this purpose, the selected transformation can aid in reducing the data correlation (interdependency, redundancy), alter the data statistical distribution and/or pack a large amount of information into few data samples or sub band regions. So, for this stage, discrete cosine transforms (DCT) technique is used.

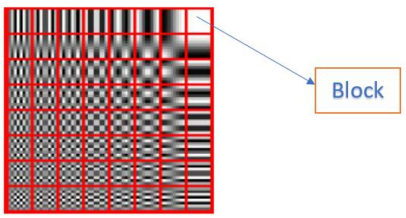


fig .3.18. block image

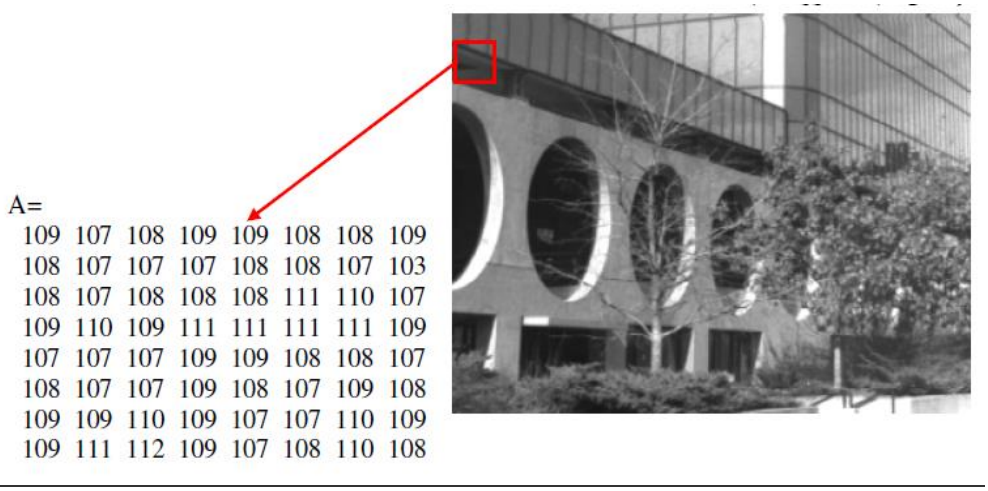


Fig. 3.19. pixel value in block image

3.3.3. Discrete Cosine Transform (DCT)

Discrete Cosine Transform (DCT) The discrete cosine transforms (DCT) is a technique for converting a signal into elementary frequency components. It is widely used in image compression. These functions illustrate the power of Mathematical in the prototyping of image processing algorithms. For this project, 2-D DCT is used. For analysis of two-dimensional (2D) signals such as images, we need a 2D version of the DCT.

helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality). The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain.

$$F(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \left(\frac{2}{M}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(i).\Lambda(j).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] cos\left[\frac{\pi.v}{2.M}(2j+1)\right].f(i,j)$$

The basic operation of the DCT is as follows:

- The input image is N by M;
- f(i,j) is the intensity of the pixel in row i and column j;
- F(u,v) is the DCT coefficient in row k1 and column k2 of the DCT matrix.

- For most images, much of the signal energy lies at low frequencies; these appear in the upper left corner of the DCT.
- Compression is achieved since the lower right values represent higher frequencies, and are often small - small enough to be neglected with little visible distortion.
- The DCT input is an 8 by 8 array of integers. This array contains each pixel's gray scale level;
- 8-bit pixels have levels from 0 to 255.

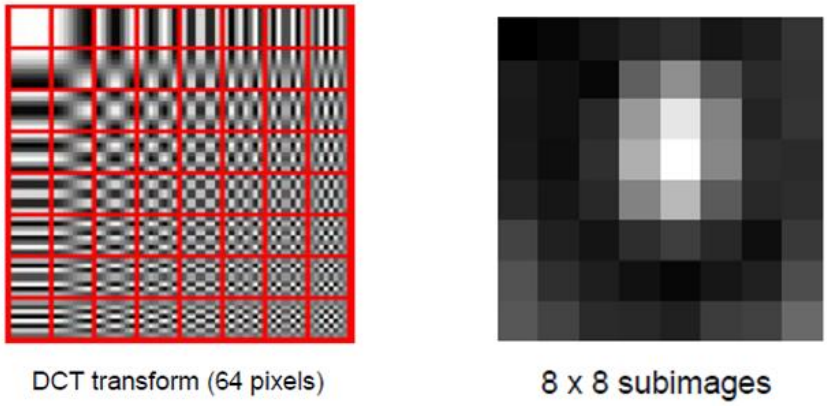


Fig. 3.20. Dct transform and sub images

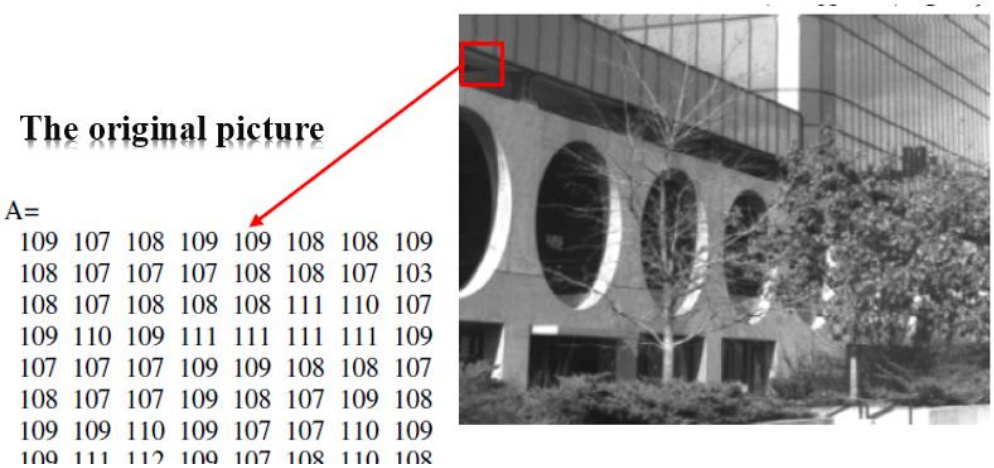


Fig. 3.21. the original image

DCT

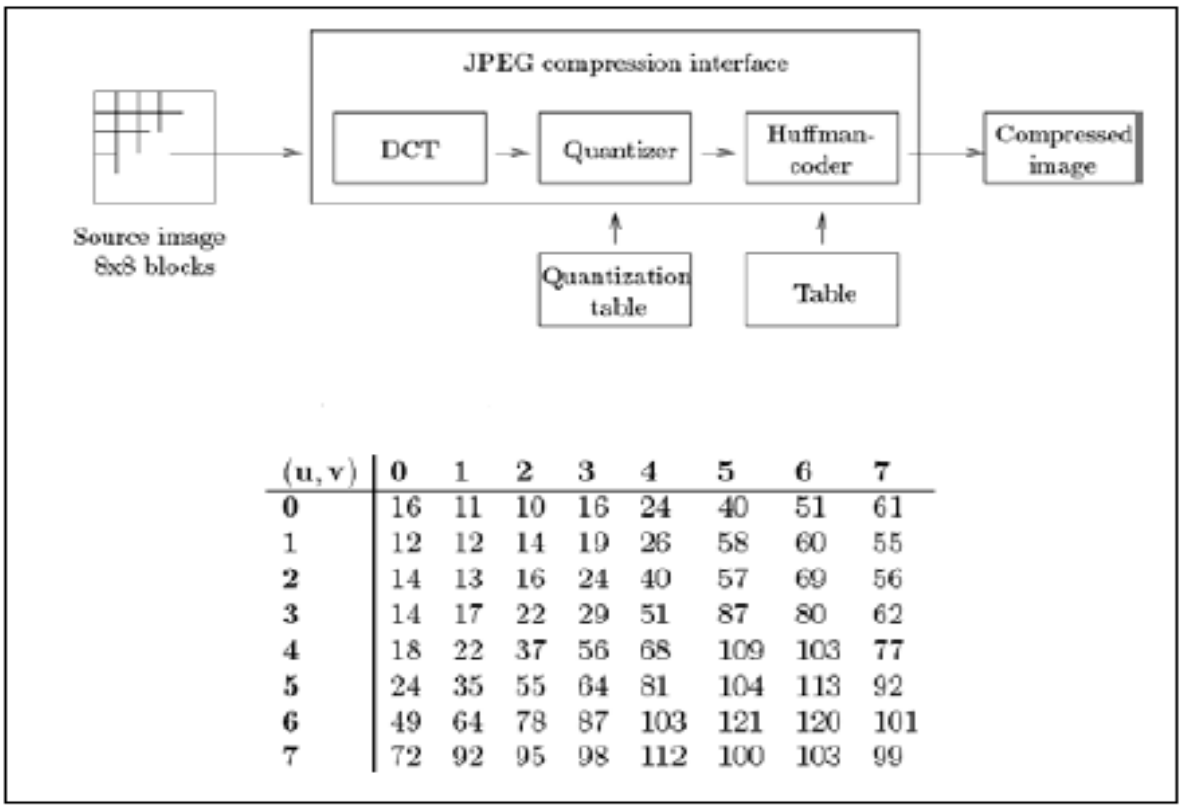
dct= 867.3750	0.1757	-1.6612	0.9119	-1.1250	2.6625	-1.0708	0.3336
-2.3703	-0.9451	-1.9990	3.7880	1.2178	-0.5544	0.9263	-0.2489
-0.6649	2.5583	1.8687	-1.9661	-0.8562	0.3054	0.9205	0.9395
-3.0711	-0.7383	0.2974	-0.7568	3.6270	-0.1458	-0.4965	0.2076
3.6250	-0.0825	-0.4900	-1.2196	0.6250	-1.1156	-0.2029	-0.2713
4.8439	-1.6469	1.4110	-1.2797	0.7333	0.1835	0.0751	-0.2812
0.6813	-1.6536	0.6705	-0.1762	0.2194	0.3228	0.6313	-0.3457
-1.4745	-0.8828	0.4350	-0.4027	-0.1092	0.2273	1.0001	0.0184

3.3.4. Quantization

Quantization is a process of reducing the number of possible values. Hence, it will reduce the number of bit used. For example, we have number ranging from 0 till 7, means that it has 8 real numbers. Rounding to the nearest integer it can be represented as 2.78452, thus it can be rounded to 3. So, it can be represented as 3 bits.

Image with higher spatial frequency is almost unnoticeable. Thus, the quantization factor needs to be large as possible as the higher frequency

$$F^{Q}(u,v) = IntegerRound\left(\frac{F(u,v)}{Q(u,v)}\right)$$



(u, v)	0	1	2	3	4	5	6	7
0	16	11	10	16	24	40	51	61
1	12	12	14	19	26	58	60	55
2	14	13	16	24	40	57	69	56
3	14	17	22	29	51	87	80	62
4	18	22	37	56	68	109	103	77
5	24	35	55	64	81	104	113	92
6	49	64	78	87	103	121	120	101
7	72	92	95	98	112	100	103	99

Fig. 3.22. JPEG process outline & Quantization values used in JPEG

1	2	4	8	16	32	64	128
2	4	4	8	16	32	64	128
4	4	8	16	32	64	128	128
8	8	16	32	64	128	128	256
16	16	32	64	128	128	256	256
32	32	64	128	128	256	256	256
64	64	128	128	256	256	256	256
128	128	128	256	256	256	256	256

High Pressure Compression Table

1	1	1	1	1	2	2	4
1	1	1	1	1	2	2	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

Low pressure quantization table

Fig. 3.23. high and low quantization table

```
quant=  867   0   0   0   0   0   0   0
         -1   0   0   0   0   0   0   0
          0   1   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
```

3.3.5. Zig Zag ordering

In **Zig Zag** ordering, DC coefficient is treated separately from 64 AC coefficient. DC coefficient is a measure of the average value of 64 image. The DC value is calculated using Differential Pulse Code Modulation (DPCM). The remaining AC coefficients are ordered into Zig Zag sequence, which help to facilitate entropy coding to set the low frequency coefficient before high frequency coefficient. The output of DPCM and Zig Zag ordering is then will be encoded using entropy coding separately.

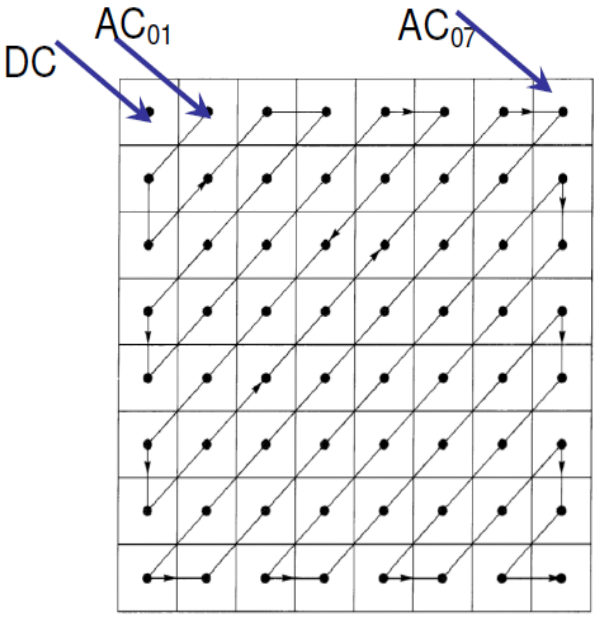


Fig. 3.24. DC and AC coefficient (Zig Zag ordering)

red=
867 0 -1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0

Output of implementation

3.3.6. Huffman Coding

Huffman coding is a lossless data encoding algorithm. The process behind its scheme includes sorting numerical values from a set-in order of their frequency. The least frequent numbers are gradually eliminated via the Huffman tree, which adds the two lowest frequencies from the sorted list in every new “branch.” The sum is then positioned above the two eliminated lower frequency values, and replaces them in the new sorted list. Each time a new branch is created, it moves the general direction of the tree either to the right (for higher values) or the left (for lower values). When the sorted list is exhausted and the tree is complete, the final value is zero if the tree ended on a left number, or it is one if it ended on the right. This is a method of reducing complex code into simpler sequences and is common in video encoding.

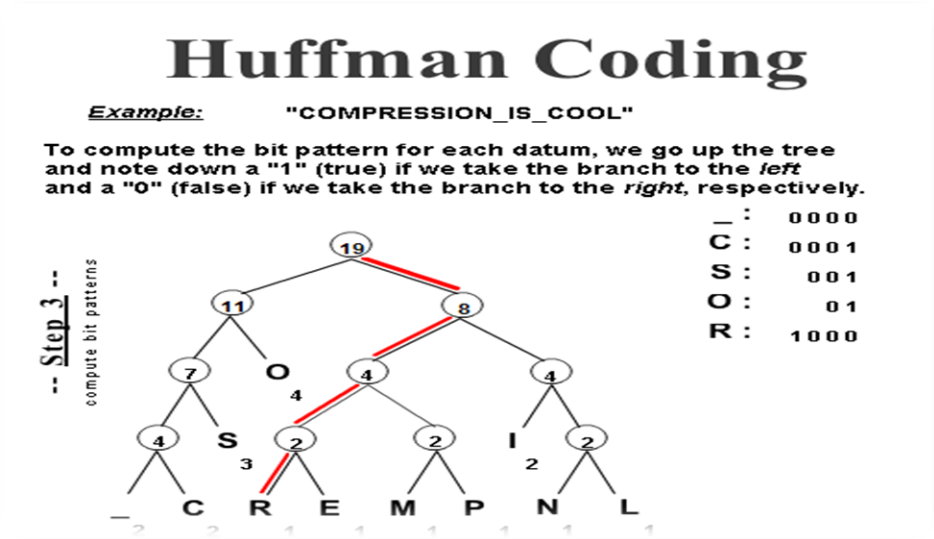


Fig. 3.25. Huffman coding

3.3.7. Entropy coding

An entropy encoding is a coding scheme that assigns codes to symbols to match code lengths with the probabilities of the symbols. Typically, entropy encoders are used to compress data by replacing symbols represented by equal-length codes with symbols represented by codes proportional to the negative logarithm of the probability

Entropy of an image is a process of decreasing the number of bits required to represent an image. So, the lower value of entropy means lower number of bits required to represent an image.

3.3.8. Baseline Decoding

Decoding Text Example:

The Huffman code for a, b, c, d is: a = 0 b = 1 1 c = 1 0 0 d= 1 0 1 Now the Huffman code for the string "aaabbcd" will be 0001111100101 – total 13 bits. ASCII code that needs 56 bits (7-character x 8 bit/character) for the string 'aaabbcd'.

3.3.9. PNSR Block Computes

The PSNR block computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is often used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image.

The Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR) are the two-error metrics used to compare image compression quality. The MSE represents the cumulative squared error between the compressed and the original image, whereas PSNR represents a measure of the peak error. The lower the value of MSE, the lower the error.

To compute the PSNR, the block first calculates the mean-squared error using the following equation:

$$MSE = M, N\ [I1(m, n) - I2(m, n)]\ 2M * N$$

In the previous equation, M and N are the number of rows and columns in the input images, respectively. Then the block computes the PSNR using the following equation:

$$PSNR=10log10(R2MSE)$$

In the previous equation, R is the maximum fluctuation in the input image data type. For example, if the input image has a double-precision floating-point data type, then R is 1. If it has an 8-bit unsigned integer data type, R is 255, etc.

Recommendation for Computing PSNR for Color Images

Different approaches exist for computing the PSNR of a color image. Because the human eye is most sensitive to luma information, compute the PSNR for color images by converting the image to a color space that separates the intensity (luma) channel, such as YCbCr. The Y (luma), in YCbCr represents a weighted average of R, G, and B. G is given the most weight, again because the human eye perceives it most easily. With this consideration, compute the PSNR only on the luma channel.

3.3.10. F5 Technique Execution by Java

Algorithm

Fig. 3.26. F5 algorithm

Project Structure

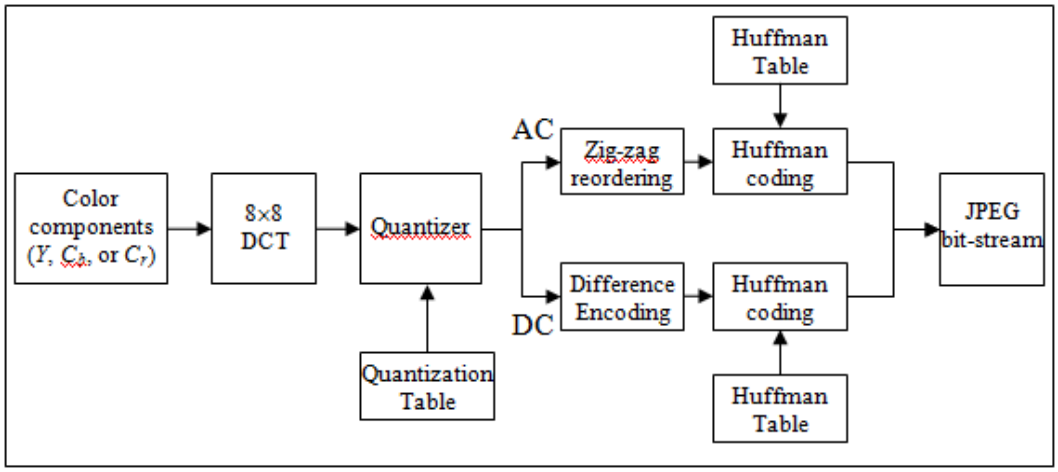


Fig. 3.27. Baseline JPEG Encoding

CH04

Results

4.1. Application's User interface for LSB Techniques

Fig.4.1 : LSB-Implementation: "Hide" Window

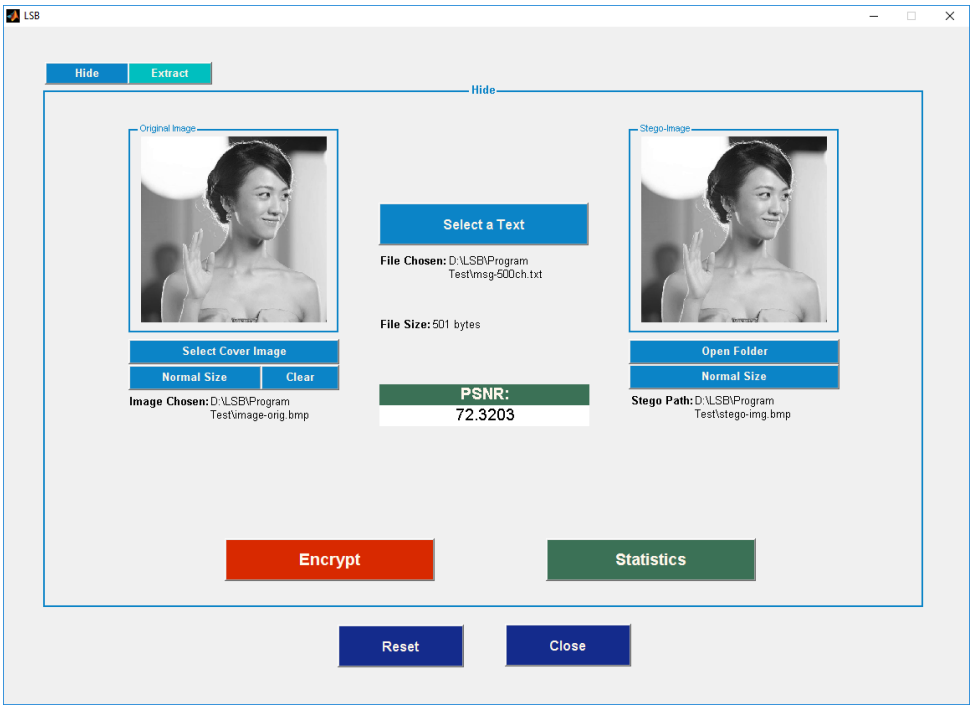
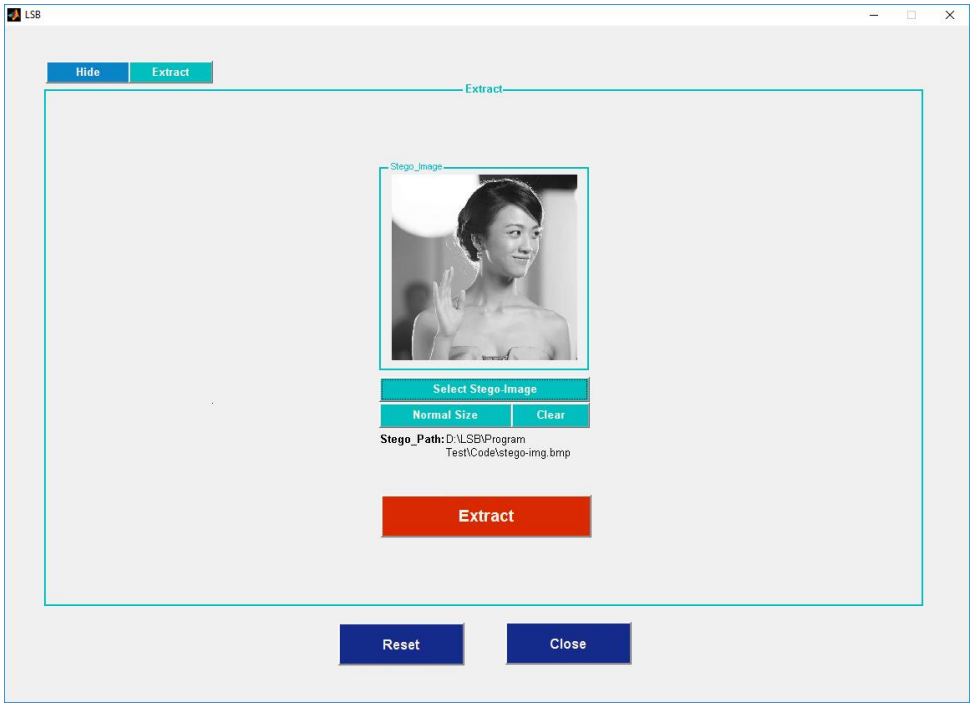


Fig.4.2 : LSB-Implementation: "Extract" Window:



First the original image, as shown in the Fig. 4.1 and the secret message are taken. Then the secret data has to be converted into binary format. Binary conversion is done by taking the American Standard Code of Information Interchange (ASCII) values of the character and converting them into binary format and generating stream of bits.

Similarly, in cover image, bytes representing the pixels are taken in single array and byte stream is generated. Message bits are taken sequentially and then are placed in LSB bit of image byte. Same procedure is followed till all the message bits are placed in image bytes. Image generated you can choose a name or the default is 'Stego-Image'

Fig. 4.2: is the inverse method of LSB technique. Choose the stego-image then click Extract, this will extract the hidden data.

4.1.1. LSB-Steganography Method Steps:

- Convert the Cover Image and The Secret Message into binary.
- Scan Cover Image pixel by pixel.
- Determine the Least significant bit of the pixel.
- Replace this Least significant bit of pixel by one-bit from Secret Message.
- Repeat this process in every pixel.

Functions

We have three main functions:

1. **LSB_HIDE** : use for hiding the secret message inside the image.
2. **LSB_EXTRACT**: use for getting the secret message from the image.
3. **Calc_PSNR**: use to calculate the PSNR between two images (cover and stego images).

4.1.1.1. LSB_HIDE code:

```
1. function stego_img = HIDE(cover_img,txt_file)
2. stego_img=cover_img;
3. stego_img=double(stego_img);
4. f_id=fopen(txt_file,'r');
5. [msg,len_total]=fread(f_id,'ubit1');
6. [m,n]=size(stego_img);
7. if len_total>m*n
8.     error('overflow');
9. end
10.     p=1;
11.     for f2=1:n
12.         for f1=1:m
13.             stego_img(f1,f2)=stego_img(f1,f2)-
   mod(stego_img(f1,f2),2)+msg(p,1);
14.             if p==len_total
15.                 break;
16.             end
17.             p=p+1;
18.         end
19.         if p==len_total
20.             break;
21.         end
22.     end
23.     stego_img=uint8(stego_img);
24.     end
```

4.1.1.2. LSB_Extract code:

```
1. function LSB_EXTRACT(img,path,name)
2. img=double(img);
3. [m,n]=size(img);
4. len_total=m*n;
5. path=strcat(path,name);
6. msg=fopen(path,'w+');
7. p=1;
8. for f2=1:n;
9.     for f1=1:m;
10.             if bitand(img(f1,f2),1)==1
11.                 fwrite(msg,1,'ubit1');
12.             else
13.                 fwrite(msg,0,'ubit1');
14.             end
15.             if p==len_total
16.                 break;
17.             end
18.             p=p+1;
19.         end
20.         if p==len_total;
21.             break;
22.         end
23.     end
```

```
24.     fclose(msg);
25.     end
```

4.1.1.3. Calc_PSNR code:

```
1. function psnr = Calc_PSNR(img1,img2)
2. [row,col] = size(img1);
3. size_host = row*col;
4. o_double = double(img1);
5. w_double = double(img2);
6. s=0;
7. for j = 1:size_host;
8.     s = s+(w_double(j) - o_double(j))^2 ;
9. end
10.     mes=s/size_host;
11.     psnr =10*log10((255)^2/mes);
12.     end
```

PSNR scales the **MSE** according to the image range. The PSNR between cover image and stego-image is given by Eq.1. A higher PSNR indicates that the quality of the stego-image is similar to the cover image.

$$\mathrm{PSNR} = 10 \times \lg\left(\frac{255^2}{\mathrm{MSE}}\right)$$

$$\mathrm{MSE} = \frac{1}{M \times N}\sum_{i=1}^{N}\sum_{j=1}^{M}\left[I(i,j) - I'(i,j)\right]^2$$

-

PSNR Equation

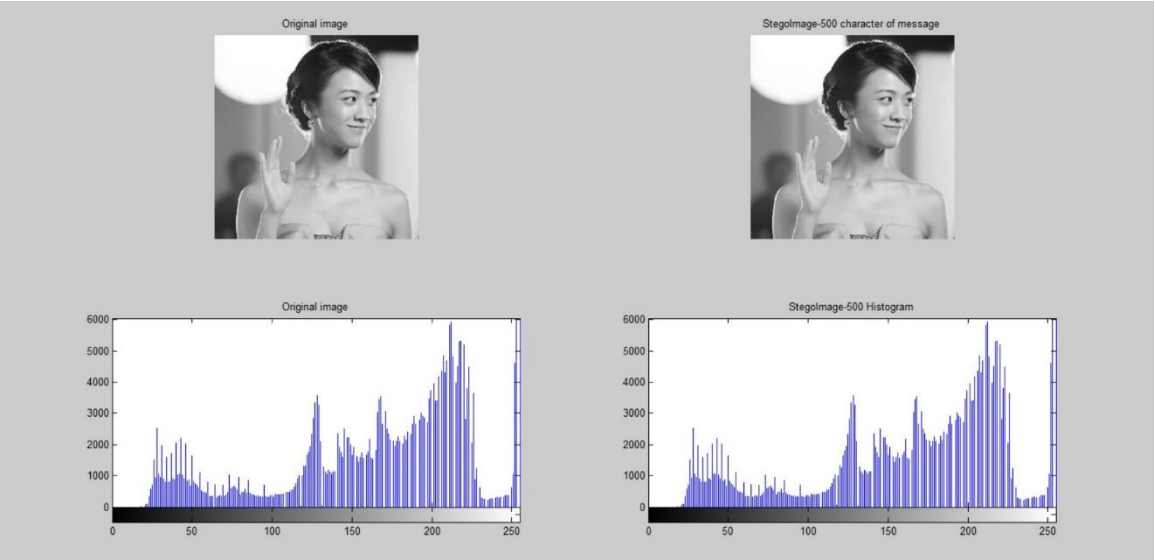
- **Results**
- **Histogram**

An image histogram is a bar chart that shows the distribution of intensities in an image. Fig. 3 shows the histogram of the original image and stego-image with hidden data of 500 character. That results a 72.3203 PSNR value.

Fig. 4 shows the histogram of the original image and stego-image with hidden data of 4000 character. That results a 61.9974 PSNR value.

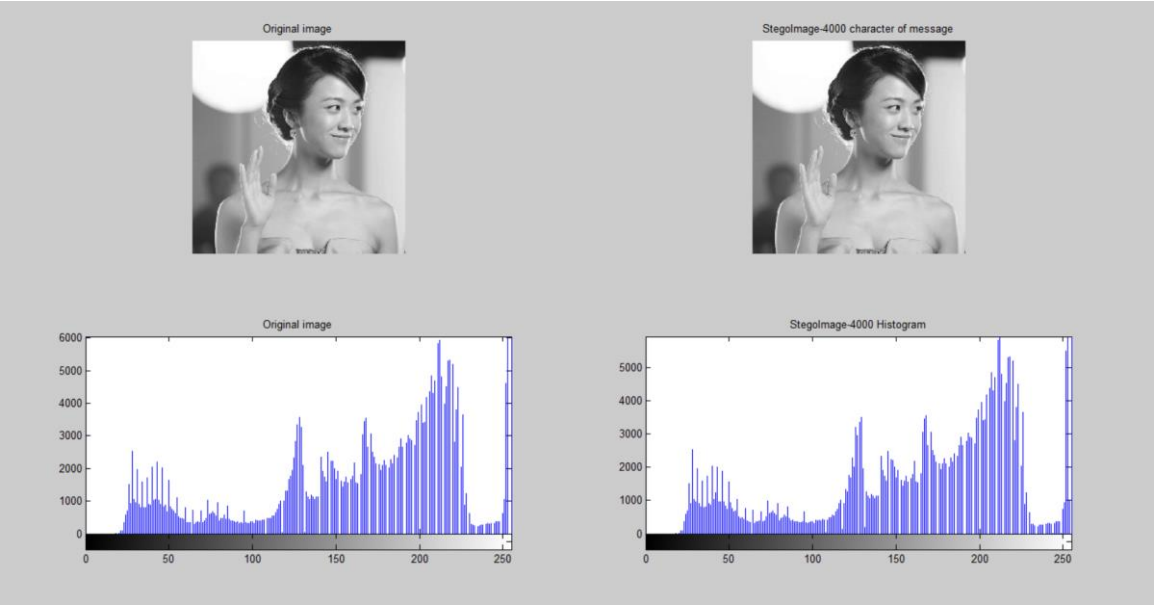
Fig. 5 shows the histogram of the original image and stego-image with hidden data of 32000 character. That results a 52.5338 PSNR value.

Fig. 4.3 : Hide message with length 500 character:



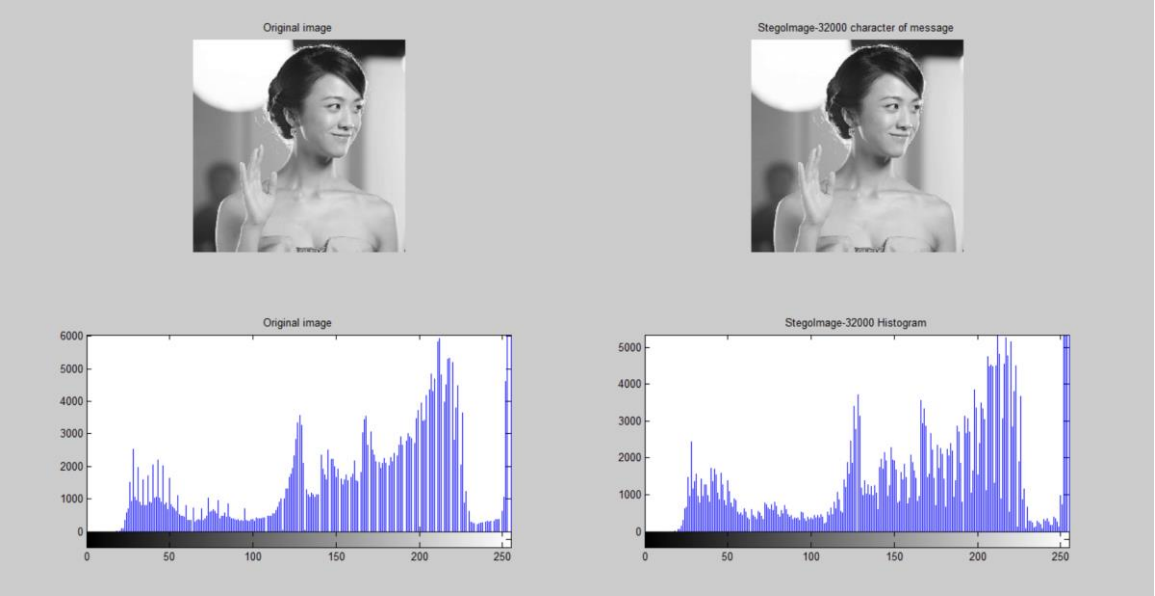
PSNR = 72.3203

Fig. 4.4 : Hide message with length 4000 character:



PSNR = 61.9974

Fig. 4.5 : Hide message with length 32000 character:



PSNR = 52.5338

- **Compare between different size of hiding message inside the cover Image:**

Image (2,880,000 bit)	Message Size inside it	PSNR with Cover Image
image-stego-500.bmp	500*8= 4000 bit	72.3203
image-stego-4000.bmp	4000*8= 32,000 bit	61.9974
image-stego-32000.bmp	32000*8= 256,000 bit	52.5338

4.2. Application's User interface for F5 Techniques

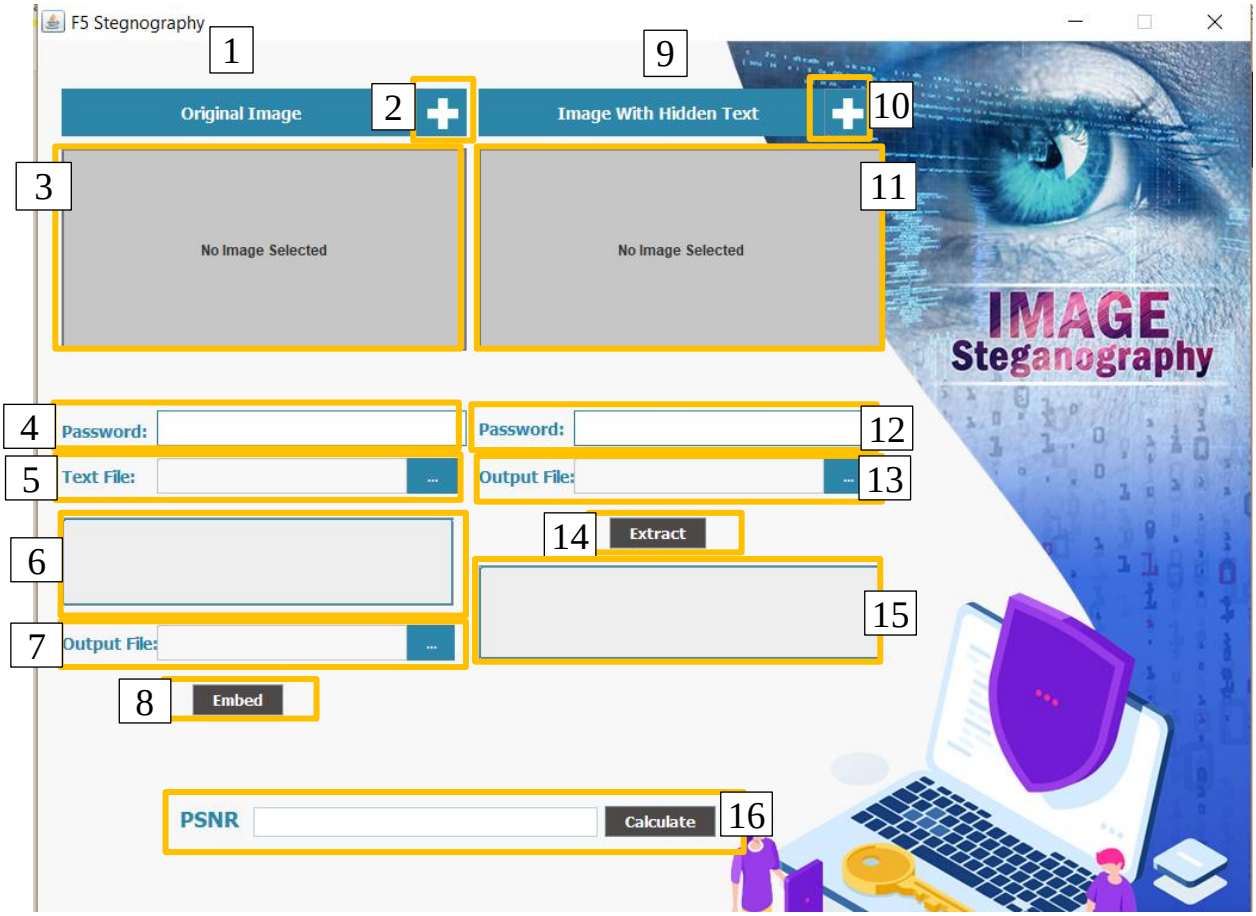


Fig. 4.6. Gui F5 Techniques

1- Embedder form	9- Extractor form
2- Select embed image	10- Select extract image
3- Selected embed image	11- Selected extract image
4- Embed password	12- Extract password
5- Embed text file	13- Output text file name
6- Embed text file content	14- Extract button

7- Output image file name 8- Embed button	15- Extracted text 16- PSNR calculation

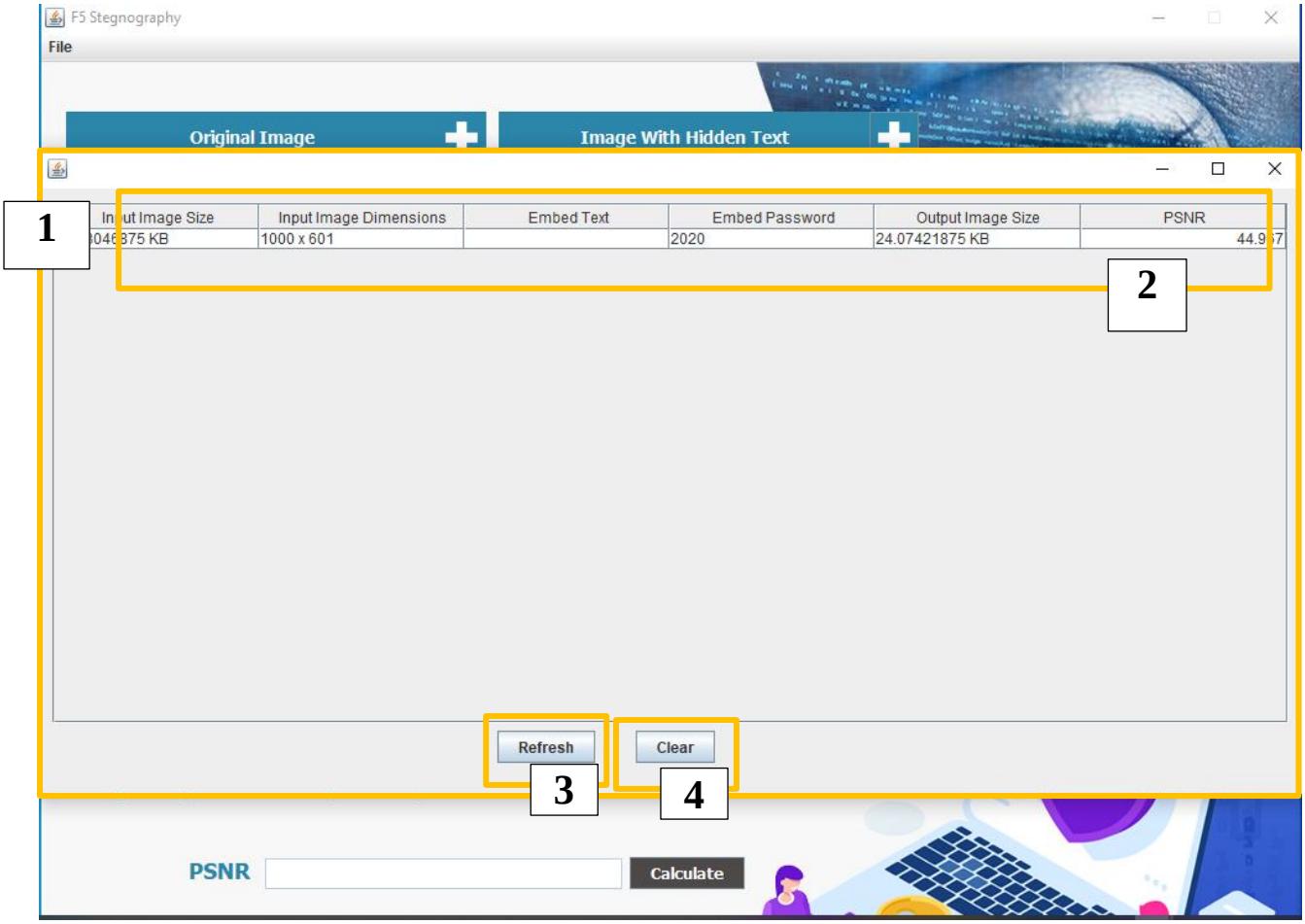


Fig. 4.7. Gui F5 Techniques

1- Window to display testing of image and show details for each test. 2- Place that display details as table.	3- Bottom Refresh data when update file. 4- Bottom Clear to delete all data of file.

4.3. Usage Guide

4.3.1. Embedding text to image

1- In Embedder form click the select image button

2- After image is loaded, write the embed password.

3- Click the select text to embed file and select file contains text you want to embed.

4- Click select output file and write your output image file name.

5- Click Embed button to start embedding the text into the selected image.

4.3.2. Extracting text from image

1- In the Extractor form click the select image button

2- After image is loaded, write the extract password.

3- Select text output file

4- Click the Extract button to extract the text from the selected image.

4.3.3. Calculating PSNR

1- In the Embedder form, select the original image.

2- In the Extractor form, select the encrypted image.

3- Click the Calculate button in PSNR calculation section.

4.4. Source Code F5 and LSB on GitHub

The full project source code could be obtained from the below link

https://github.com/MahmoudAdlyAbdelZaher/Image-Steganography-Graduation-project-2019.git

- **Compare between different size of hiding message inside the cover Image:**

Image.jpg	Size Image	Size Text	Size Image with text	PSNR
Original.jpg	858 KB	35.8 MB	249 KB	31.4604
Original.jpg	858 KB	5.94 KB	282 KB	32.3118
Original.jpg	858 KB	546 bytes	286 KB	32.4218

- **Conclusion:**
  - PSNR **Decrease** when text **Increase**.
  - Size Image **Decrease** when Text Increase.

- **Histogram for images:**

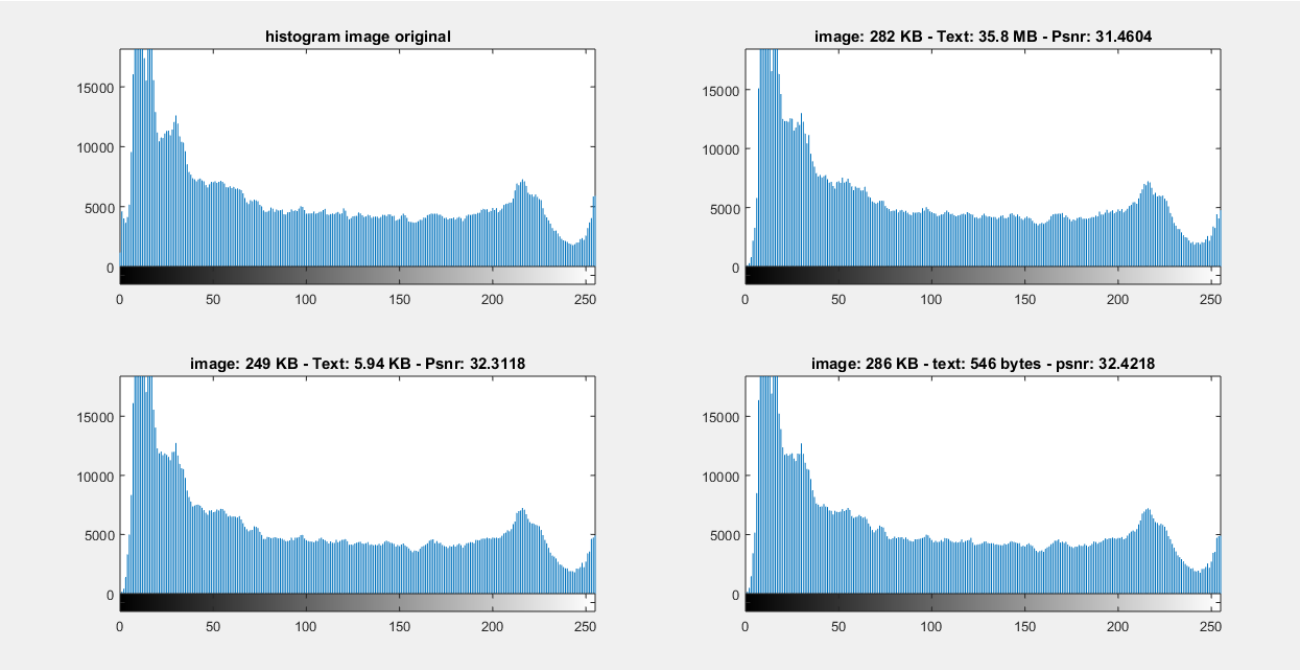


Fig. 4.8. Histogram Compare between Image

4.5. Some Coding of program F5

4.5.1. Embedded Image with Text

```
1. package f5stegnography;
2.
3. import java.awt.Image;
4. import java.awt.Toolkit;
5. import java.awt.image.BufferedImage;
6. import java.io.File;
7. import java.io.FileInputStream;
8. import java.io.FileNotFoundException;
9. import java.io.FileOutputStream;
10.       import java.io.IOException;
11.       import java.io.StringBufferInputStream;
12.       import java.util.logging.Level;
13.       import java.util.logging.Logger;
14.       import javax.imageio.ImageIO;
15.
16.       public class Embedder {
17.
18.           private static File inFile;
19.           private static File outFile;
20.           private static File embedFile;
21.           private static FileOutputStream dataOutStream;
22.           private static Image image;
23.           private static MJpegEncoder jpg;
24.
25.
26.           public static String embed(String embMsgFile, String
    inFileName, String outFileName, String password) {
27.               outFile = new File(outFileName); //identify output file
28.               inFile = new File(inFileName);
29.               String PSNRValue = "";
30.               //embedFile = new File(embMsgFile);
31.               //String password = "abc123";
32.               if (inFile.exists()) {
33.                   try {
34.                       dataOutStream = new FileOutputStream(outFile);
35.                       image =
    Toolkit.getDefaultToolkit().getImage(inFileName);
36.                       jpg = new MJpegEncoder(image, 80, dataOutStream,
    "comment");
37.                       //jpg.Compress(new FileInputStream(embedFile),
    password);
38.                       jpg.Compress(new
    StringBufferInputStream(embMsgFile), password);
39.
40.                       //jpg.Compress();
41.                       dataOutStream.close();
42.       //------------------------  PNSR ------------ Claculate ---------
    ----------------------------
43.                       BufferedImage im1 = ImageIO.read(inFile);
44.                       BufferedImage im2 = ImageIO.read(outFile);
45.                       Psnr PSNR = new Psnr();
46.                       PSNRValue = PSNR.PSNR(im1, im2);
47.       //-----------------------------------------------------------------
    ----------------------------
```

```
48.
49.                     } catch (FileNotFoundException ex) {
50.                         //catch clause not set
51.                     } catch (IOException ex) {
52.
   Logger.getLogger(Embedder.class.getName()).log(Level.SEVERE, null, ex);
53.                     }
54.                 }
55.
56.                 return PSNRValue;
57.             }
58.         }
```

4.5.2. Extract Text of Image

```
1. package f5stegnography;
2.
3. import java.io.File;
4. import java.io.FileInputStream;
5. import java.io.FileOutputStream;
6. import java.io.IOException;
7. import java.io.InputStream;
8. import java.io.OutputStream;
9.
10.        import huffman.HuffmanDecode;
11.        import crypt.F5Random;
12.        import crypt.Permutation;
13.        import java.io.BufferedOutputStream;
14.        import java.util.logging.Level;
15.        import java.util.logging.Logger;
16.
17.        public class Extractor {
18.
19.            private static File f; // carrier file
20.
21.            private static byte[] carrier; // carrier data
22.
23.            private static int[] coeff; // dct values
24.
25.            private static FileOutputStream fos; // embedded file (output
   file)
26.            private static StringBuilder builder = new StringBuilder(new
   String());
27.            // private static String embFileName; // output file name
28.
29.            ///private static String password;
30.            private static byte[] deZigZag = {
31.                0, 1, 5, 6, 14, 15, 27, 28, 2, 4, 7, 13, 16, 26, 29, 42,
   3, 8, 12, 17, 25, 30, 41, 43, 9, 11, 18, 24, 31,
32.                40, 44, 53, 10, 19, 23, 32, 39, 45, 52, 54, 20, 22, 33,
   38, 46, 51, 55, 60, 21, 34, 37, 47, 50, 56, 59, 61,
33.                35, 36, 48, 49, 57, 58, 62, 63};
34.
35.            public static void extract(final InputStream fis, final int
   flength, final OutputStream fos, final String password)
36.                    throws IOException {
37.                carrier = new byte[flength];
38.                fis.read(carrier);
39.                final HuffmanDecode hd = new HuffmanDecode(carrier);
```

```
40.                 System.out.println("Huffman decoding starts");
41.                 coeff = hd.decode();
42.                 System.out.println("Permutation starts");
43.                 final F5Random random = new
    F5Random(password.getBytes());
44.                 final Permutation permutation = new
    Permutation(coeff.length, random);
45.                 System.out.println(coeff.length + " indices shuffled");
46.                 int extractedByte = 0;
47.                 int availableExtractedBits = 0;
48.                 int extractedFileLength = 0;
49.                 int nBytesExtracted = 0;
50.                 int shuffledIndex = 0;
51.                 int extractedBit;
52.                 int i;
53.                 System.out.println("Extraction starts");
54.                 // extract length information
55.                 for (i = 0; availableExtractedBits < 32; i++) {
56.                     shuffledIndex = permutation.getShuffled(i);
57.                     if (shuffledIndex % 64 == 0) {
58.                         continue; // skip DC coefficients
59.                     }
60.                     shuffledIndex = shuffledIndex - shuffledIndex % 64 +
    deZigZag[shuffledIndex % 64];
61.                     if (coeff[shuffledIndex] == 0) {
62.                         continue; // skip zeroes
63.                     }
64.                     if (coeff[shuffledIndex] > 0) {
65.                         extractedBit = coeff[shuffledIndex] & 1;
66.                     } else {
67.                         extractedBit = 1 - (coeff[shuffledIndex] & 1);
68.                     }
69.                     System.out.println(extractedBit);
70.                     extractedFileLength |= extractedBit <<
    availableExtractedBits++;
71.                 }
72.                 // remove pseudo random pad
73.                 extractedFileLength ^= random.getNextByte();
74.                 extractedFileLength ^= random.getNextByte() << 8;
75.                 extractedFileLength ^= random.getNextByte() << 16;
76.                 extractedFileLength ^= random.getNextByte() << 24;
77.                 int k = extractedFileLength >> 24;
78.                 k %= 32;
79.                 final int n = (1 << k) - 1;
80.                 extractedFileLength &= 0x007fffff;
81.                 System.out.println("Length of embedded file: " +
    extractedFileLength + " bytes");
82.                 availableExtractedBits = 0;
83.                 if (n > 0) {
84.                     int startOfN = i;
85.                     int hash;
86.                     System.out.println("(1, " + n + ", " + k + ") code
    used");
87.                     extractingLoop:
88.                     do {
89.                         // 1. read n places, and calculate k bits
90.                         hash = 0;
91.                         int code = 1;
92.                         for (i = 0; code <= n; i++) {
93.                             // check for pending end of coeff
94.                             if (startOfN + i >= coeff.length) {
```

```
95.                                 break extractingLoop;
96.                             }
97.                             shuffledIndex =
   permutation.getShuffled(startOfN + i);
98.                             if (shuffledIndex % 64 == 0) {
99.                                 continue; // skip DC coefficients
100.                            }
101.                            shuffledIndex = shuffledIndex - shuffledIndex
   % 64 + deZigZag[shuffledIndex % 64];
102.                            if (coeff[shuffledIndex] == 0) {
103.                                continue; // skip zeroes
104.                            }
105.                            if (coeff[shuffledIndex] > 0) {
106.                                extractedBit = coeff[shuffledIndex] & 1;
107.                            } else {
108.                                extractedBit = 1 - (coeff[shuffledIndex]
   & 1);
109.                            }
110.                            if (extractedBit == 1) {
111.                                hash ^= code;
112.                            }
113.                            code++;
114.                        }
115.                        startOfN += i;
116.                        // 2. write k bits bytewise
117.                        for (i = 0; i < k; i++) {
118.                            extractedByte |= (hash >> i & 1) <<
   availableExtractedBits++;
119.                            if (availableExtractedBits == 8) {
120.                                // remove pseudo random pad
121.                                extractedByte ^= random.getNextByte();
122.                                fos.write((byte) extractedByte);
123.                                extractedByte = 0;
124.                                availableExtractedBits = 0;
125.                                nBytesExtracted++;
126.                                // check for pending end of embedded data
127.                                if (nBytesExtracted ==
   extractedFileLength) {
128.                                    break extractingLoop;
129.                                }
130.                            }
131.                        }
132.                    } while (true);
133.                } else {
134.                    System.out.println("Default code used");
135.                    for (; i < coeff.length; i++) {
136.                        shuffledIndex = permutation.getShuffled(i);
137.                        if (shuffledIndex % 64 == 0) {
138.                            continue; // skip DC coefficients
139.                        }
140.                        shuffledIndex = shuffledIndex - shuffledIndex %
   64 + deZigZag[shuffledIndex % 64];
141.                        if (coeff[shuffledIndex] == 0) {
142.                            continue; // skip zeroes
143.                        }
144.                        if (coeff[shuffledIndex] > 0) {
145.                            extractedBit = coeff[shuffledIndex] & 1;
146.                        } else {
147.                            extractedBit = 1 - (coeff[shuffledIndex] &
   1);
148.                        }
```

```
149.                     extractedByte |= extractedBit <<
    availableExtractedBits++;
150.                     if (availableExtractedBits == 8) {
151.                         // remove pseudo random pad
152.                         extractedByte ^= random.getNextByte();
153.                         fos.write((byte) extractedByte);
154.
155.                         extractedByte = 0;
156.                         availableExtractedBits = 0;
157.                         nBytesExtracted++;
158.                         if (nBytesExtracted == extractedFileLength) {
159.                             break;
160.                         }
161.                     }
162.                 }
163.             }
164.             if (nBytesExtracted < extractedFileLength) {
165.                 System.out.println("Incomplete file: only " +
    nBytesExtracted + " of " + extractedFileLength
166.                         + " bytes extracted");
167.             }
168.         }
169.
170.         public static String extract(String inFileName, String
    password, String embFileName) {
171.
172.             // password = "abc123";
173.             FileInputStream fis = null;
174.             try {
175.
176.                 f = new File(inFileName);
177.
178.                 fis = new FileInputStream(f);
179.                 fos = new FileOutputStream(new File(embFileName));
180.                 extract(fis, (int) f.length(), fos, password);
181.
182.             } catch (final Exception e) {
183.                 e.printStackTrace();
184.             } finally {
185.                 try {
186.                     fos.close();
187.                     fis.close();
188.                 } catch (IOException ex) {
189.
    Logger.getLogger(Extractor.class.getName()).log(Level.SEVERE, null, ex);
190.                 }
191.             }
192.
193.             return "Builder String: " + builder.toString();
194.         }
195.     }
```

4.5.3. PSNR:

```
1. package f5stegnography;
2.
3. import java.awt.image.BufferedImage;
4. import java.awt.image.Raster;
5.
```

```
6. public class Psnr {
7.
8.     public Psnr() {
9.     }
10.
11.
12.          public double logbase10(double x) {
13.              return Math.log(x) / Math.log(10);
14.          }
15.      //    public String  PSNR(BufferedImage im1, BufferedImage im2)
16.
17.          public   String PSNR(BufferedImage im1, BufferedImage im2) {
18.              assert (im1.getType() == im2.getType()
19.                      && im1.getHeight() == im2.getHeight()
20.                      && im1.getWidth() == im2.getWidth());
21.
22.              double mse = 0;
23.              int width = im1.getWidth();
24.              int height = im1.getHeight();
25.              Raster r1 = im1.getRaster();
26.              Raster r2 = im2.getRaster();
27.              for (int j = 0; j < height; j++) {
28.                  for (int i = 0; i < width; i++) {
29.                      mse += Math.pow(r1.getSample(i, j, 0) -
   r2.getSample(i, j, 0), 2);
30.                  }
31.              }
32.
33.              mse /= (double) (width * height);
34.              double psnr = 10.0 * logbase10(Math.pow(255, 2) / mse);
35.              System.err.println("PSNR = " + psnr);
36.              return String.format("%.4f", psnr);
37.          }
38.      }
```

4.5.4. Histogram by MATLAB For F5 Techniques:

```
1. im = imread('C:\Users\mahmo\Desktop\image\original.jpeg');
2. im1 =
   imread('C:\Users\mahmo\Desktop\image\image_with_text_35.8MB_psnr_31.4604
   .jpg');
3. im2 =
   imread('C:\Users\mahmo\Desktop\image\image_with_text_5.94KB_psnr_32.3118
   .jpg');
4. im3 =
   imread('C:\Users\mahmo\Desktop\image\image_with_text_546bytes_psnr_32.42
   18.jpg');
5.
6. input = rgb2gray(im);
7. input1 = rgb2gray(im1);
8. input2 = rgb2gray(im2);
9. input3 = rgb2gray(im3);
10.      % ----------- original iamge --------------
11.      subplot(2,2,1);
12.      imhist(input);
13.      title('histogram image original');
14.
15.      % ----------- image: 282 KB - text: 35.8MB - psnr: 31.4604 ------
   --------
```

```
16.       subplot(2,2,2);
17.       imhist(input1);
18.       title('image: 282 KB - Text: 35.8 MB - Psnr: 31.4604');
19.
20.       % ----------- image: 249 KB - Text: 5.94KB - Psnr: 32.3118 ------
   --------
21.       subplot(2,2,3);
22.       imhist(input2);
23.       title('image: 249 KB - Text: 5.94 KB - Psnr: 32.3118');
24.
25.       % ----------- image: 286 KB - text: 546 bytes - psnr: 32.4218 ---
   -----------
26.       subplot(2,2,4);
27.       imhist(input3);
28.       title('image: 286 KB - text: 546 bytes - psnr: 32.4218');
29.
30.
```

CH05

Conclusion and Future Work

5.1. Conclusion

Steganography is an effective way of secure communication. You can first encrypt a confidential file and then hide it inside an image of other kind of file before sending it to some other. It will decrease the chance of being intercept. If you just send the file by encrypting, attacker will try to decrypt it by various ways. However, if he will only find a normal image file, he will have no clue. This technique is very easy to use but very difficult to detect. So, it can be used by government organizations to use as the way to send and receive files securely.

Steganography is not intended to replace cryptography but rather to supplement 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. Steganography is still a fairly new concept to the general public although this is likely not true in the world of secrecy and espionage. Digital watermark technology is currently being used to track the copyright and ownership of digital content. Efforts to improve the robustness of the watermarks are necessary to ensure that the watermarks and embedded information can securely defend against watermarking attacks. With continuous advancements in technology it is expected that in the near future more efficient and advanced techniques in steganalysis willemerge that will help law enforcement to better detect illicit materials transmitted through the Internet. The tutorial introduces a tiny part of the art of steganography. Steganography goes well beyond simply hiding text information in an image. Steganography applies not only to digital images but to other media as well, such as audio files, communication channels, and other text and binary files.

5.2. The Future of Image Steganography

1) first encrypted and later hidden
2) using aspiring for better outcome in image steganography it is important to keep three aspects in mind.
3) the capacity of the cover image i.e., it should be able to store utmost data inside it.
4) being imperceptibility should provide an unaltered image after data hiding
5) it should be secure against attacks. aspects efficaciously will help in achieving the goal of image steganography

References

1) Digital Watermarking and Steganography.
2) Steganography in digital Media principles, algorithms and application
3) Image Steganography Techniques", Ravi K Sheth and Rashmi M. Tank, 2015.
4) "Information Hiding in image and audio files", Kekre, H B, 2011.
5) "An Overview of Image Steganographic Techniques", Archana.O. Vyas and Dr. Sanjay.V. Dudul, 2015.
6) https://resources.infosecinstitute.com/steganography-and-tools-to-perform-steganography/#gref
7) https://www.malc0de.org/list-of-best-steganography-tools/
8) https://www.greycampus.com/blog/information-security/top-must-have-tools-to-perform-steganography
9) http://download.cnet.com/Xiao-Steganography/3000-2092_4-10541494.html
10) http://imagesteganography.codeplex.com/
11) https://www.softpedia.com/get/Security/Encrypting/SteganPEG.shtml
12) http://sourceforge.net/projects/crypture/
13) http://www.bestfreewaredownload.com/freeware/t-free-steganographx-plus-freeware-yeipgmrk.html
14) http://www.softpedia.com/get/Security/Security-Related/rSteg.shtml
15) http://www.ssuitesoft.com/ssuitepicselsecurity.htm
16) http://download.cnet.com/Our-Secret/3000-2144_4-75553911.html

17) http://camouflage.unfiction.com/Download.html

18) http://sourceforge.net/projects/openstego/files/

19) http://download.cnet.com/SteganPEG/3000-2193_4-75914262.html

20) http://download.cnet.com/Hide-N-Send/3000-2092_4-75728348.html

21) Faculty of Electrical Engineering Master's thesis2009_5

Appendix:

The full project source code could be obtained from the below link

https://github.com/MahmoudAdlyAbdelZaher/Image-Steganography-Graduation-project-2019.git