Image Steganography using Block differences

By

Sara Naureen

Submitted to the department of Computer Engineering in fulfillment of the requirement for the degree of

Masters in Science
In
Computer Software Engineering

Thesis Supervisor
Brig Dr Muhammad Younus Javed

College of Electrical and Mechanical Engineering
National University of Science and Technology
2009

Acknowledgments

All praise to Allah, the most beneficent and most merciful.

I would like to express my gratitude to my supervisor, Brig Dr Muhammad Younus Javed, whose expertise, understanding, and patience, added considerably to my graduate experience. I appreciate his vast knowledge. I would like to thank the other members of my committee, Lt. Col. Dr. Farooq Azhar, and Lt. Col. Dr Rashid Ahmed , Dr. Asia Khanam for the assistance they provided at all levels of the research project.

I would also like to thank my parents and sisters for constantly encouraging me to complete my thesis, especially my mother who took care of my son while I worked and my husband and mother-on-law for providing me a facilitating environment for completing my research. I would not have been able to complete this thesis without the support of these people.

Dedicated to my mother

Abstract

Steganography is a means to hide the existence of information exchange. Using this technique the sender embeds the secret information in some other media. This is done by replacing useless data in ordinary computer files with some other invisible information. The hidden information could be simple text, cipher text or even images. The media used as the embedding plane could be an image, audio, video or a text files. Using steganography ensures that no one apart from the sender and the receiver knows about the existence of the message.

In this thesis, a steganography method based on block differences is used. Each block consists of three pixels. Two different differences are calculated for each block – between the first and the second pixel and also between the second and the third pixel. The most recurring absolute block difference in the whole image is used for embedding the secret message. To embed bit "1" or "0"., the selected difference is either increased or left unchanged respectively. At the most a single block can embed two bits "11" and only the middle pixel needs to be changed. Data Compression is used to reduce the size of information so that the data hiding capacity of the images could be increased.

The major advantage of using this technique is that the image quality is not degraded even if the number of embedded characters is increased. The algorithm operates above 50 db in most of the test cases. The average payload capacity is also considerably high.

TABLE OF CONTENTS

TABLE OF FIGURES

List of tables

Chapter 1
The Art of Information Hiding

1.1 Introduction

Steganography is the art of hiding information within some other information. The secret message is embedded into another message in such a way that none apart from the sender and recipient can know about the presence of the message. In other words, using this technology even conceals the fact that a secret message is being transmitted [1].

Merely this fact makes Steganography an essential part of modern day digital communications. The other technique that is used for secret communication is Cryptography where the sender uses an encryption key to scramble the message, this scrambled message is transmitted through public channel, and the reconstruction of the original, unscripted message is possible only if the receiver has the appropriate decryption key [3].

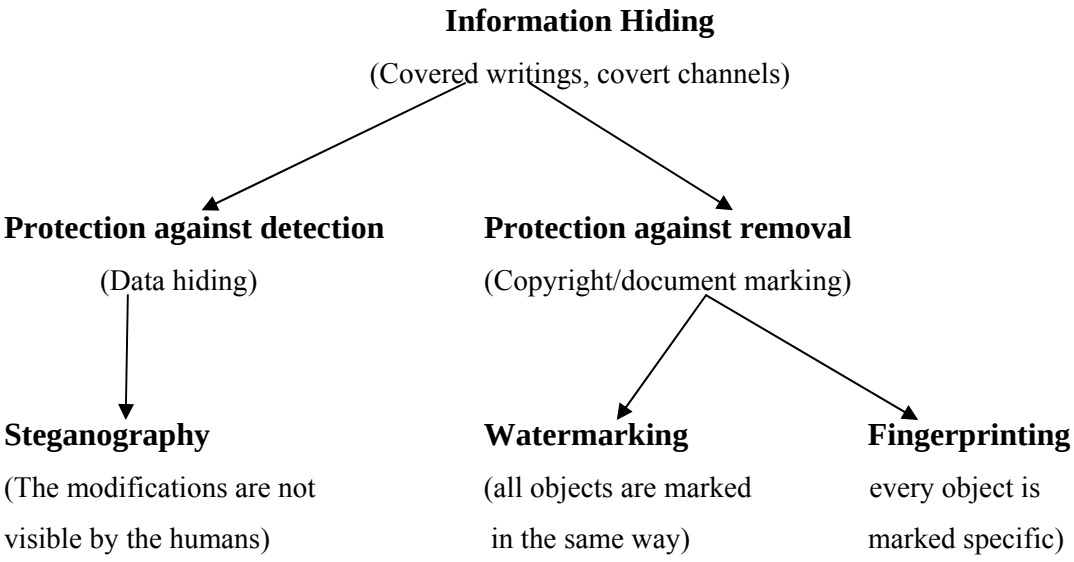


Figure 1: Directions within Information Hiding

Two general directions can be distinguished within information hiding.

- **Protection against detection**

  This is achieved using schemes that do not transform the original unmarked object in a perceptible manner; the modifications are not visible to the humans or computers.

- **Protection against removal**

  These techniques emphasize that the schemes should be robust to common attacks; however it is impossible to remove the hidden data without degrading the object's quality rendering it useless. They are two different techniques for protection against removal. In both cases, the information is supposed to be invisible, but it should be very difficult to remove it.

    - Watermarking is the process of embedding marks in digital documents (sounds, images, binaries, etc) for later identification like the watermarks used for marking a bank note.

    - Fingerprinting is the process of embedding a serial number into every copy of an object. This serial number can be used to detect the break of a license agreement.

If in one application we want to achieve confidentiality, then we can either use Encryption or Steganography for protection against detection. If we use encryption, only the owner of the key (that is the receiver) can read the secret message, but anybody can see that the two parties are having a secret communication. If we use Steganography, the very existence of the secret message is hidden without using encryption.

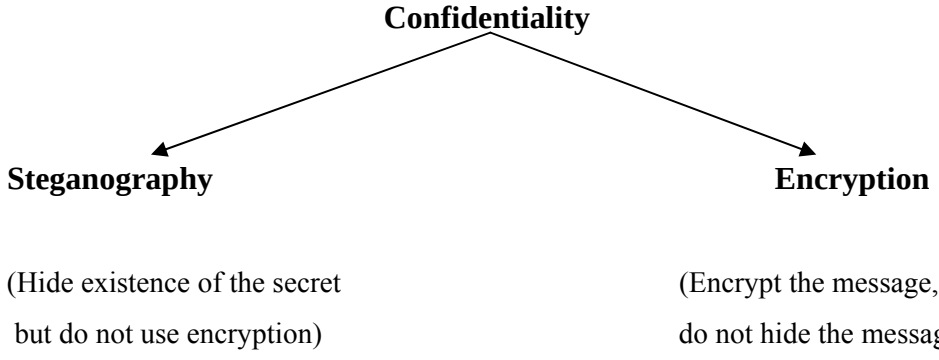


Figure 2: Achieving confidentiality

Protecting the object's contents is another goal in many applications (Figure 3). What the author wants is to seal his document or make it difficult to trace. Sealing the documents is achieved using digital signatures. Unlike digital signatures, the watermarks used for tracing documents are spread over the covered text, while the length of the document is unchanged.

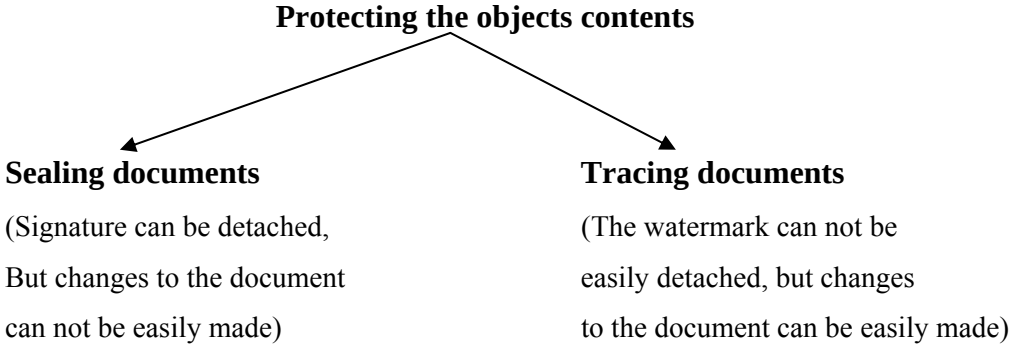


Figure 3: Protecting the documents.

1.2 History of Steganography

Throughout history, quite a lot of methods and variation have been used to hide information. David Kahn's "The Code breakers" provides an excellent accounting of this history. Bruce Norman describes numerous tales of cryptography and Steganography during times of war in Secret Warfare; The Battle of Codes and Ciphers. [4]

"Histories of Herodotus" is one of the earliest documents with the mention of Steganography. In ancient Greece, text was written on wax covered tablets. In one story Demeratus wanted to notify Sparta that Xerxes intended to invade Greece. To avoid capture, he unraveled the wax from wax tablets and wrote his message on the underlying wood. The tablets were then covered again with the wax in order to conceal the presence of any text.

Another creative scheme was to shave the head of the messenger. The bald was then used to print a tattoo (a message or an image) on the messenger's head. The messenger's hair was then allowed

to grow so that the tattoo could be obscured. This used to make the message undetectable unless the messenger's head was shaved again.

Another common form of invisible writing is through the use of invisible inks. Such inks were used with much success in World War II. An innocent letter may contain a very different message written between the lines. Early in World War II stenographic technology consisted almost exclusively of invisible inks. Common sources for invisible inks are milk, vinegar, and fruit juices. All of these darken when heated.

As the time advanced, new sophisticated invisible inks were developed that react to various chemicals. Null ciphers (unencrypted messages) were also used where the original message is "camouflaged" in a naïve looking message.

As message detection improved, new technologies were developed which could pass more information and appear less conspicuous. The Germans developed microdot technology. Microdot are photographs the size of printed period having the clarity of standard-sized typewritten pages. The first microdot was discovered masked as a period on a typed envelope. The message was not hidden, nor encrypted. It was just so small as to not draw attention to itself (for a while).

Nowadays, Steganography is not forgotten; there are many real life applications of it. Stenographic software is very new but its potential for hiding data is worth appreciating. For example, the images are represented in computers as an array of numbers. One of the modifications we can make is to replace the least significant bit of the original image with the secret bits and the image will not change – most graphic standards specify more gradations of colors then the human eye can notify. We can store 64 kilobytes message in a 1024x1024 gray scale picture this way.

We have seen that concerns about communicating via hidden channels exist since a very long time. We have seen that the method used for hiding information evolved from very simple techniques like writing underneath the wax of writing tablets to imperceptible modifications that can distinguish only by certain persons, who have the clue.

1.3 Terminology

The various information hiding techniques can be classified as given in figure 4

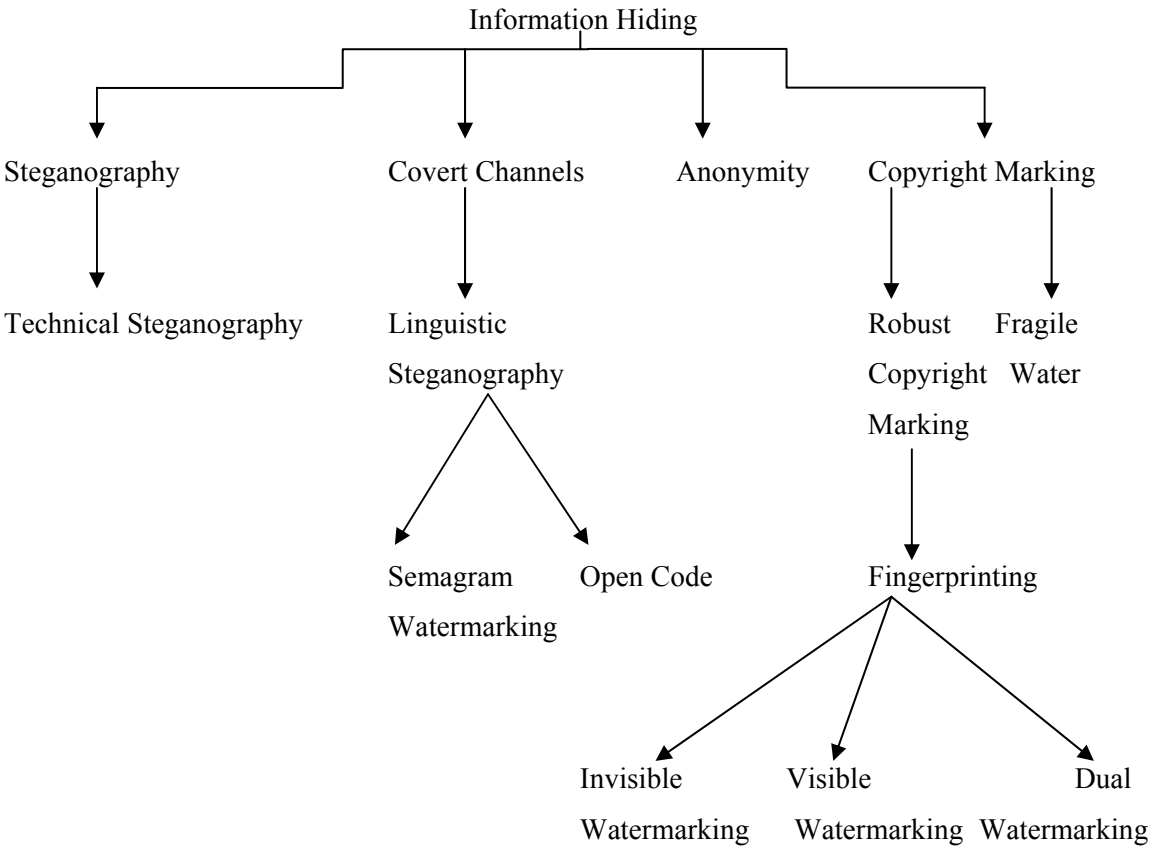


Figure 4: Information Hiding Tree

1.3.1 Steganography

It is the art of communication in a way which conceals the existence of secret message in the main information [1]. The word Steganography comes from Greek meaning (covered writing). Steganography is an ancient art of embedding private messages in seemingly innocuous messages in such a way that prevents the detection of the secret messages by a third party. In other words, Steganography means establishing covert channels. A covert channel is a secret communication channel used for transmitting information.

1.3.2 Watermarking

Watermarking is the process that embeds data called a watermark, tag or label into a multimedia object such that watermark can be detected or abstracted later to make an assertion about the object. The object may be an image or audio or video. It may also be text only.

A watermark is an identifying feature, like a company logo, which can be used to provide protection of some "cover" data. A watermark may be either visible i.e. perceptible, or invisible i.e. imperceptible, both of which offer specific advantages when it comes to protecting data. Or any piece of data may be used as a watermark. The most common watermarks used, include corporate or academic logos, number sequences, and also watermarks consisting of black and white dots.

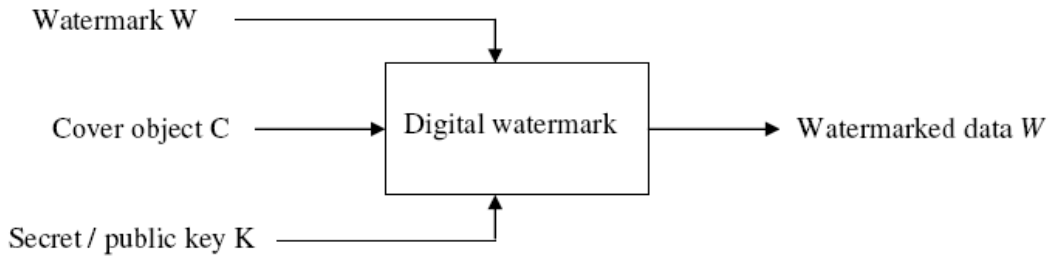


Figure 5: Digital watermarking – Embedding

A watermark is a pattern of bits inserted into a digital image file that identifies the file's copyright information (author, rights, etc.). The name “watermark” is derived from the faintly visible marks imprinted on organizational stationery.

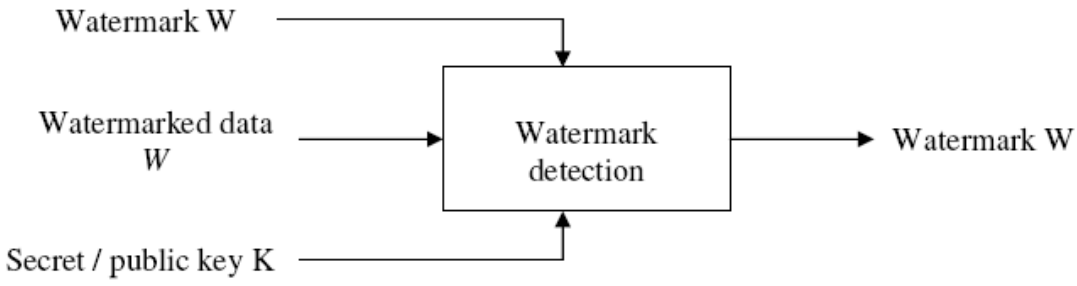


Figure 6: Digital watermarking – Decoding

Watermarks may be used to prove ownership of data, and also as an attempt to enforce copyright restrictions. For example, a web-based crawler may identify the watermark in a "copied" file.

1.3.3 Fingerprinting & Labelling

Fingerprints are sometimes called as labels. Digital watermarking differs from “digital fingerprinting”. Fingerprinting is the process of adding fingerprints to an object and recording them, or identifying and recording fingerprints that are already intrinsic to the object making that object unique as compared to other similar objects. Digital fingerprinting produces a metafile that describe the contents of the source file. [5]

1.3.4 Digital Signature

A digital signature is based upon the idea of public key encryption, a form of asymmetric cryptography [6]. A private key is used to encrypt a hashed version of an image. This encrypted file forms the unique: signature” for the image since only the entity signing the image has the knowledge of the private key. An associated public key can be used to decrypt the signature. The image under question can be hashed using the same hashing function as used originally. If these hashes match then the image is authentic. In particular, when combined with secure timestamp, a digital signature can be used as a proof of first authorship.

1.3.5 Cryptography

Cryptography is the art that consists of mangling information into apparent unintelligible information. The basic service that cryptography offers is the ability of transmitting information between persons in a way that prevent the third party from reading it.

1.3.6 Encryption

It is the process of disguising a message in such a way as to hide its substance. This process requires the encoding process and the key (which may be the same key) for the decoding process. Using such a technique, two parties that share a pair of keys can easily communicate in a secure way across an insecure environment. A third party can not see the contents of the message if he does not have the decrypting key. The attackers can truncate, modify, reply, absolve, analyze the message, but they can not remove encryption.

1.4 Steganography vs. Cryptography

The term Steganography means “covert writing” whereas cryptography means “secret writing”. Cryptography is the study of methods of sending messages in such a form that only the intended recipient can remove the disguised message and read it. Encryption protects the contents of the message during transmission. Steganography hides messages in some other medium and does not require secret transmission. The message is then carried within some other data.

Cryptography techniques “scramble” the original message so if intercepted, the messages can not be understood by any unintended recipient. Steganography, in an essence, “camouflages” a message to hide its existence and makes it seem “invisible” thus concealing the fact that a message is being sent. An encrypted message may draw suspicion while an invisible message does not.

Steganography is not intended to replace cryptography rather it supplements it. The two techniques could be combined to provide a two tier protection to the secret communication. Hiding an encrypted message with a Steganographic method ensures that if the Steganographic message is detected then the interceptor also needs to decrypt that message.

1.5 Steganography vs. Digital Watermarking

Primarily these two techniques differ by the intent of use. A watermark can be perceived as an attribute of the carrier (cover). It may contain information such as copyright, license, tracking and authorship etc whereas in case of Steganography, the embedded message may have nothing to do with the cover. With Steganography the issue of concern is bandwidth for the hidden message whereas robustness is of more concern with watermarking [2].

Steganography is the act of adding a hidden message to an image or other media file. It is similar to encrypting a document, but instead of running it through a cipher, the document is broken up and stored in unused, or unnoticeable, bits within the overall image.

1.6 Category of Steganography

Steganography algorithms can be classified in several categories of increasing quality [15] [16]:

1. Adding data at the end of the carrier file.
2. Inserting data in some junk or comment field in the header of the file structure.
3. Embedding data in the carrier byte stream, in a linear, sequential and fixed way.
4. Embedding date in the carrier bytes stream, in a pseudo-random way depending on a password.
5. Embedding data in the carrier byte stream, in a pseudo-random way depending on a password, and changing other bits of the carrier file to compensate for the modifications induced by the hidden data, to avoid modifying statistical properties of the carrier file.

1.7 Need of Digital Steganography

The major purpose of Steganography is to hide the secret data that is normally encrypted to avoid interception. However, the main problem with encryption is that it transforms intelligible information to un-structured random data which is quite rare in today's computer world. From small TCP/IP packet to largest files on hard disks everything is stored and transmitted in strict hierarchical structure. So any random data suddenly appearing within a stream of structured data is likely to draw attention. The objective of using Steganography is just opposite. It aims at mixing the random data to decoy information so that it looks like structured format.

1.8 Structure of Steganography System

Steganography embeds a secret message in a cover message, this process is usually parameterized by a stego-key, and reading of embedded information is possible only having this key. Figure 7 shows a simple representation of the generic embedding and decoding process in Steganography.

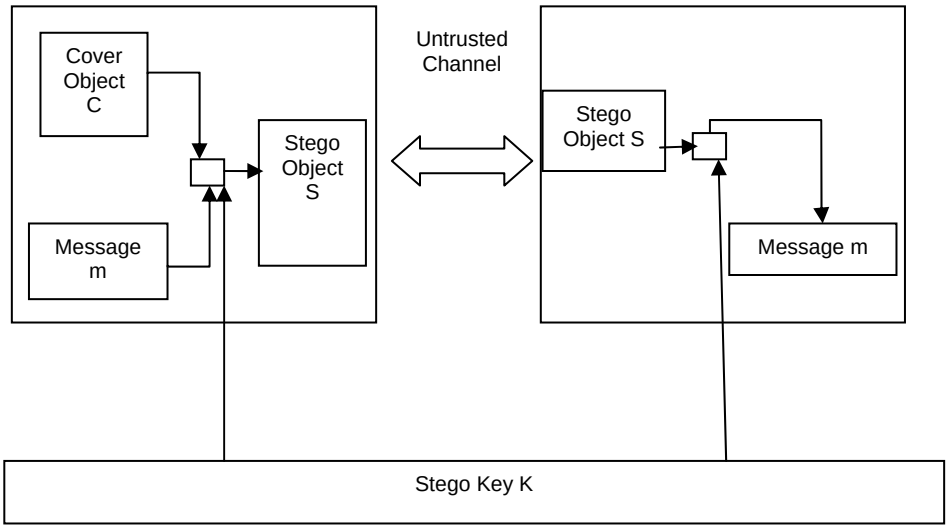


Figure 7: Overview of Steganography System

1.9 Requirements of Steganography

There are different requirements depending on the purpose of Steganography. The following is the list of main requirements that Steganography technique must satisfy.

1.9.1 Capacity

It is an important factor in applications, when a lot of information should be embedded into a cover image. For example, when transmitting medical images, the personal data and the diagnosis could be embedded into the same picture.

1.9.2 Imperceptibility

Imperceptibility is important when a secret communication occurs between two parties and the secret communication needs to be kept as a secret.

1.9.3 Robustness

Watermarking, Fingerprinting and all copyright protecting applications demand robust stenographic method, i.e. where the embedded information can not be removed without serious degradation of the image.

1.10 Limitation of Steganography

A major limitation of Steganography is the size of the medium being used to hide the data. In order for Steganography to be useful, the message should be embedded without any major changes to the object it is being embedded in. This leaves limited room for hiding information without noticeably changing the original object.

Steganography is constrained by the same assumption that exists for encryption. If Alice wants to send an image with a hidden message to Bob, she must first privately agree with Bob on a method of Steganography. Under the encryption model, Bob can be fairly sure when he's got some cipher text. However, in the Steganography model, it will be difficult for Bob to know when an image is just an image. Consider the scenario in which Alice borrows Bob's digital camera and neglects to tell him to pay special attention to every 73rd byte in the images she sends him. Because Bob is ignorant of Alice's Steganographic efforts, the large number of pictures he receives from her will only decrease the chance that Bob will let Alice borrow his digital camera again. The amount of data that can be effectively hidden in a given medium tends to be restricted by the size of the medium itself. The fewer constraints that exist on the integrity of the medium, the more potential it has for hiding data. For example, this paragraph is constrained by the rules of the English language and a specific topic of discussion. It would be difficult for me to hide a secret message in this paragraph due to the limited number of ways one can reasonably alter this text under those constraints.

1.11 Detecting Hidden Code

Steganalysis, the official countermeasure to Steganography, is the art of detecting and often decoding hidden data within a given medium. Two major tools in Steganalysis, information theory and statistical analysis, reveal in clear terms the tremendous potential for hidden information in Internet data — as long as a set of data can be compressed to a smaller size, there is room for hidden data within the medium. Accordingly, the path of seeking hidden data is untrustworthy and vague. Unless hidden data is encoded in a common, well-defined format, it may be virtually impossible to detect in the carrier data. In other words, a set bit can represent the any specific information, depending on how I choose to define my encoding. A Steganalysis expert is unable to determine the chosen encoding, for him a bit is just a bit.

Steganalysis, though of great interest to businesses and governments, has not received the attention it deserves. The ability to control sensitive information is a critical part of maintaining a large institution. Steganalysis research has been stimulated by the increasing number of tools available for digital Steganography. Steganalysis remains difficult to perform with great accuracy on some media. The growing field of cyber forensics — detective works in the digital domain — should create greater demand for Steganalysis tools in the near future.

Chapter 2
Different Methodologies of Steganography

In a digital world, Steganography and Cryptography both are used for providing security to secret information so that it could only be shared with the intended recipients. Both of these techniques could be used but none of them is completely secure. However, as most of the experts would suggest, it is better to use both in conjunction in order to provide multiple layers of security to the secret communication.

Multiple data formats are in use for Steganography these days such as .bmp, .doc, .gif, .jpeg, .mp3, .txt and .wav. These data formats are used for Steganography because they are widely used on the internet. These data formats provide the capability of embedding secret data by replacing the noise or redundant data. Keeping in view its use, Steganography promises a bright future in the world of digital communications and privacy on open systems such as the Internet. Steganographic research aims at removing the weaknesses of the cryptographic systems in order to ensure complete security. Steganography can play an important role by hiding the secret message within other files. This increases the security to a large extent as only the parties having secret communication can know about the existence of the secret message. To make the systems fool proof, multiple layers of security could be added by using cryptography and Steganography together.

In this chapter different types of Steganography is covered along with some of the other principles that are used in Steganography.

2.1 Detailed Look at Steganography

Let's first try to explain Steganography with the help of an example. Let's consider three friends named A, B and C. A wants to send a secret message (M) to B using a random (R) cover message

to create a cover (C) which can be sent to B without raising suspicion. A then changes the cover message (C) to a stego-object (S) by embedding the secret message (M) into the cover message (C) by using a stego-key (K). A then sends the stego-object (S) to B and C will not be able to detect the secret message. B could then read the hidden message (M) because he already knows the stego-key (K) used to embed the message into the cover message (C).

As Fabien A.P. Petit colas [15] points out, in a 'perfect' system, a normal cover should not be discernible from a stego-object, neither by a human nor by a computer if various statistical patterns are considered. However, practically this is not always the case. A secret message could only be embedded into the cover image if it contains considerable amount of noise or redundant data. This is because the Steganography actually replaces noise with the secret message. This limits the types of data that can be used with Steganography.

2.2 Types of Steganographic Protocols

There are three types of Steganographic protocols in use

Pure Steganography

Secret Key Steganography

Public Key Steganography

2.2.1 Pure Steganography

This type of Steganography is defined as a system that does not require the exchange of a cipher such as a stego-key. This technique is not as much secure as compared to rest of the techniques. Because in this case, the sender and receiver have to rely upon the presumption that no one else is aware of this secret message.

2.2.2 Secret Key Steganography

It is defined as a Steganographic system that requires the exchange of a secret key (stego-key) prior to communication. This technique takes the cover object and embeds the secret message by using a secret key (stego-key). Using this methodology ensures that only the parties who know the secret key can read the secret message. In contrast to Pure Steganography where a perceived imperceptible communication channel is present, Secret Key Steganography exchanges a stego-key, which makes it more prone to interception. But it is still secure in a sense that only the parties who know the secret key can extract the secret message.

2.2.3 Public Key Steganography

Public Key Steganography is defined as a Steganographic system that uses a public and a private key pair to protect the secret communication between the two parties. The sender will use the public key during the embedding process and only the private key, which is mathematically related to the public key, can decrypt the secret message. This method of Steganography provides a more robust way of implementing a Steganographic system because it has multiple levels of security. As a result the interceptor must first suspect the use of Steganography and then he would need a way to break the algorithm used by the public key system before they could actually read the hidden message.

2.3 Techniques in Use

The major techniques used for embedding secret information are discussed below:

2.3.1 Embedding Secret Messages in Text

Embedding secret messages in text files is a very difficult task. This is because text files don't have enough amount of redundant data to be substituted with secret message. Another drawback is that any third party can alter the hidden message by changing the text itself or reformatting the text to some other form (from .TXT to .PDF, etc.). There are a variety of methods available to achieve text based Steganography. A few of the methods are briefed below:

2.3.1.1 Line-shift encoding

As the name suggests, comprises of shifting each line of text vertically up or down by as little as 3 centimeters. A secret message could be embedded depending on whether the line was up or down from the stationary line [16].

In this method, the lines of the text are vertically shifted to some degree (for example, each line shifts (1/300 inches up or down) and information is hidden by creating a unique shape of the text. This method is proper for printed texts. However, in this method, the distances can be observed by using special instruments of distance assessment and necessary changes can be introduced to destroy the hidden information. Also if the text is retyped or if character recognition programs (OCR) are used, the hidden information would get destroyed.

2.3.1.2 Word-shift encoding

This method of encoding is more secure as it is less visible than line-shift encoding. However, an important requirement of this method is that the text being used supports variable spacing [15].

In this method, by shifting words horizontally and by changing distance between words, information is hidden within the text. This method is acceptable for texts where the distance between words is varying. This method can be identified less, because change of distance between words to fill a line is quite common. But if somebody was aware of the algorithm of distances, he can compare the present text with the algorithm and extract the hidden information by using the difference. The text image can be also closely studied to identify the changed distances. Although this method is very time consuming, there is a high probability of finding information hidden in the text.

2.3.1.3 Feature specific encoding

In this method, some of the features of the text are altered. For example, the end part of some characters such as h, d, b or so on, are elongated or shortened a little thereby hiding information in the text [13]. In this method, a large volume of information can be hidden in the text without making the reader aware of the existence of such information in the text. This is up till now the most secure encoding method. It is difficult to intercept as each type of formatted text has quite a lot of features that can be used for embedding the secret message. By placing characters in a fixed shape, the information is lost. Retyping the text or using OCR program (as in methods 1 and 2) destroys the hidden information.

2.3.1.4 Semantic Methods

In this method, synonyms for certain words are used thereby hiding information in the text [14]. A major advantage of this method is the protection of information in case of retyping or using OCR programs. However, this method may alter the meaning of the text.

2.3.1.5 Abbreviation

Another method for hiding information is the use of abbreviations. In this method, very little information can be hidden in the text. For example, only a few bits can be hidden in a file of several kilobytes [11].

2.3.1.6 Open Spaces

In this method, hiding information is done through adding extra white-spaces in the text. These whitespaces can be placed at the end of each line, at the end of each paragraph or between the words [12]. This method can be implemented on any arbitrary text and does not raise attention of the reader. However, the volume of information hidden under this method is very little. Also, some text editor programs automatically delete extra white-spaces and thus destroy the hidden information.

2.3.1.7 SMS-Texting

This method is based on the use of abbreviations for Steganography in SMS messages [22]. This method can be used on devices such as Pocket PC and PDAs. Also this method can be implemented on desktop PCs using SMS gateway for sending and receiving SMS messages. This technique of text Steganography can be developed and privatized in other areas as well. For example, the abbreviated equivalents of engineering terminology can be used in engineering texts.

In addition to English, SMS can be sent in other languages such as Arabic, Greek and so on. Therefore, the abbreviation text Steganography method in these languages can also be used for hiding information in SMS. A small amount of information can be hidden in this method. However, the size of input data can be decreased by compression before hiding the data. Cryptography of the intended data can also add the security of this method. Both of these operations need further computations which can not therefore be implemented on any type of mobile phone and if used, the number of mobile phones capable of executing this program will be decreased.

2.3.2 Encoding Secret Messages in Audio

Encoding secret messages in audio is the most challenging technique to use when dealing with Steganography. This is because the human auditory system (HAS) has such a dynamic range that it can listen over. To put this in perspective, the (HAS) perceives over a range of power greater than one million to one and a range of frequencies greater than one thousand to one making it extremely hard to add or remove data from the original data structure. The only weakness in the (HAS) comes at trying to differentiate sounds (loud sounds drown out quiet sounds) and this is what must be exploited to encode secret messages in audio without being detected.

There are two concepts to consider before choosing an encoding technique for audio. They are the digital format of the audio and the transmission medium of the audio.

2.3.2.1 Digital Audio Formats

There are three main digital audio formats namely Sample Quantization, Temporal Sampling Rate and Perceptual Sampling.

2.3.2.1.1 Sample Quantization

It is a 16-bit linear sampling architecture used by popular audio formats such as (.WAV and. AIFF).

2.3.2.1.2 Temporal Sampling

This method uses selectable frequencies (in the KHz) to sample the audio. Generally, the higher the sampling rate is, the higher the usable data space gets.

2.3.2.1.3 Perceptual Sampling

The last audio format is Perceptual Sampling. This format changes the statistics of the audio hugely by encoding only the parts the listener perceives, thus maintaining the sound but changing the signal. This format is used by the most popular digital audio on the Internet today in ISO MPEG (MP3).

2.3.2.2 Transmission medium

Path the audio takes from sender to receiver must also be considered when encoding secret messages in audio. W. Bender [15] introduces four possible transmission mediums:

a) **Digital end to end** - from machine to machine without modification.

b**) Increased/decreased resampling** - the sample rate is modified but remains digital.

c**) Analog and resampled** - signal is changed to analog and resampled at a different rate.

d) **Over the air** - signal is transmitted into radio frequencies and resampled from a microphone.

2.3.2.3 Popular encoding methods

The most popular encoding methods for hiding data inside of audio are low-bit encoding, phase-coding and spread spectrum.

2.3.2.3.1 Low-bit encoding

Low-bit encoding embeds secret data into the least significant bit (LSB) of the audio file. The channel capacity is 1KB per second per kilohertz (44 kbps for a 44 KHz sampled sequence). This method is easy to incorporate but is very susceptible to data loss due to channel noise and resampling.

2.3.2.3.2 Phase coding

This technique substitutes the phase of an initial audio segment with a reference phase that represents the hidden data. This can be thought of, as sort of an encryption for the audio signal by using what is known as Discrete Fourier Transform (DFT), which is nothing more than a transformation algorithm for the audio signal.

2.3.2.3.3 Spread spectrum

This method encodes the audio over almost the entire frequency spectrum. It then transmits the audio over different frequencies which will vary depending on what spread spectrum method is used. Direct Sequence Spread Spectrum (DSSS) is one such method that spreads the signal by multiplying the source signal by some pseudo random sequence.known as a (CHIP). The sampling rate is then used as the chip rate for the audio signal communication.

Spread spectrum encoding techniques are the most secure means by which to send hidden messages in audio, but it can introduce random noise to the audio thus creating the chance of data loss.

2.3.3 Embedding Secret Messages in Images

Embedding secret messages in digital images is the most extensive form of Steganography in practice these days. The main reason is that this technique utilizes the limited power of human visual system. This method has the capability of embedding any type of data that could be converted into a bit stream. This could include cipher text, image and text image. With the research being put into image based Steganography, this field is growing at a very fast pace.

Before describing different techniques, a brief explanation of digital image architecture and digital image compression techniques is discussed.

As Duncan Sellers [9] explains; "To a computer, an image is an array of numbers that represent light intensities at various points, or pixels. These pixels make up the images raster data." When dealing with digital images 8-bit and 24-bit per pixel image files are typical. Both are discussed below 8-bit images are relatively small in size. The only short coming of these image files is that they can only have 256 possible colors. This can create problems during the embedding process. For 8-bit images normally gray scale color palette is used such as (.GIF) because its slow change in gradient would be hardly susceptible after the secret message has been embedded. 24-bit images are better suited for Steganography. 24 bit images can have a very wide range (over 16 million) colors which makes the secret message hard to detect after the secret message has been embedded. The other advantage is that these 24 bit images have a larger capacity to embed a secret message. The one major disadvantage is their large size (usually in MB), which makes these images more suspicious than 8-bit digital images (usually in KB) when transmitted over an open system such as the Internet.

2.3.3.1 Major Categories of Image Steganography

The major categories include

1. Least Significant Bit Steganography
2. Transform Domain Steganography
3. Spread Spectrum Steganography
4. Steganography using Fractals

2.3.3.1.1 Least significant bit (LSB) encoding

This is by far the most popular form of coding technique used for digital images. In this method last bit of every byte is altered to embed one bit of the secret message. Using this technique, 3 bits can be stored in a single pixel from a 24-bit image and one bit in each pixel of an 8-bit image. This is why a24-bit image has greater capacity to hide a message as compared to an 8-bit image. Similarly if a gray scale image is used then two least significant bits can be used to store bits of the secret message [10]. The changes made to the cover image will not be detectable by the human visual system. The only drawback of this system is that it vulnerable to changes in image formats like changing from GIF to JPEG.

To hide information in the LSBs of each byte of a 24-bit image, it is possible to store 3 bits in each pixel.

1 pixel:

(00100111 11101001 11001000)

Insert 101:

(00100111 11101000 11001001)
red green blue

Figure 8: A simplified example with a 24-bit image

LSB insertion works well with gray-scale images as well. It is possible to hide data in the least and second least significant bits and the human eye would still not be able to discern it.

Unfortunately LSB insertion is vulnerable to slight image manipulation such as cropping and compression. For example, converting a GIF or a BMP image, which reconstructs the original message exactly (lossless compression), to a JPEG format, which does not (lossy compression), and then converting back, can destroy the data in the LSBs.

2.3.3.1.2 Transform based Steganography

Another widely used method of Steganography is hiding data in the Fourier domain of an image using the Discrete Cosine Transform (DCT), which allows the data to be more resistant to becoming corrupted [17]. The DCT method is widely used over simple Steganography methods because hiding data in the Least Significant Bit (LSB) is more easily detectable and data can be lost due to encoding schemes like JPEG. With the DCT method, data can be hidden in a robust format so that the data can be recovered even if some of the image data is lost or modified [22]. Steganography using the DCT is usually accomplished by modulating the size of two or more DCT coefficients within a block of part of an image [19]. The DFT method is similar to the DCT by taking frequencies in the mid-band region, and modifying their coefficients [20]. Images are padded with zeros before taking Discrete Fourier Transform (DFT) to avoid aliasing [21].

2.3.3.1.3 Spread Spectrum Steganography

This system hides and recovers messages of substantial length within digital imagery while maintaining the original image size and dynamic range. The hidden messages can be recovered using appropriate keys without any knowledge of the original image. Image processing, error control coding, and spread-spectrum techniques utilized are described, and the performance of the technique is illustrated. A message embedded by this method can be in the form of text, imagery, or any other digital signal.

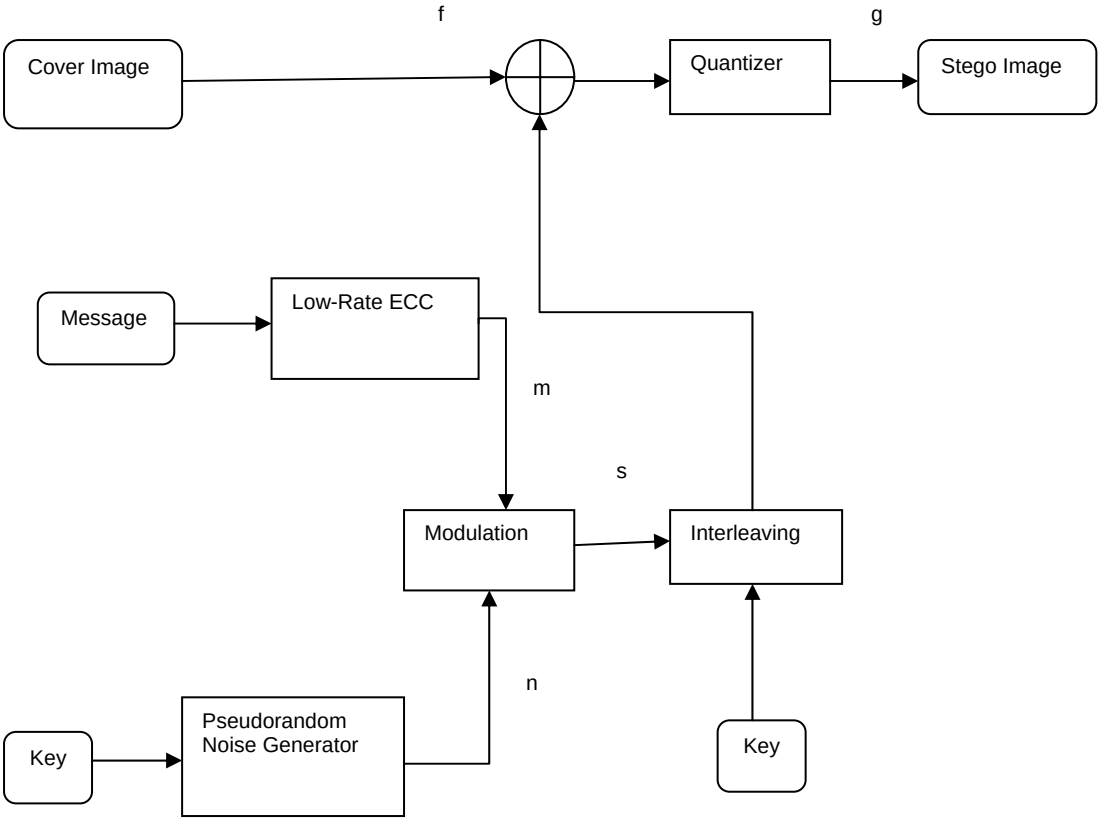


Figure 9: SSIS Encoder

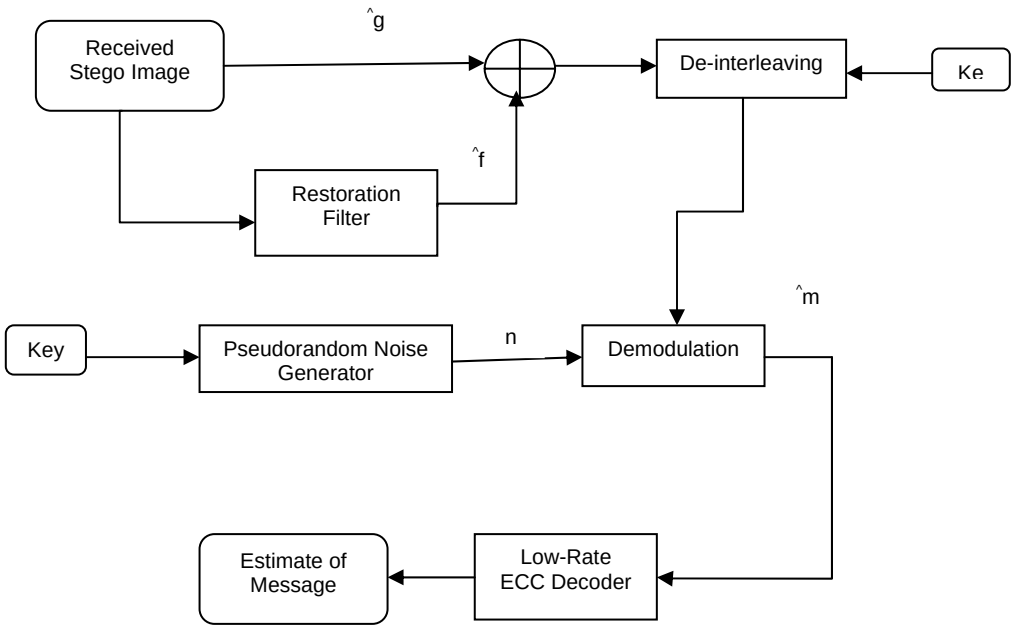


Figure 10: SSIS Decoder

Techniques of spread spectrum communication, error-control coding, and image processing are combined to accomplish SSIS. The SSIS encoder and decoder are shown in Figures 10 and 11. The fundamental concept of SSIS is the embedding of the hidden information within noise, which is then added to the digital image [3]. This noise is typical of that inherent to the image acquisition process and, if kept at low levels, is not perceptible to the human eye nor is it susceptible to detection by computer analysis without access to the original image.

2.3.3.2 Digital image compression

Compression can provide a better alternative for large images such as 24-bit images. There are two types of compression techniques in use namely lossy and lossless.

a) Lossy compression

Lossy compression such as (.JPEG) compresses the size of a digital image by removing redundant image data and creates a very close approximation to the image. This form of compression is normally used with large digital images as mentioned earlier. It also has some

disadvantages as it increases the probability that the secret message will lose parts because lossy compression tends to remove any excess data.

b) **Lossless compression**

These techniques, as the name suggests, does not remove any excess data from the image thus keeps the original digital image intact. It is for this reason that it is widely used in Steganography. Examples of lossless compression techniques are (.GIF and .BMP). The only disadvantage of this method is that it does not compress the size of the image well.

2.3.3.3 Masking and filtering techniques

These can also be used for Steganography such a watermarking. (i.e. Integrating a company's logo on there content). These methods are mostly used with lossy compression techniques such as (.JPEG). This technique actually extends an image data by masking the secret data over the original data as opposed to hiding information inside of the data. The major advantage of these techniques is that they are impervious to image manipulation.

2.3.4 Grey Scale Image Steganography

2.3.4.1 Labelling Method

Motameni, Norouzi, Jahandar, and A. Hatami in their paper "Labeling Method in Steganography" [23] propose a method of hiding information in gray scale images using labeling. The method first finds the binary value for each character in text and then finds dark places in the gray image by converting input image to binary image on 8-connectivity basis. Then the image is converted to RGB image in order to find the dark places. In this way, each sequence of gray color turns into RGB color. If the Gary image is very light, the histogram must be changed manually to find just dark places. In the final stage each 8 pixels of dark places are considered as a byte and binary value of each character is put in low bit of each byte that was created manually by dark places pixels for increasing security of the main way of Steganography (LSB).

2.3.4.2 Hiding Color image in a Gray Scale Image

Chaumont and W. Puech propose a method to embed the color information of an image in a corresponding grey-level image [24]. The objective of this work is to allow free access to the

grey-level image and give color image access only if you own a secret key. This method is made of three major steps which are the color quantization, the ordering and the data hiding.

2.3.5 Color Image Steganography

2.3.5.1 Using Edge Detection

In this method, edge pixels are selected in order to hide the data. One common way to hide the data is "Least Significant Bit Insertion". This method modifies the low order bit of each pixel to match the message to hide. The selection of pixels in which the message will be embedded is very important because modified pixels in areas of the image where there are pixels that are most like their neighbors are much more noticeable to the naked eye. A single modified pixel stands out among its uniform neighbor pixels thus making the image suspicious. One possible solution for this problem is to select the edge-pixels of the image to hide the message [25]. It is not noticeable when a single pixel is modified when its surrounding pixels are least like it.

2.3.5.2 Using Variable-Bits

This paper presents an algorithm for storing variable number of bits in each channel (R, G or B) of pixel based on the actual color values of that pixel: lower color component stores higher number of bits [26]. The procedure followed is as follows

1. One of the Red, Green and Blue channels is used as an indicator channel. This has to be agreed upon by the sender and receiver prior to communication.
2. Data is stored in either of the other two channels depending on their intensity values. The channel that has the lowest color value will embed data in LSBs.
3. To retrieve data, the receiver analyzes the least significant bits of the other two channels. If the last two bits are same then the channel following the indicator channel stores the data. Otherwise the channel preceding the indicator channel contains the embedded data.

2.3.5.3 Using differential phase-shift keying techniques

Here following steps are used to achieve Steganography

(1) Compress the secret image to reduce the number of secret bits. The set partitioning in hierarchical trees (SPIHT) codec were used to obtain a high reconstructed image quality and low bit rate image compression.

(2) A neighbor block signal phase comparison (NBSPC) mechanism is used to offer the location for secret data embedding.
(3) A fold phase distribution differential phase-shift keying FPDPSK mechanism is used to improve the quality of the cover image [27]

2.3.5.4 Using best-block matching and k-means clustering

An image-hiding technique using block-matching procedure is proposed in [28]. The method suggests to

1. Split the entire image into multiple non-overlapping blocks.
2. For each block, best matching block is searched from a series of numbered candidate blocks [generated from the cover image.]
3. The blocks that are not well matched, a *k*-means clustering method is applied to determine some representative blocks.
4. These representative blocks are used to replace some of the candidate blocks that were not referenced in the block matching step.
5. Finally, the obtained indices of blocks are encoded using Huffman coding scheme, and then recorded in the LSB Channels of the cover image.

Chapter 3
System Design

The method first compresses the input text to increase the data hiding capacity of any image as suggested by Wen-Yuan Chen in [27]. There are various data compression algorithms available but the technique chosen here is LZW.

3.1 LZW Coding

LZW coding is used in conjunction with this method to see if it can further increase the capacity of the stego-image to embed the secret message. It intends to increase the capacity and efficiency of this block difference method used for converted communication.

At the start of the coding process, a dictionary is constructed that includes the code symbols to be encoded. The code words that are not part of the dictionary are placed in next free unused location. This new location is determined according to LZW algorithm. Once the code sequence is added to the dictionary, next time if the same sequence is encountered, the algorithm uses the location of that sequence to represent that sequence. Here, the size of the dictionary is an important determinant; if the dictionary is small then it's less likely to detect the sequence and if the dictionary is large it might make compression ineffective.

Let's consider the following example.

40	40	200	200
40	40	200	200
40	40	200	200
40	40	200	200

Table 1: 8-bit image sample

For this, we consider a mono-chrome, 8 bit image with 256 gray values. In order to perform LZW compression, an initial 512 word dictionary is created.

Dictionary Location	Entry
0	0
1	1
255	255
.	.
.	.
.	.
511	.

Table 2: Dictionary prior to compression

Here, in this example the matrix is traversed in left to right and top to bottom order as shown below

Currently recognized Sequence	Pixel being processed	Encoded Output	Dictionary Location	Dictionary Entry
	40			
40	40	40	256	40-40
40	200	40	257	40-200
200	200	200	258	200-200
200	40	200	259	200-40
40	40			
40-40	200	256	260	40-40-200
200	200			
200-200	40	258	261	200-200-40
40	40			
40-40	200			
40-40-200	200	260	262	40-40-200-200
200	40			
200-40	40	259	263	200-40-40
40	200			
40-200	200	257	264	40-200-200
200		200		

Table 3: LZW Example

Currently recognized sequence variable is used to hold the concatenated sequence. Whenever a new value is encountered; the dictionary is searched for the concatenated sequence. If the sequence is found, no output code is generated. If it isn't found, the location of the currently recognized sequence becomes the next encoded output and the new sequence is added to the dictionary. In the example shown above, additional code words are generated whereas the complete dictionary comprises of 256 code words and as can be seen this method could spot several repeating gray levels.

3.2 Block Difference Method

The method adopted in this thesis works on the basis of block differences as suggested by ZanWang and Yao-De Tsai [28]. First of all the whole image is divided into non-overlapping three pixel blocks. Then the difference between the first and second pixel and second and third pixel is calculated. This is called absolute difference. The valid values for the image pixels are 0-255. Let g (d) be the number of pixels with absolute difference d. The value of d could be between 0 and 253 i.e. $0 \leq d \geq 253$. The blocks containing pixels with values 0 and 256 are not considered during the embedding process. The proposed scheme then selects a pair of maximum and minimum differences such that $g(M) \geq g(m)$.The Maximum and Minimum difference is calculated and stored for future reference. The Maximum difference is calculated such that $g(M) > g(M`)$ where M` stands for all the differences ≤ 0. Similarly, the Minimum difference is calculated such that $g(m) < g(m`)$ for all $m` \leq 253$.

Let (b_{i0},b_{i1},b_{i2}) denote a block i with pixel values equal to b_{i0}, b_{i1}, and b_{i2}, and $\max(b_{i0},b_{i1},b_{i2})$ and $\min(b_{i0},b_{i1},b_{i2})$ denote the maximum and minimum pixel values in the block, respectively. A block is selected if it fulfills the following two conditions

(1) $1 \leq b_{i0},b_{i1},b_{i2} \leq 254$;

(2) $\min (b_{i0},b_{i1},b_{i2}) = 1$ or $\max (b_{i0},b_{i1},b_{i2}) = 254$.

For all the selected blocks following actions are performed

(1) Increase d_{i0} by 1 if $M + 1 \leq d_{i0} \leq m - 1$, and increase d_{i0} by 1 if $M + 1 \leq d_{i0} \leq m - 1$, where $d_{i0} = |b_{i0} - b_{i1}|$ and $d_{i1}=|b_{i1}-b_{i2}|$;

(2) Embed a message into block i if $d_{i0}=M$ or $d_{i1}=M$;

(3) Note the index of block i as overhead information if $\min (b_{i0},b_{i1},b_{i2}) = 0$ or $\max (b_{i0},b_{i1},b_{i2}) = 255$.

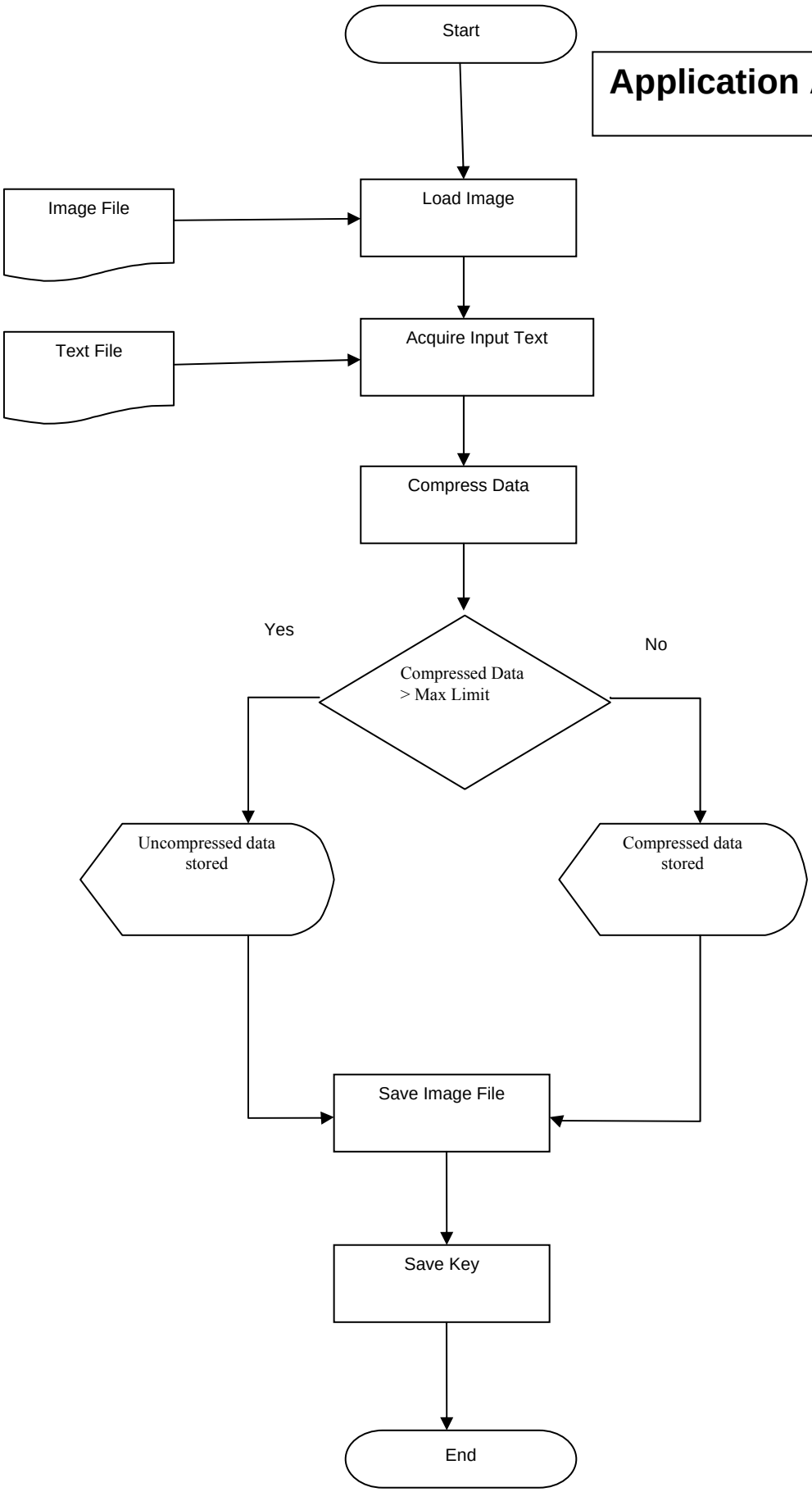
Following actions are performed on each selected block i that has $2 \leq (b_{i0},b_{i1},b_{i2}) \leq 253$

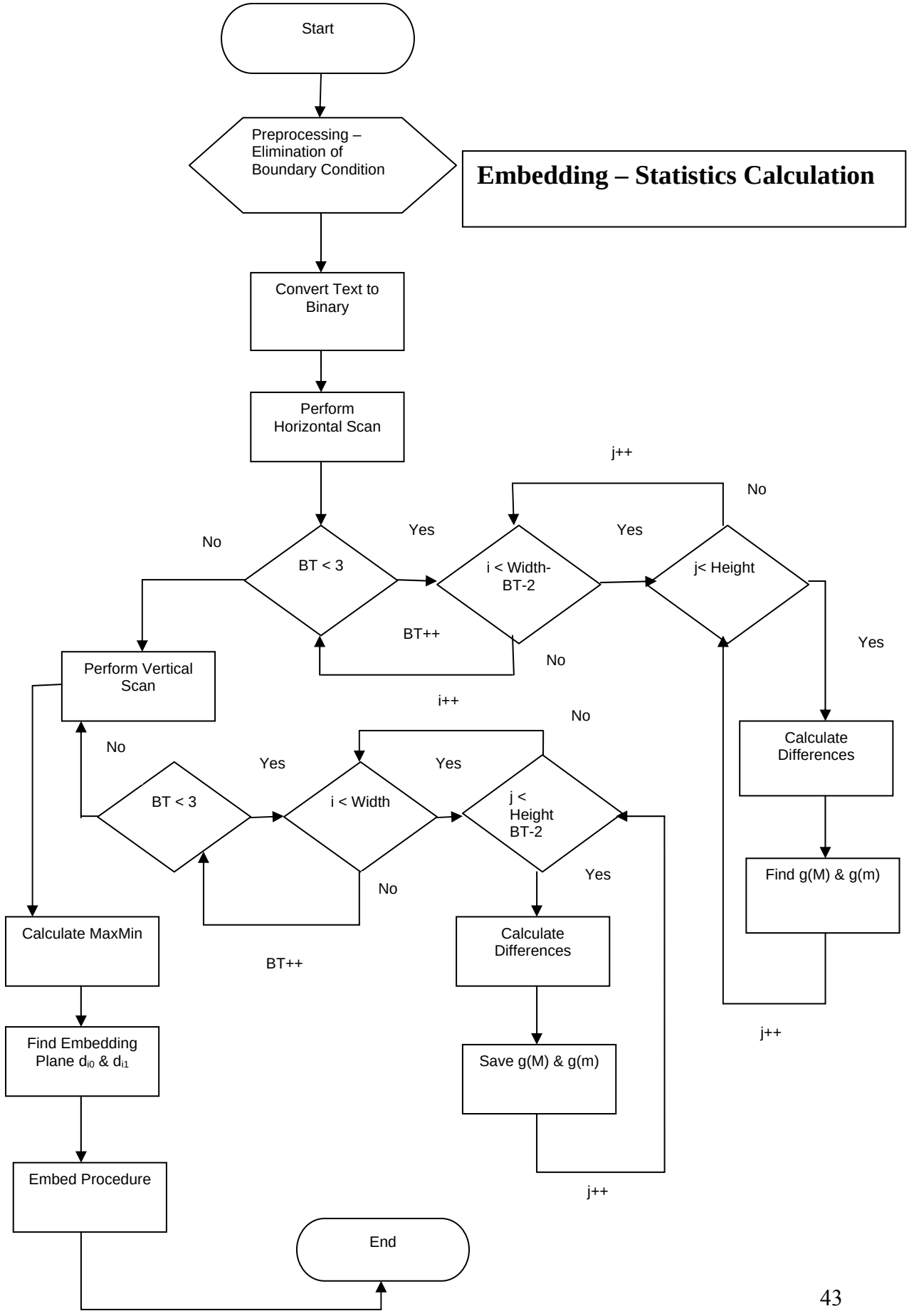
(1) increase d_{i0} by 1 if $M+1 \leq d_{i0} \leq m-1$, and increase d_{i1} by 1 if $M + 1 \leq d_{i1} \leq m - 1$;

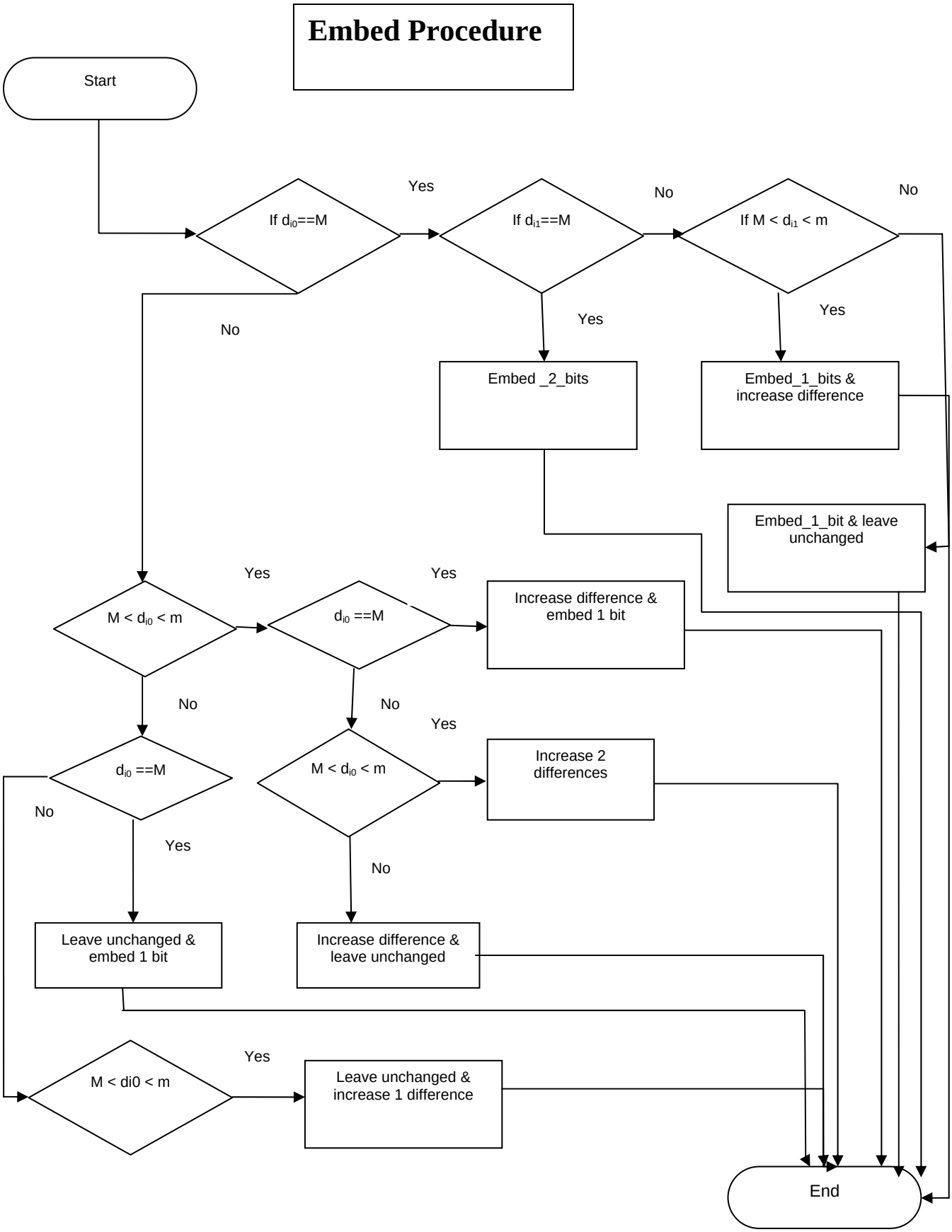
(2) Embed the overhead information and the residual message into block i if $d_{i0} = M$ or $d_{i1} = M$.

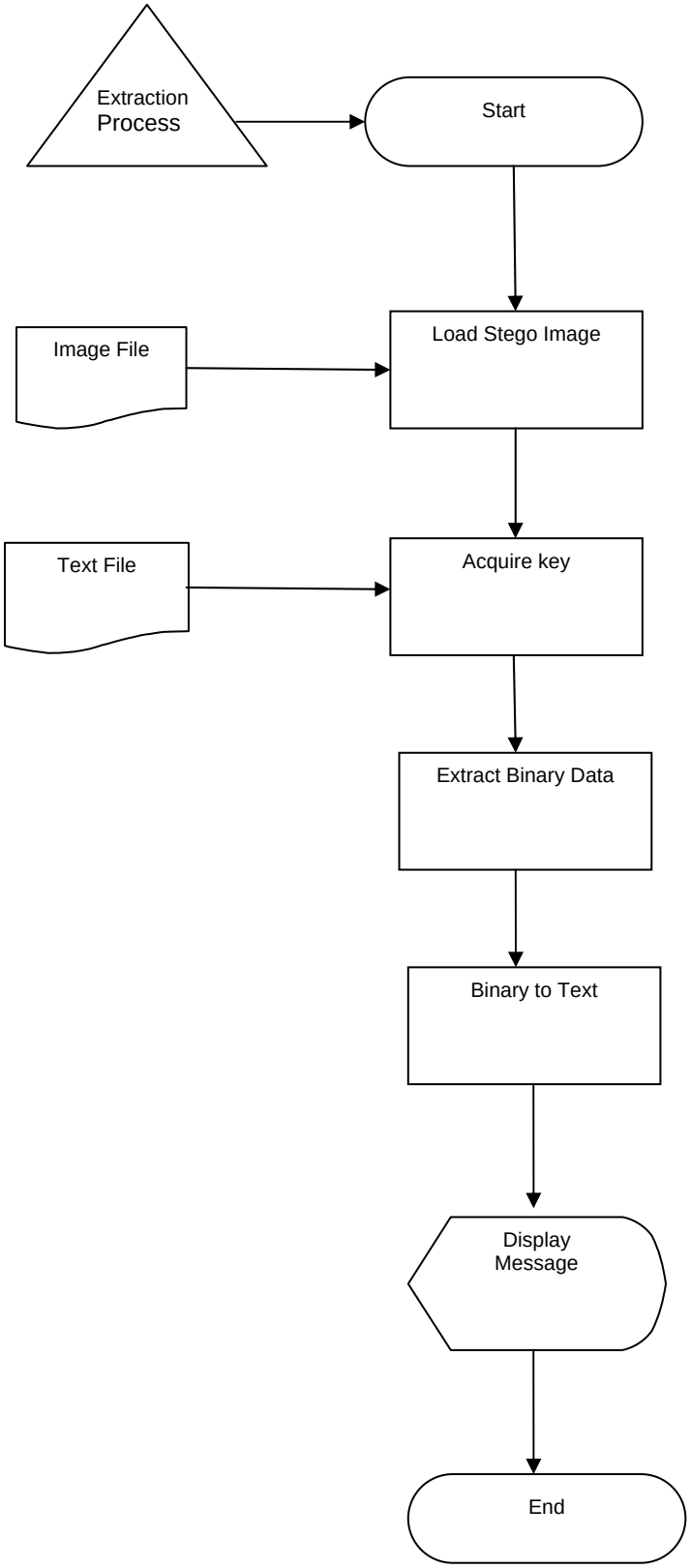
3.3FLOW CHARTS

Application Architecture

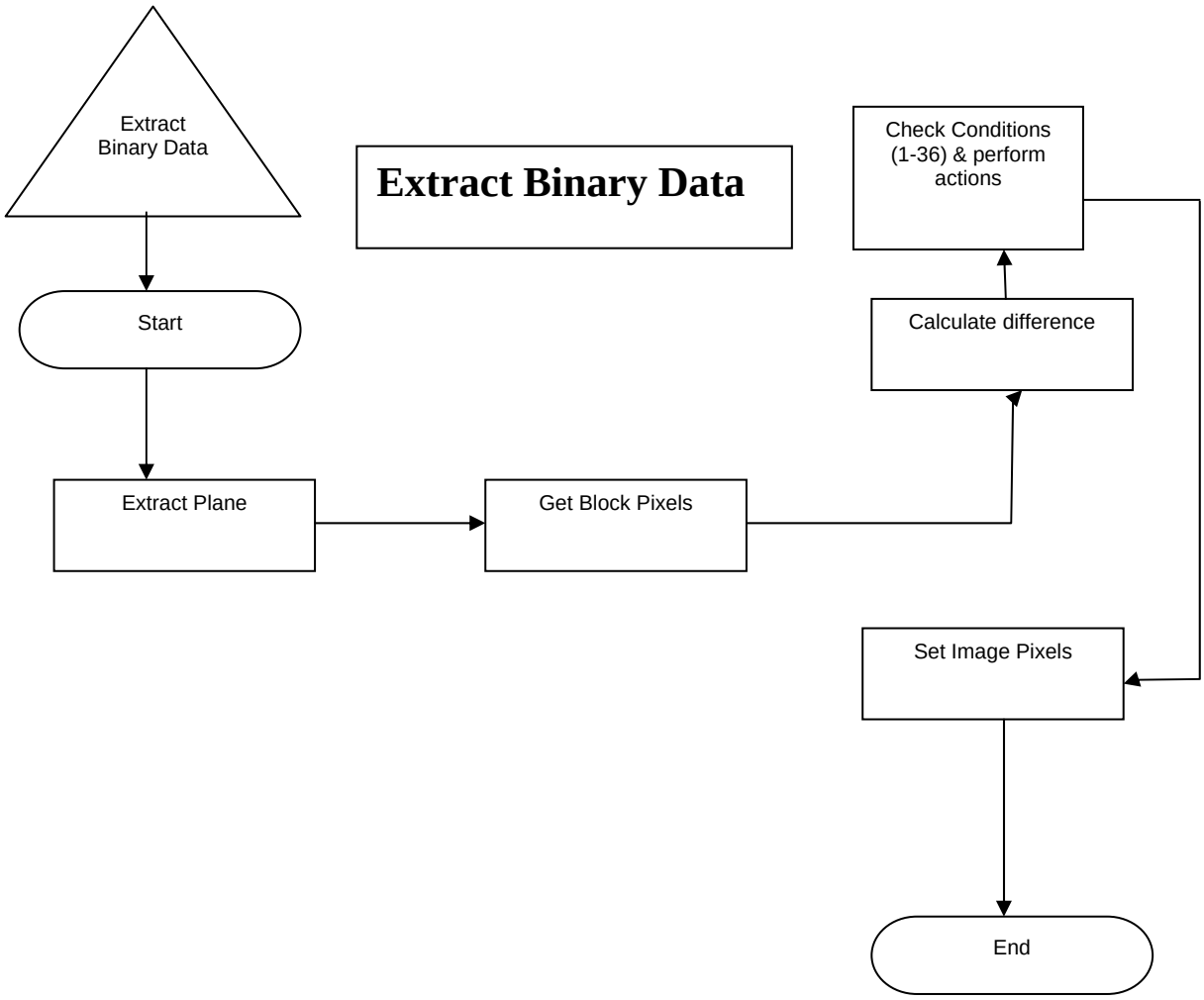



Start
Preprocessing – Elimination of Boundary Condition
Embedding – Statistics Calculation
Convert Text to Binary
Perform Horizontal Scan
j++
No
Yes
Yes
No
BT < 3
i < Width-BT-2
j< Height
BT++
No
Yes
Perform Vertical Scan
i++
No
Calculate Differences
No
Yes
Yes
BT < 3
i < Width
j < Height BT-2
Find g(M) & g(m)
Yes
No
Calculate MaxMin
BT++
Calculate Differences
j++
Find Embedding Plane d_{i0} & d_{i1}
Save g(M) & g(m)
Embed Procedure
j++
End


Embed Procedure
Start
If d_{i0}==M
Yes
If d_{i1}==M
No
If M < d_{i1} < m
No
Yes
No
Yes
Embed _2_bits
Embed_1_bits & increase difference
Embed_1_bit & leave unchanged
M < d_{i0} < m
Yes
d_{i0} ==M
Yes
Increase difference & embed 1 bit
No
No
d_{i0} ==M
M < d_{i0} < m
Yes
Increase 2 differences
No
Yes
No
Leave unchanged & embed 1 bit
Increase difference & leave unchanged
M < di0 < m
Yes
Leave unchanged & increase 1 difference
End



Extraction Process


Extract
Binary Data
Extract Binary Data
Check Conditions
(1-36) & perform
actions
Start
Calculate difference
Extract Plane
Get Block Pixels
Set Image Pixels
End

3.4 Embedding Process - Algorithm

1) Partition the image I into non-overlapping horizontal Type-t blocks where t=0 to 2

(2) For each block i with $1 \leq b_{i0}, b_{i1}, b_{i2} \leq 254$, calculate $d_{i0} = | b_{i0} - b_{i1} |$ and $d_{i1} = | b_{i1} - b_{i2} |$. Then, calculate $g(d)$, the number of pixel pairs with absolute difference equal to d, where $0 \leq d \leq 253$. Ignore the blocks containing pixel values equal to 0 or 255.

(3) Select the maximum value of $g(M)$ and the minimum value of $g(m)$, Make sure M < m. where M and m are the differences at which $g(M)$ and $g(m)$ are the maximum and the minimum, If more than one pair of M and m is found, the pair with the smallest value of $|M - m|$ is selected, Note that $0 \leq M$ and $m \leq 253$.

(4) For t =0 to 2, divide image I into non-overlapping vertical Type-t (3×1) blocks and do steps 2 and 3.

(5) To embed the message choose such a block orientation (BO, horizontal or vertical) and block type (BT, Type-0, Type-1, or Type-2) that has the largest $g(M)$as calculated in step 3. Record the following as overhead information

- block orientation (horizontal or vertical)
- Block type (Type-0, Type-1, or Type-2)
- M
- m

These four things (called the Key of Four) forms an *embedding Channel that* can provide a payload capacity of $g(M)$ bits.

(6) Index the blocks in step 5 from 0 onwards. Then scan image I in the indexed order. Let $ga(M)$ be the number of pixel pairs with difference equal to M and the blocks to which they belong satisfy the following two conditions:

(1) $1 \leq b_{i0}, b_{i1}, b_{i2} \leq 254$;

(2) $\max(b_{i0}, b_{i1}, b_{i2}) = 254$ or $\min(b_{i0}, b_{i1}, b_{i2}) = 1$.

For each block i satisfying the above conditions (1) and (2), call the procedure *embedding_procedure* listed in Fig. 12 to embed the message. After invoking the embedding

procedure, if max(b_{i0}, b_{i1}, b_{i2}) = 255 or min(b_{i0}, b_{i1}, b_{i2}) = 0, record block i as an FB in the overhead information.

(7) If $g(M) - ga(M) <$ the size of the overhead information to be embedded, abort the embedding process. This means that the image does not have appropriate capacity to embed the given message. If $g(M) - ga(M) \geq$ the size of both the overhead information and the remaining message, mark this as a last embedding Channel. Scan image I again in the order as that in step 6. For each block i with $2 \leq bi0, bi1, bi2 \leq 253$, call the procedure *embedding_procedure* listed in Fig. 12 to embed the overhead information illustrated in Fig. 11, and then the residual message.

(a)

#FBs	#IFBs

(b)

#FBs	#IFBs	BO	BT	M	m

(c)

#FBs	#IFBs	BO	BT	M	M	#EPs

(a) first embedding plane; (b) middle embedding plane; (c) last embedding plane

Figure 11: Overhead Information in embedding plane

(8) Go to step 1 until the message is completely embedded.

(9) Send the stego-image and the Key of Four of the last embedding Channel to the receiver.

The last embedding Channel containing the overhead information is actually the key to extract the information. This information can be separately sent to the receiver or can be alternatively stored within the same image. This could be negotiated with the receiver and according to the agreement, the information could be stored either in the first few pixels by changing the values of the Least Significant bits of the first few pixels if the image. Here we store this information in a separate file to be sent to the receiver. The receiver would need this information to restore the original message.

3.4.1 Embedding_procedure:

```
if di0 == M
{
        if di1 == M
                call embed_2_bits;
        else
        {
                if M < di1 < m
                        call embed_1_bit_and_increase_difference;
                else
                        call embed_1_bit_and_leave_unchanged;
        }
}
else if M < di0 < m
{
        if di1 == M
                call increase_difference_and_embed_1_bit;
        else
        {
                if M < di1 < m
                        call increase_2_differences;
                else
                        call increase_difference_and_leave_unchanged;
        }
}
else
{
        if di1 == M
                call leave_unchanged_and_embed_1_bit;
        else
        {
                if M < di1 < m
                        call leave_unchanged_and_increase_difference;
                else
                        do nothing;
        }
}
```

Figure 12: Embedding procedure

i. Procedure embed_2_bits

Conditions	Actions
$b_{i0} > b_{i1} > b_{i2}$	$b_{i0} = b_{i0} + eb_1$, $b_{i2} = b_{i2}\ eb_2$
$b_{i0} < b_{i1} < b_{i2}$	$b_{i0} = b_{i0}\ eb_1$, $b_{i2} = b_{i2} + eb_2$
$b_{i0} = = b_{i1} = = b_{i2}$	if $eb_1 == 1$ and $eb_2 == 1$ $b_{i1} = b_{i1} + 1$ else $b_{i0} = b_{i0}\ eb_1$, $b_{i2} = b_{i2} + eb_2$
$b_{i0} < b_{i1} > b_{i2}$	if $eb_1 == 1$ and $eb_2 == 1$ $b_{i1} = b_{i1} + 1$ else $b_{i0} = b_{i0}\ eb_1$, $b_{i2} = b_{i2}\ eb_2$
$b_{i0} > b_{i1} < b_{i2}$	if $eb_1 == 1$ and $eb_2 == 1$ $b_{i1} = b_{i1}$- 1 else $b_{i0} = b_{i0} + eb_1$, $b_{i2} = b_{i2} + eb_2$

ii. Procedure embed_1_bit_and_increase_difference:

Conditions	Actions
$b_{i0} > b_{i1} > b_{i2}$	$b_{i0} = b_{i0} + eb_1$, $b_{i2} = b_{i2}$ -1
$b_{i0} < b_{i1} < b_{i2}$	$b_{i0} = b_{i0}\ eb_1$, $b_{i2} = b_{i2} + 1$
$b_{i0} \leq b_{i1} > b_{i2}$	if $eb_1 == 1$ $b_{i1} = b_{i1} + 1$ else $b_{i2} = b_{i2}$- 1
$b_{i0} \geq b_{i1} < b_{i2}$	if $eb_1 == 1$ $b_{i1} = b_{i1}$ -1 else $b_{i2} = b_{i2} + 1$

iii. Procedure embed_1_bit_and_leave_unchanged

Conditions	Actions
$b_{i0} > b_{i1}$	$b_{i0} = b_{i0} + eb_1$
$b_{i0} \leq b_{i1}$	$b_{i0} = b_{i0} - eb_1$

iv. Procedure increase_difference_and_embed_1_bit

Conditions	Actions
$b_{i0} > b_{i1} > b_{i2}$	$b_{i0} = b_{i0} + 1$, $b_{i2} = b_{i2} - eb_1$
$b_{i0} < b_{i1} < b_{i2}$	$b_{i0} = b_{i0} - 1$, $b_{i2} = b_{i2} + eb_1$
$b_{i0} < b_{i1} \geq b_{i2}$	if $eb_1 == 1$ $b_{i1} = b_{i1} + 1$ else $b_{i0} = b_{i0} - 1$
$b_{i0} > b_{i1} \leq b_{i2}$	if $eb_1 == 1$ $b_{i1} = b_{i1} - 1$ else $b_{i0} = b_{i0} + 1$

v. Procedure increase_2_differences

Conditions	Actions
$b_{i0} > b_{i1} > b_{i2}$	$b_{i0} = b_{i0} + 1$, $b_{i2} = b_{i2} - 1$
$b_{i0} < b_{i1} < b_{i2}$	$b_{i0} = b_{i0} - 1$, $b_{i2} = b_{i2} + 1$
$b_{i0} < b_{i1} > b_{i2}$	$b_{i1} = b_{i1} + 1$
$b_{i0} > b_{i1} < b_{i2}$	$b_{i1} = b_{i1} - 1$

vi. Procedure increase_difference_and_leave_unchanged

Conditions	Actions
$b_{i0} > b_{i1}$	$b_{i0} = b_{i0} + 1$
$b_{i1} < b_{i2}$	$b_{i0} = b_{i0} - 1$

vii. Procedure leave_unchanged_and_embed_1_bit

Conditions	Actions
$b_{i1} > b_{i2}$	$b_{i2} = b_{i2} - eb_1$
$b_{i1} < b_{i2}$	$b_{i2} = b_{i2} + eb_1$

viii. Procedure leave_unchanged_and_increase_difference

Conditions	Actions
$b_{i1} > b_{i2}$	$b_{i2} = b_{i2} - 1$
$b_{i1} < b_{i2}$	$b_{i2} = b_{i2} + 1$

The procedures invoked in Figure 12 are listed above. Please note that in all the above tables two consecutive equal signs "==" stand for equality and the single equal sign " = " denotes an assignment. Also, note that eb_1 and eb_2 represent the next two (message bits/overhead information) bits to be embedded. For example, in the procedure *embed_2_bits*, if $b_{i0} > b_{i1} > b_{i2}$ then eb_1 will be added to b_{i0} and eb_2 will be subtracted from b_{i2}.

In addition, only the differences that satisfy $M<m$ *condition are considered*. If $m<M$ and $|M-m|$ is the smallest in this step, the differences satisfying the conditions $m<d_{i0}<M$ or $m<d_{i1}<M$ will be decreased by 1, and the differences satisfying the conditions $m \leq di0<M$ or $m \leq di1<M$ will be increased by 1 during the extraction process.

Similarly, when embedding a bit "1" into block i, the differences satisfying the conditions $di0 = M$ or $di1 = M$ will be decreased by 1, instead of being increased by 1 as shown in Fig. 12. Moreover, the differences satisfying the condition $di0 = M - 1$ or $di1 = M - 1$ will be increased by 1 after a bit "1" is extracted.

The Key of Four of the last embedding Channel is the key to extract the embedded message and restore the original cover image. It can be separately sent to the receiver or embedded into the LSBs of the pixels (for example, the first several pixels), by simple LSB replacement, in the stego-image negotiated in advance by both the sender and the receiver. If the latter is used, an embedding space of 19 bits (1/2/8/8 bits for BO/BT/*M*/*m*, respectively are needed to save the Key of Four of the last embedding Channel. If one pixel embeds one bit, the last embedding Channel should not include the 19 pixels used to embed the Key of Four. In addition, the 19 LSBs to be replaced by the Key of Four should be saved in the overhead information so that the cover image can be restored. For simplicity we assume that the Key of Four has been separately exchanged between the sender and the receiver.

3.5 Overhead information formats

Lets consider a grayscale image *I* of i rows × j columns, this image can be divided into maximum of a=max*(i*×[*j*/3],*j* ×[i/3]*)* non-overlapping three-pixel blocks. In this situation, at least B = [log2a] bits are required to determine an index of FB (IFB). In addition, the number of embedding Channels (#EPs), *M*, and *m* should be less than 256 for a grayscale image in which each pixel takes up eight bits, since we cannot increase or decrease a pixel value more than 255 times and the block difference is less than 256. We require 1 and 2 bits to save Block Orientation [BO} and Black Type [BT], respectively, as BO could have only two values i.e. 0 (horizontal) or 1 (vertical) and BT can have three values (0/1/2 for Type-0/Type-1/Type-2, respectively). Therefore, the Key of Four of an embedding Channel requires 19 bits collectively.

Every embedding Channel stores the number of FBs and Ifbs for future reference. The number of embedding Channels is stored in the last embedding Channel. But here we are sending this information separately.

3.6 Extraction process

Given M and m, for each block *i* with $1 \leq b_{i0},b_{i1},b_{i2} \leq 254$, the receiver performs the following actions:

1. Extract the overhead information or a message if $d_{i0} \in \{M, M+1\}$ or $d_{i1} \in \{M, M+1\}$;
2. Decrease d_{i0} by 1 if $M+2 \leq d_{i0} \leq m$, and decrease d_{i1} by 1 if $M+2 \leq d_{i1} \leq m$.

3. Then, the receiver extracts the remaining message and recovers original blocks from the blocks with block indexes recorded in the overhead information extracted in the above extraction process.

3.6.1 Algorithm

Following steps are followed to recover the original message from the image provided the key of four is available. The receiver then uses this key of four to go through the following steps and recover the embedded text message.

(1)Divide the image into 3 pixels non overlapping blocks according the BO and BT information available in the overhead information [key of four].

(2) Now scan the image to recover the original message in the order it was stored.

For each block i with $1 \leq b_{i0}, b_{i1}, b_{i2} \leq 254$, perform the actions in accordance with the conditions listed in Figure 13.

(3) After completing the above steps, save the extracted message bits in the list List-1. If $\min(b_{i0}, b_{i1}, b_{i2}) = 1$ or $\max(b_{i0}, b_{i1}, b_{i2}) = 254$, store the extracted information in the list List-2 if $2 \leq b_{i0}, b_{i1}, b_{i2} \leq 253$.

List-1 contains the information embedded in step 6 and

List-2 contains the overhead information (List-2`) and the message (List-2``) embedded in step 7 in Section 3.4.

(4) Decode the overhead information in List-2` according to the formats defined in Figure 11.

(5) For each FB, which is recorded in List-2`, perform the actions listed in Table 4 if they fulfill the listed conditions. After performing these actions save the extracted message bits in the list List-3.

(6) According to the indexes of the blocks where the message bits are extracted, reorder the message bits saved in List- 1 and List-3 to form the list List-4 which is the message embedded in step 6 in Section 3.4.

(7) Push the message in List-2``first, then the message obtained in step 6 (List-4) onto a stack.

(8) Go to step 1 until the message embedded in each embedding Channel is completely extracted.

(9) Pop the message pushed in step 7 off the stack to obtain the message embedded in the stego-image

Items	Conditions	Actions
1	$d_{i0} == M$ and $d_{i1} == M$	Extract “00”
2	$d_{i0} == M$ and $d_{i1} == M + 1$ and $b_{i1} > b_{i2}$	Extract “01”, $b_{i2} = b_{i2} + 1$
3	$d_{i0} == M$ and $d_{i1} == M + 1$ and $b_{i1} < b_{i2}$	Extract “01”, $b_{i2} = b_{i2} -1$
4	$d_{i0} == M$ and $M + 1 < d_{i1} \leq m$ and $b_{i1} > b_{i2}$	Extract “0”, $b_{i2} = b_{i2} + 1$
5	$d_{i0} == M$ and $M + 1 < d_{i1} \leq m$ and $b_{i1} < b_{i2}$	Extract “0”, $b_{i2} = b_{i2} -1$
6	$d_{i0} == M$ and ($d_{i1} < M$ or $d_{i1} > m$)	Extract “0”
7	$d_{i0} == M + 1$ and $d_{i1} == M$ and $b_{i0} < b_{i1}$	Extract “10”, $b_{i0} = b_{i0} + 1$
8	$d_{i0} == M + 1$ and $d_{i1} == M$ and $b_{i0} > b_{i1}$	Extract “10”, $b_{i0} = b_{i0} -1$
9	$d_{i0} == M + 1$ and $d_{i1} == M + 1$ and $b_{i0} < b_{i1} < b_{i2}$	Extract “11”, $b_{i0} = b_{i0} + 1$, $b_{i2} = b_{i2} -1$
10	$d_{i0} == M + 1$ and $d_{i1} == M + 1$ and $b_{i0} > b_{i1} > b_{i2}$	Extract “11”, $b_{i0} = b_{i0} -1$, $b_{i2} = b_{i2} + 1$
11	$d_{i0} == M + 1$ and $d_{i1} == M + 1$ and $b_{i0} < b_{i1} > b_{i2}$	Extract “11”, $b_{i1} = b_{i1} -1$
12	$d_{i0} == M + 1$ and $d_{i1} == M + 1$ and $b_{i0} > b_{i1} < b_{i2}$	Extract “11”, $b_{i1} = b_{i1} + 1$
13	$d_{i0} == M + 1$ and $M + 1 < d_{i1} m$ and $b_{i0} > b_{i1} > b_{i2}$	Extract “1”, $b_{i0} = b_{i0} -1$, $b_{i2} = b_{i2} + 1$
14	$d_{i0} == M + 1$ and $M + 1 < d_{i1} m$ and $b_{i0} < b_{i1} < b_{i2}$	Extract “1”, $b_{i0} = b_{i0} + 1$, $b_{i2} = b_{i2} -1$
15	$d_{i0} == M + 1$ and $M + 1 < d_{i1} m$ and $b_{i0} < b_{i1} > b_{i2}$	Extract “1”, $b_{i1} = b_{i1} – 1$
16	$d_{i0} == M + 1$ and $M + 1 < d_{i1} m$ and $b_{i0} > b_{i1} < b_{i2}$	Extract “1”, $b_{i1} = b_{i1} + 1$
17	$d_{i0} == M + 1$ and ($d_{i1} < M$ or $d_{i1} > m$) and $b_{i0} < b_{i1}$	Extract “1”, $b_{i0} = b_{i0} + 1$
18	$d_{i0} == M + 1$ and ($d_{i1} < M$ or $d_{i1} > m$) and $b_{i0} > b_{i1}$	Extract “1”, $b_{i0} = b_{i0} -1$
19	$M + 1 < d_{i0} \leq m$ and $d_{i1} == M$ and $b_{i0} > b_{i1}$	Extract “0”, $b_{i0} = b_{i0} -1$
20	$M + 1 < d_{i0} \leq m$ and $d_{i1} == M$ and $b_{i0} < b_{i1}$	Extract “0”, $b_{i0} = b_{i0} + 1$
21	$M + 1 < d_{i0} \leq m$ and $d_{i1} == M + 1$ and $b_{i0} > b_{i1} > b_{i2}$	Extract “1”, $b_{i0} = b_{i0} -1$, $b_{i2} = b_{i2} + 1$
22	$M + 1 < d_{i0} \leq m$ and $d_{i1} == M + 1$ and $b_{i0} < b_{i1} < b_{i2}$	Extract “1”, $b_{i0} = b_{i0} + 1$, $b_{i2} = b_{i2} -1$
23	$M + 1 < d_{i0} \leq m$ and $d_{i1} == M + 1$ and $b_{i0} < b_{i1} > b_{i2}$	Extract “1”, $b_{i1} = b_{i1} -1$
24	$M + 1 < d_{i0} \leq m$ and $d_{i1} == M + 1$ and $b_{i0} > b_{i1} < b_{i2}$	Extract “1”, $b_{i1} = b_{i1} + 1$
25	$M + 1 < d_{i0} \leq m$ and $M + 1 < d_{i1} \leq m$ and $b_{i0} > b_{i1} > b_{i2}$	$b_{i0} = b_{i0} -1$, $b_{i2} = b_{i2} + 1$
26	$M + 1 < d_{i0} \leq m$ and $M + 1 < d_{i1} \leq m$ and $b_{i0} < b_{i1} < b_{i2}$	$b_{i0} = b_{i0} + 1$, $b_{i2} = b_{i2} -1$
27	$M + 1 < d_{i0} \leq m$ and $M + 1 < d_{i1} \leq m$ and $b_{i0} < b_{i1} > b_{i2}$	$b_{i1} = b_{i1} -1$
28	$M + 1 < d_{i0} \leq m$ and $M + 1 < d_{i1} \leq m$ and $b_{i0} > b_{i1} < b_{i2}$	$b_{i1} = b_{i1} + 1$
29	$M + 1 < d_{i0} \leq m$ and ($d_{i1} < M$ or $d_{i1} > m$) and $b_{i0} < b_{i1}$	$b_{i0} = b_{i0} + 1$
30	$M + 1 < d_{i0} \leq m$ and ($d_{i1} < M$ or $d_{i1} > m$) and $b_{i0} > b_{i1}$	$b_{i0} = b_{i0} -1$
31	($d_{i0} < M$ or $d_{i0} > m$) and $d_{i1} == M$	Extract “0”
32	($d_{i0} < M$ or $d_{i0} > m$) and $d_{i1} == M + 1$ and $b_{i1} < b_{i2}$	Extract “1”, $b_{i2} = b_{i2} -1$
33	($d_{i0} < M$ or $d_{i0} > m$) and $d_{i1} == M + 1$ and $b_{i1} > b_{i2}$	Extract “1”, $b_{i2} = b_{i2} + 1$
34	($d_{i0} < M$ or $d_{i0} > m$) and $M + 1 < d_{i1} \leq m$ and $b_{i1} < b_{i2}$	$b_{i2} = b_{i2} -1$
35	($d_{i0} < M$ or $d_{i0} > m$) and $M + 1 < d_{i1} \leq m$ and $b_{i1} > b_{i2}$	$b_{i2} = b_{i2} + 1$
36	($d_{i0} < M$ or $d_{i0} > m$) and ($d_{i1} < M$ or $d_{i1} > m$)	Do nothing

Table 4: Conditions and their respective actions for extracting a message

Chapter 4
Implementation of Block Difference Method

The major functions developed in the implementation of this method are listed below along with a brief explanation.

4.1 Load Image

This Menu Item enables the user to load the input or stego-image into the application. This function then calls another function Command Enabling.

4.2 Command Enabling

This function enables all the tool items and buttons.

4.3 Merge

This function opens another form for providing the text message that needs to be embedded into the image. Only text input is being catered in this application. If the user enters new text and press OK then another function called Compress Data () is called. This function returns the compressed message using LZW compression algorithm. The result returned is then passed to the Merge Data function which actually hides the message into the image.

4.4 Merge Data

This function first creates a clone of the input image then converts the text input to the binary format so that the resulting bit stream could be embedded into the image. Initialize Statistics function is called afterwards to initialize the arrays that will hold the information regarding g(M)(the number of pixels with minimum difference M), g(m)(the number of pixels with maximum difference m), M (minimum difference) and m(Maximum difference).

It then scans the image horizontally, 3 times for each block type (type-1, type-2, and type-3), takes three consecutive pixels and checks if anyone of them is 0 or 255. If it finds any block that satisfies this condition, it quits; otherwise it takes the absolute difference of the first two and then the subsequent pixels and checks if the differences are not equal to 0 or 255. If so, it increments the corresponding location in the global array that is declared to store all the difference corresponding to all the Block types and Block Orientations.

The same procedure is then repeated for the vertical scan.

Calculate MaxMin() procedure is then called to calculate the Maximum and Minimum differences in the image.

The image is then again scanned horizontally for each block type and every block's pixels are picked up one by one and passed by reference to another function Embed Procedure along with the bytes array that holds the information bits to be embedded. The returned pixel values are then used to set the corresponding pixel in the stego-image. The same procedure is repeated for the vertical scan.

4.5 Convert to Binary

This function creates a bytes array of the size of the text multiplied by 8 for storing the 8 bytes corresponding to every message character. It then calls the Get Bytes built in function to convert the whole text into bytes. These bytes are then stored in the bytes array by using a pointer.

4.6 Initialize Statistics

This function scans the image horizontally, then vertically and initializes the arrays declared for storing the key of four for both the orientations.

4.7 Calculate MaxMin

To calculate the maximum and minimum differences, the global array is scanned vertically and then horizontally to find the maximum (m) and minimum (M) differences in the whole image. The Max and Min difference values are then stored in the corresponding arrays.

4.8 Embed_Procedure

This function fist calculates the differences d_{i0} between the first two pixels and then difference d_{i1} between the second and third pixels. It then checks for a couple of conditions to decide which procedure to call to embed the sender's message bits.

If the difference between the first two pixels is equal to the Minimum difference (M) for input image, it checks the difference between the second and third pixel. If that also happens to be equal to the Minimum Difference M, it calls the procedure embed_2_bits**.** If the difference between the first two pixels is equal to the Minimum difference (M) but the difference between the second and third pixel is not equal to the Minimum Difference M, the function further checks two conditions. If the difference d_{i1} lies between M and m, it calls procedure embed_1_bit_and_increase_difference otherwise embed_1_bit_and_leave_unchanged procedure is called.

If the difference between first two pixels lies between M and m and the difference between the second and third pixel is equal to M *increase_difference_and_embed*_1_*bit* procedure is called.

If both the differences lie between M and m then increase_2_differences procedure is called.

If d_{i0} lies between M and m and d_{i1} does not satisfy the same condition and is not equal to M then procedure increase_difference_and_ leave_ unchanged is called.

If the first difference is not equal to M and it does not lie between M and m then this function checks if d_{i1} is equal to M, if so it calls the procedure leave_unchanged_embed_1_bit. Else it calls d_{i1} lies between M and m it calls *leave_unchanged_and_increase_difference function is called.*

4.9 Embed_2_Bits

This function first traverses the binary data array using a pointer initialized to zero. It picks up two bits from this array, checks whether these bits are 0 or 1 and store them in two separate integer type variables eb_1 and eb_2. If the three pixel values are $b_{i0}>b_{i1}>b_{i2}$ then it adds the first bit to the first pixel and subtracts the second message bit from the third pixel.

If $b_{i0}<b_{i1}<b_{i2}$ then the first message bit is subtracted from the first pixel and the second message bit is added to the third pixel.

If all the three pixel values are equal and the both the message bits are 1 then only the second pixel value is increased by 1. If both the message bits are not one then the first message bit is subtracted from the first pixel and the second message bit is added to the third pixel.
If the first pixel value is less the second pixel and second pixel value is greater than the third pixel and both the message bits are 1 then the procedure increases the second pixel value. However, if both the message bits are not 1 then this function subtracts the first message bit from the first pixel and adds the second pixel value to the third pixel.

If the first pixel value is greater than the second pixel value and the second pixel value is less than the third pixel value and both the message bits are 1 then the second pixel value is increased. Otherwise first and second message bits are added to the first and third pixels respectively.

4.10 embed_1_bit_and_increase_difference

This function picks only one message bit from the binary array and checks the following condtions to perform corresponding actions.

If the first pixel value is greater then the second pixel value and second pixel value is greater than the third pixel value, then the procedure adds the message bit to the first pixel value and 1 is subtracted from the third pixel value.

If the first pixel value is less than the second pixel value and second pixel value is less than the third pixel value then the first message bit is subtracted from the first pixel value and second message bit is added to the third pixel value.

If first pixel value is less than or equal to the second pixel value and second pixel value is greater than the third pixel value and the first message bit is equal to 1 then the middle pixel value is increased and if the first message bit is not 1 then the third pixel value is decreased.

If the first pixel value is greater than or equal to the second pixel value and second pixel value is less than the third pixel value and the first message bit is 1 then the middle pixel value is decreased other wise the third pixel’s value is increased.

4.11 embed_1_bit_and_leave_unchanged

This method picks up one message bit from the binary data array and checks for following condiotns to perform corresponding actions.

If first pixel value is greater than the second pixel value then the message bit is subtratced from the third pixel. Otherwise if second and third pixel values are equal then message bit is added to the third pixel value.

4.12 increase_difference_and_embed_1_bit

As the name suggests this function also picks one message bit from the binary array and checks the following conditions and decides what action to perform.

If the first pixel is greater than the second pixel value and second pixel value is greater than the third pixel value, it adds 1 to the first pixel and subtracts the message bit from the third pixel value.

If the first pixel value is less than the second pixe value and second pixel value is less than the third pixel value then the first pixel value is decreased and third pixel value is increased by adding the message bit to the pixel.

If the first pixel value is less than the second pixel value and the second pixel value is greater than or equal to the third pixel value then the method checks the value of the message bit. If the message bit is 1, middle pixel value is increased by 1 otherwise first pixel's value is decreased by 1.

Similarly if the first pixel value is greater than the second pixel value and second pixel value is less than or equal to the third pixel value then again the message bit is checked. If its value is 1 then the second pixel value is decreased else the value of first pixel is increased.

4.13 increase_2_differences

This method does not pick any message bit. Rather it only alters the difference between the block pixels.

If the first pixel is greater than the second pixel value and second pixel value is greater than the third pixel value the first pixel value is increased and third pixel value is decreased.

If the first pixel is less than the second pixel value and second pixel value is less than the third pixel value, the first pixel value is decreased and the third pixel value is increased.

If first pixel value is less than the second pixel value and second pixel value is greater than the third pixel value, then the second pixel value is increased

If first pixel value is greater than the second pixel value and second pixel value is less than the third pixel value, second pixel value is decreased by 1.

4.14 increase_difference_and_leave_unchanged

If first pixel value is greater than the second pixel, then the first pixel value is increased. However, if the second pixel value is greater thent the first pixel value, first pixel value is decreased.

4.15 leave_unchanged_and_embed_1_bit

It picks one message bit from the binary data array and checks if the second pixel value is greater than the first pixel value. If so the message bit is subtracted from the third pixel value. If the second pixel value is less than or equal to the third pixel value then the message bit is aded to the third pixel value.

4.16 leave_unchanged_and_increase_difference

If the first pixel value is greater than the second pixel value then the second pixel value is decreased by 1 otherwise the second pixel value is increased.

4.17 Save File

Save file method stores the key of four in a text file to be used during the extraction process.

4.18 Extraction

To extract the embedded message, the application requires the user to load the previously saved image. When the extract data button is clicked, the application asks for the input key file so that it can start the extraction process. It then parses the key file and stores the corresponding values in respective variables to be used later.

4.19 Update Statistics

This function extracts the information related to M and m and g(m) and stores it in text fields for all the types (i.e. type-0, type-1, type-2) for both the horizontal and vertical orientations.

4.20 Extract Data

This function declares a Boolean array and calls another function Extract Binary Data. The result returned by the Extract Binary data function is then stored in the Boolean array to be used in the BoolToText() function. This then finally returns the embedded text.

4.21 Extract Binary Data

This function scans the image horizontally first for all the block types. It picks up the values of M and m from global M and m arrays to get Mi and mi pertaining to each block type (in horizontal direction).

Simultaneously this function runs a scan on the image till the image width and picks pixel values for each block and stores them in color type variables. It then cuts the red Channel off from the newly declared color type variables [because red Channel is being used here to embed the message. The function then passes the red Channel values of the three pixel block to the next function extract_bits. The result returned by the extract bits function is then used to construct new colors of the pixels. For this the Green and Blue color components are taken from the function's input pixel values as they have not been altered by this procedure and the red component is taken from the result returned by the extract bits function. These new color values are then passed to the Set Pixel built in function to set the corresponding pixels of the stego-image. It then checks if the length of the result returned by the extract bits function is greater than or equal to the data length, if so, the function quits.

The same procedure is repeated for the vertical scan as well.

The function then converts the result into binary format and returns it to the calling function.

4.22 Extract Bits

The input parameters to this function include three pixel values plus the values of Maximum and Minimum differences. This function first calculates the absolute differences d_{i0} and d_{i1} between the first & second and the second & third pixels respectively. It then calls Log Events function to add an entry to the list box for the three pixels. It then checks 36 different conditions and selects corresponding actions.

It sees if both the differences are equal to M. If so, it returns 00 as the result. It also add an entry to the log saying which actions has been taken, what were the difference values and also states that this is Action1. [Condition # 1]

It then checks if d_{i0} is equal to the minimum difference and d_{i1} is equal to minimum difference plus 1 and the second pixel value is greater than the third pixel value, it increases the third pixel value and returns "01". Just like the first condition, this condition also adds an entry to the log for the conditions fulfilled and the result returned. [Condition # 2]

If d_{i0} is equal to the minimum difference and d_{i1} is equal to minimum difference plus one and the second pixel value is greater than the third pixel value, it decreases the third pixel value and returns "01". An entry to the log is added stating the condition fulfilled and the action taken. [Condition # 3]

If d_{i0} is equal to M, d_{i1} is greater than M+1, second pixel is greater than the third pixel value and $d_{i1}$is less than equal to m, the third pixel value is increased by 1 and 0 is returned as a result. Log Actions function is also called to add an entry to the log for this function. [Condition # 4]

If d_{i0} is equal to M, d_{i1} is greater than M+1, second pixel is less than the third pixel value and d_{i1} is less than or equal to m, the third pixel value is decreased by 1 and 0 is returned as a result. Log Actions function is also called to add an entry to the log for this function. [Condition # 5]

The functions returns 0 after calling Log Events () function if the first difference d_{i0} is equal to M and the second difference d_{i1} is less than M or greater than m. [Condition # 6]

If the first difference is equal to M+1, second difference is equal to M and the first pixel value is less than the second pixel value, this function increases the first pixel value by 1 and return “10” as the result. All these actions are logged through the Log Events function. [Condition # 7]

If the first difference is equal to M+1, second difference is equal to M and the first pixel value is greater than the second pixel value, the first pixel value is decreased by 1 and “10” is returned as the result. Log Events function is called to log this action. [Condition # 8]

If both the differences are equal to M+1 and the first pixel value is less than the second pixel value and second pixel value is less than the third pixel value, first pixel value is increased and third pixel value is decreased. Here “11” is returned as the result. The conditions, action and the result is added to the log. [Condition # 9]

If both the differences are equal to M+1 and the first pixel value is greater than the second pixel value and the second pixel value is greater than the third pixel value then the first pixel value is decreased and third pixel value is increased. “11” is returned as the result and the conditions, action and result is added to the log. [Condition # 10]

If both the differences are equal to M+1 and the first pixel value is less than the second pixel value is greater than the third pixel value, the second pixel value is decreased by 1 and “11” is returned as the result. All these activities are logged. [Condition # 11]

If both the differences are equal to M+1 and the first pixel value is greater than the first pixel value and second pixel value is less than the third pixel value, second pixel value is increased and “11” is returned. Log Action function is called for logging purpose. [Condition # 12]

If the first difference is equal to M and the second difference is greater than M+1 but less than or equal to m and first pixel value is greater than the second pixel value which is greater than the third pixel value then first pixel value is decreased and third pixel value is increased. As a result of these two actions “1’ is returned as a result. [Condition # 13]

If the first difference is equal to M and the second difference is greater than M+1 but less than or equal to m and first pixel value is less than the second pixel value which is further less than the

third pixel value then the first pixel value is increased and the third pixel value is decreased. As a result of these two actions "1' is returned as a result. [Condition # 14]

If the first difference is equal to M and the second difference is greater than M+1 but less than or equal to m and the first pixel value is less than the second pixel value which is greater than the third pixel value then the second pixel value is decreased and "1" is returned as a result. [Condition # 15]

If the first difference is equal to M and the second difference is greater than M+1 but less than or equal to m and the first pixel value is greater than the second pixel value which is less than the third pixel value then the second pixel value is increased and "1" is returned as a result. [Condition # 16]

If the first difference is equal to M and the second difference is less than M or greater than m and the first difference value is less than the second pixel value then the first pixel value is increased by 1 and "1" is returned an the result. [Condition # 17]

If the first difference is equal to M and the second difference is less than M or greater than m and the first difference value is greater than the second pixel value then the first pixel value is decreased by 1 and "1" is returned an the result. [Condition # 18]

If the first difference is greater than M+1 but less than or equal to m and second difference is equal to M and the first pixel value is greater than the second pixel value then the first pixel value is decreased and 0 is returned as the result. [Condition # 19]

If the first difference is greater than M+1 but less than or equal to m and second difference is equal to M and the first pixel value is less than the second pixel value then the first pixel value is increased and 0 is returned as the result. [Condition # 20]

If the first difference is greater than M+1 but less than or equal to m and second difference is equal to M+1 and the first pixel value is greater than the second pixel value which is in turn greater than the third pixel value then the first pixel value is decreased, second pixel value is increased and 1 is returned as the result. [Condition # 21]

If the first difference is greater than M+1 but less than or equal to m and second difference is equal to M+1 and the first pixel value is less than the second pixel value which is in turn less than the third pixel value then the first pixel value is increased, second pixel value is decreased and 1 is returned as the result. [Condition # 22]

If the first difference is greater than M+1 but less than or equal to m and second difference is equal to M+1 and the first pixel value is less than the second pixel value which is in turn greater than the third pixel value then the middle pixel value is decreased by1 and 1 is returned as the result. [Condition # 23]

If the first difference is greater than M+1 but less than or equal to m and second difference is equal to M+1 and the first pixel value is greater than the second pixel value which is in turn less than the third pixel value then the middle pixel value is increased by1 and 1 is returned as the result. [Condition # 24]

If the first and second difference is greater than M+1 but less than or equal to m and the first pixel value is greater than the second pixel value which is greater than the third pixel value then the first pixel value is decreased and third pixel value is increased. [Condition # 25]

If the first and second difference is greater than M+1 but less than or equal to m and the first pixel value is less than the second pixel value which is less than the third pixel value then the first pixel value is increased and third pixel value is decreased. [Condition # 26]

If the first and second difference is greater than M+1 but less than or equal to m and the first pixel value is less than the second pixel value which is greater than the third pixel value then the middle pixel value is decreased. [Condition # 27]

Similarly, if the first and second difference is greater than M+1 but less than or equal to m and the first and third pixel values are greater than the second pixel value then the second pixel value is increased. [Condition # 28]

If the first difference is greater than M+1 but less than or equal to m and second difference is less than M but greater than m and the first pixel value is less than the second pixel value then the first pixel value is decreased. [Condition # 29]

If the first difference is greater than M+1 but less than or equal to m and second difference is less than M but greater than m and the first pixel value is greater than the second pixel value then the first pixel value is increased. [Condition # 30]

If the first difference is less than M or greater than m and second difference is equal to M the function returns 0. [Condition # 31]

If the first difference is less than M or greater than m and second difference is equal to M+1 and the second pixel value is less than the third pixel value, the function decreases the third pixel value and returns 1. [Condition # 32]

If the first difference is less than M or greater than m and second difference is equal to M+1 and the second pixel value is greater than the third pixel value, the function increases the third pixel value and returns 1. [Condition # 33]

If the first difference is less than M or greater than m and second difference is greater than M+1 but less than or equal to m and the second pixel value is less than the third pixel value then the third pixel value is decreased. [Condition # 34]

If the first difference is less than M or greater than m and second difference is greater than M+1 but less than or equal to m and the second pixel value is greater than the third pixel value , then the third pixel value is increased. [Condition # 35]

If both the differences are less than M or greater than m then the function calls the Do nothing () procedure. This Do nothing () function then further calls Log Events function to log that no action is being performed. [Condition # 36]

4.23 Log Event

This function takes the text input and formats it to add to the list box for logging purpose.

4.24 Preview Paint

This function draws the image with the pen specified by multiplying image height and width with the zoom value.

4.25 draw_freq_analyze

This function draws a frequency analyzer showing difference values on x-axis and number of differences on the y-axis for all the six image scans.

4.26 Compress

This method is called when the user enter the text message to be embedded to the stego-image. The function takes two string input parameters. The function first creates a string table. It then picks up the first character from the input string and loops through the entire input text and concatenates next character to match. It then looks for the same sequence in the string table, if it is able to find a matching entry it equates the characters to match with the next string to match. Otherwise, it calls the Write Code method. The corresponding entry is added to the dictionary table.

Compressing the data is simply saving all the bits in a buffer, and then writing them into the destination container in bytes. We can't just either directly writes data into the file (storing the value 245 will cause the number to look like this: 0000000011110101, which will not be acceptable). So, we have to push the buffer data to the left in order to make room for the new data to be inserted. The number of bits we push the data changes according to the number of bits we need for the data. After pushing the data in the buffer we add to it, the data we want it to have. After doing so, we write some information into the destination file and continue with the application.

4.27 Decompress

Decompressing the file is to take all the data we had from the previous compression, and translate it into sequences of information that we can understand.

In the decompression we take the output from the extraction phase and using the dictionary (which we build again while decompressing) we get the sequence of letters that construct this code.

The decode buffer is filled in the opposite way, meaning from the end byte that needs to be written, to the beginning byte of the sequence

Chapter 5
Results and Discussion

In this section, the performance of the proposed scheme is measured by using twenty 800 x 600 colored images as opposed to the original method of Ching-Chiuan Lin, Nien-Lin Hsueh where grey scale images were used. Test images being used as cover images vary a lot from complex (waterfall) to smooth (Laden). Each pixel of the test images takes up 24 bits - 8 bits for each Red, Green Blue color. Here to show the distortion of the stego-image Peak Signal to Noise ratio is used. For an $h \times w$ grayscale image, the PSNR value is defined as follows:

$$PSNR = 10 \times \log_{10} (255 \times 255 \times h \times w)/(\sum^{h-1}\sum^{w-1}_{j=0}(p_{i,j} - q_{i,j})^2) \text{ dB}$$

where $p_{i,j}$ and $q_{i,j}$ denote the pixel values in row i and column j of the cover image and the stego-image, respectively.

In the experiments performed, text information is taken as user's message and embedded into each of the test images. The information is first compressed using LZW algorithm to enhance the data hiding capacity of the test images. All the tests where 10,000 plus characters are stored in a test images, the data is first compressed then it is embedded into the test image.

Image	**Size**	**Bits**	**PSNR (dB)**	**Channel**
Oryx Antelope	1024 X 786	179732	57.985	Red
Lena	512 X512	29540	56.402	Red
Forest	1024 X 786	50000	56.00	Red
Forest Flower	1024 X 786	148000	57.9	Red
Frangipani Flower	1024 X 786	155640	57.8	Red
Garden	1024 X 786	69450	55.2	Red

Garden Sea turtle	1024 X 786	44320	54.1	Red
Humpback Whale	1024 X 786	79000	56.4	Red
Tuco Toucan	1024 X 786	142448	57.8	Red
Tree	1024 X 786	68000	57.0	Red
Waterfall	1024 X 786	49000	56.4	Red
Winter Leaves	1024 X 786	86128	56.0	Red
Lineage	1024 X 786	155745	57.0	Red
Laden	1024 X 786	253313	57.9	Red
Cricket	1024 X 786	137801	56.9	Red
Island	1024 X 786	58963	56.7	Red
Bike	1024 X 786	93017	55.7	Red
Car	1024 X 786	143347	55.4	Red
Truck	1024 X 786	143644	55.3	Red
Arizona	1024 X 786	43404	57.0	Red
Surf	1024 X 786	104648	57.4	Red

Table 5: Embedding Results using Red Channel

Table 5, 6 and 7 shows the maximum payload capacity in bits that the test images can offer using the proposed scheme. Pure payload capacity is the size of space that can be used to embed the user's message, and the payload capacity of an image is the sum of the payload capacities of the embedding Channels. For example, Antelope can hide up to 1,79,732 bits in red channel. Forest flower can hide up to 1,48,000 and Frangipani flower has the capacity to store up to 1.55,650 bits in just one channel

Image	**Size**	**Bits**	**PSNR (dB)**	**Channel**
Oryx Antelope	1024 X 786	179258	58.2	Blue
Forest	1024 X 786	514983	55.9	Blue
Forest Flower	1024 X 786	128611	57.5	Blue
Frangipani Flower	1024 X 786	143857	57.8	Blue
Garden	1024 X 786	81423	55.6	Blue
Garden Sea turtle	1024 X 786	125163	58.7	Blue
Humpback Whale	1024 X 786	88686	57.3	Blue
Tuco Toucan	1024 X 786	139336	57.7	Blue

Tree	1024 X 786	68085	57.4	Blue
Waterfall	1024 X 786	48472	56.6	Blue
Winter Leaves	1024 X 786	76660	66.9	Blue
Lineage	1024 X 786	148504	57.9	Blue
Laden	1024 X 786	231149	57.9	Blue
Cricket	1024 X 786	141179	57.5	Blue
Island	1024 X 786	86791	57.7	Blue
Bike	1024 X 786	90599	56.2	Blue
Car	1024 X 786	137447	58.8	Blue
Arizona	1024 X 786	44553	56.3	Blue
Surf	1024 X 786	986036	56.1	Blue

Table 6: Embedding Results using Blue Channel

If only one channel is used, the proposed algorithm produces a fine quality stego image where the image distortion is reduced to the minimum. Seeing the experimental results the algorithm is known to operate well above PSNR >50 dB. Since only one channel is used at a time, the histograms of the other two channels remain unaltered.

However to further increase the embedding capacity, other two Channels Green and Blue channels could also be used as shown in the tables 8 and 9

Image	**Size**	**Bits**	**PSNR (dB)**	**Channel**
Oryx Antelope	1024 X 786	183847	50.1	Green
Arizona	512 X512	45007	56.0	Green
Forest	1024 X 786	57415	55.3	Green
Forest Flower	1024 X 786	145394	52.1	Green
Frangipani Flower	1024 X 786	148597	57.4	Green
Garden	1024 X 786	123878	56.2	Green
Garden Sea turtle	1024 X 786	127122	55.4	Green
Humpback Whale	1024 X 786	92604	51.2	Green
Tuco Toucan	1024 X 786	143948	50.6	Green
Tree	1024 X 786	76982	55.6	Green
Waterfall	1024 X 786	50804	57.1	Green

Winter Leaves	1024 X 786	88396	59.2	Green
Bike	1024 X 786	95179	50.1	Green
Car	1024 X 786	140679	57.6	Green
Cricket	1024 X 786	145751	54.0	Green
Island	1024 X 786	88663	54.0	Green
Laden	1024 X 786	260767	52.2	Green
Lineage	1024 X 786	163154	54.5	Green
Surf	1024 X 786	1031126	56.2	Green
Truck	1024 X 786	180107	56.4	Green
Tower	1024 X 786	52492	54.9	Green

Table 7: Embedding Results using Green Channel

This table shows the maximum capacities of all the test images when using Green channel as an embedding plane. The results of this plane are comparable to the other two planes. And also, the peak to signal noise ratio of the stego-images to the original image is 55 dB on average, which is quite high.

5.1 Test Results using Red, Green and Blue Channels

The overall color spread is not altered much since only the value of one of the channels in one of the three pixels is changed at a time. This produces minimum distortion and gives an edge to this scheme as it is less likely to be detected. However, the scheme for the time being assumes that the sender and the receiver have already agreed on the key of four i.e. BO, BT, M and m. The computational complexity of the proposed scheme is very less as compared to other frequency domain Steganographic schemes [27].

Following are the tests performed and their results.

5.1.1 Winter Leaves

The first test was performed on the ‘Winter Leaves’ image. The tests were performed in incremental fashion where each increment consisted of 40,000 data bits. The results using the Red channel are shown in the Graph below.

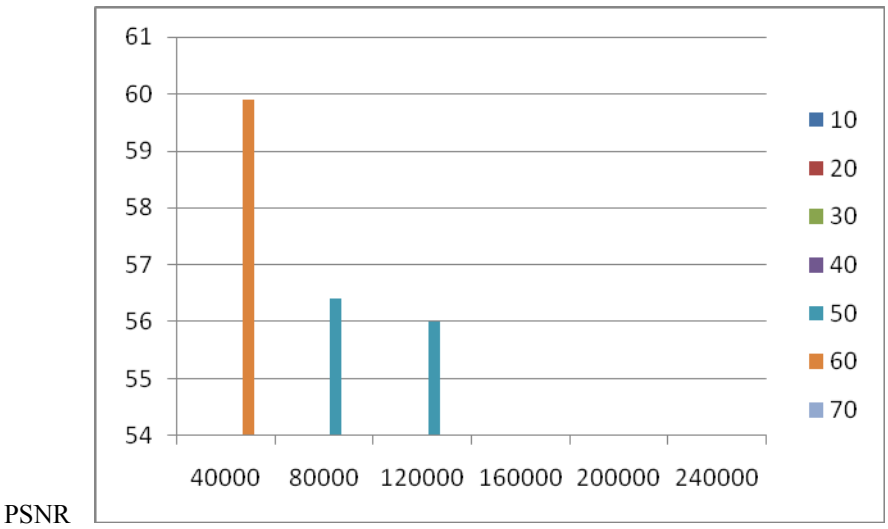


Figure 13: Graph showing PSNR values against increasing data bits in Winter Leaves Image using Red color Channel

The limit testing was performed using the text [see Annex B] consisting of 15,000 characters. (120,000 bits) Out of which only 10,766 characters (86,128 bits) could be saved using the Red channel because the image does not have more capacity in this channel. The image after storing 86,128 bits is shown below.

(a) Original Image **(b) Image after embedding 86128 bits in Red Channel**

Figure 14" Stego-image vs. original image using Red Channel (Winter Leaves)

As can bee seen in figure 13 (a & b), even after altering the red channel values; the image has not undergone much distortion and the PSNR value is also very high i.e. 56 dbs.

Please note that the compression algorithm being used is known to be effective on larger data size. Here the image has the capacity to store 86,128 bits only, and input data size 10,766 characters when compressed generated 86,864 bits. This compressed data size is more than the image capacity and the size of the original data. So the algorithm chooses to embed the uncompressed/original input data into the image.

Similarly, when the same process is repeated with the Blue channel using the same 'Winter Leaves' image , 76,660 bits could be saved at maximum. The tests and their results are summarized in the graph below.

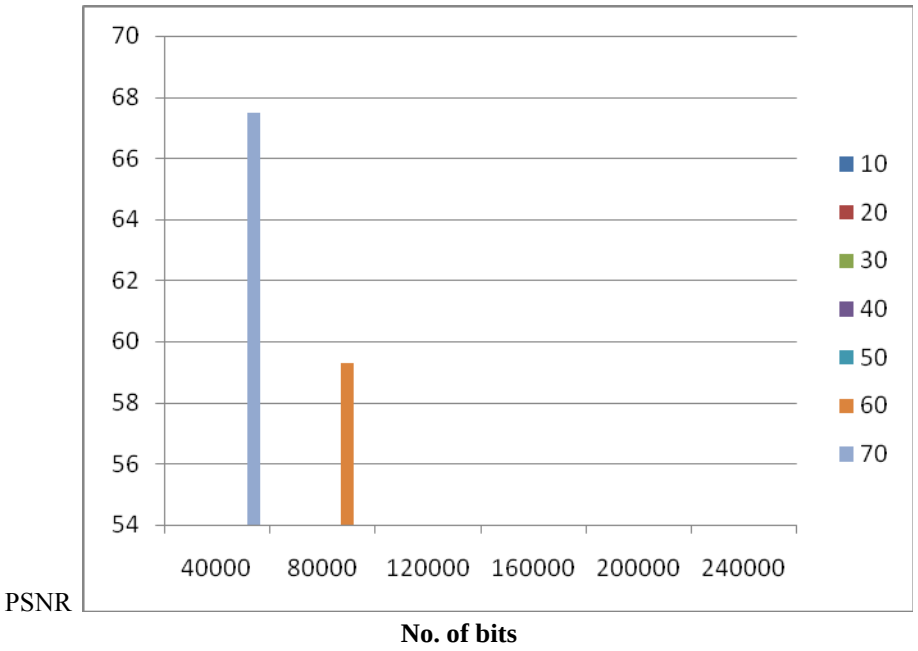


Figure 15: Graph showing PSNR values against increasing data bits in Winter Leaves Image using Blue color Channel

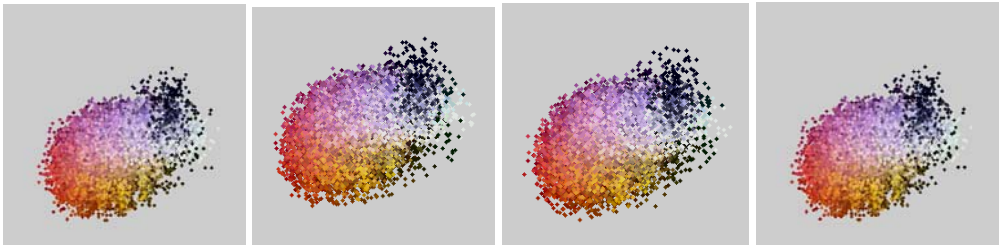
Now, here two different data sets were used. Using random data in Annex-B only 9,582 characters could be saved at maximum, because the compressed data size was 78,192 which is surely more than the image capacity. Therefore, compression could not be used to increase the image's data hiding capacity. However, when the data set provided in Annex-C was used, 14,514 characters could be saved because here the compression could reduce the data set size (1,18,456 bits) to 76,656 bits which is less than the total capacity of the image.

(a) Original Image **(b) Image after embedding 76656 bits in Blue Channel**

Figure 16: Stego-image vs. original image using Blue Channel (Winter Leaves)

It is difficult to discern the above two images as hiding data has not degraded the image quality. The PSNR of the two images is 67 dB.

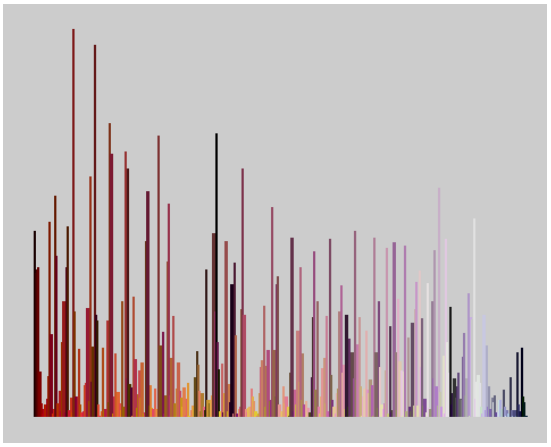
Similarly, the green Channel of this image has the capacity to hide 88,396 characters. If the text given in Annex-B is used, 10,987carcters (87,896 bits) could be saved. However, if text given in Annex-C is used as input 11,049 characters (88,392 bits) could be saved without using compression.



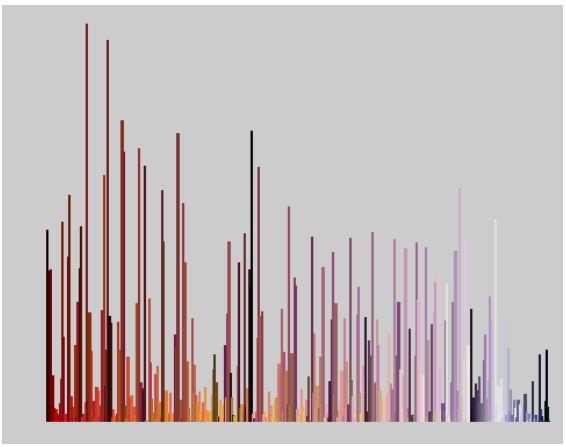
(a) Original Image **(b) After embedding data in Red Channel** **(c) After embedding data in Blue Channel** **(d) After embedding data in Green Channel**

Figure 17: Color clouds of Stego-images (Winter Leaves)

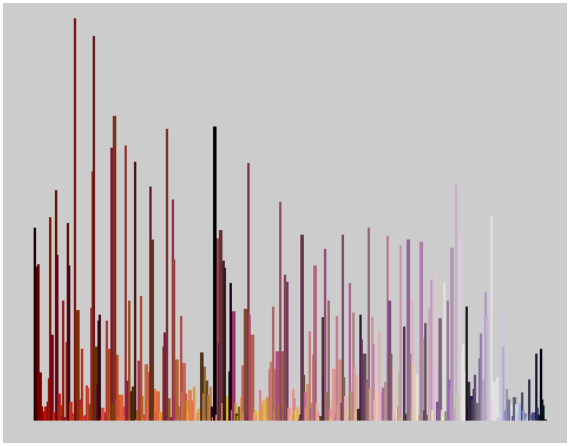
Above are the color clouds to further show the effects of the changes made by the algorithm to the original image using the maximum capacities of Red (Figure 16 b), Blue (figure 16 c) and Green channels (figure 16 d).



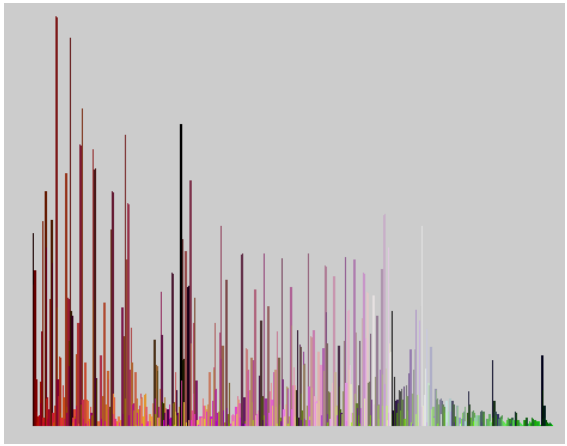
(a) Color bars for the original image



(b) Color bars for the Stego image (using Red Channel)



(c) Color bars for the Stego image (using Blue Channel)



(d) Color bars for the Stego image (using Green Channel)

Figure 18: Color Bars of Stego Images (Winter Leaves)

The color bars (Figure 17 d) show that using the green Channel changes the histogram a lot which could make the stego-image suspicious so it's better to resort to the rest of the two channels for secret communication.

5.1.2 Oryx Antelope

The second set of tests was performed on the following 'Oryx Antelope' image. See the graph below for detailed results.

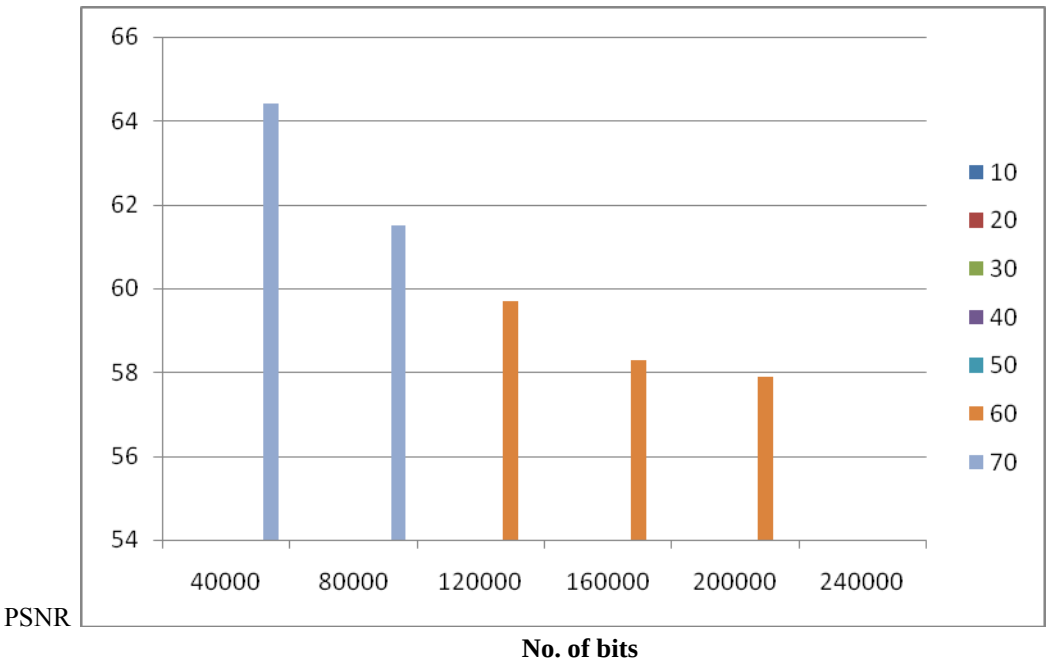
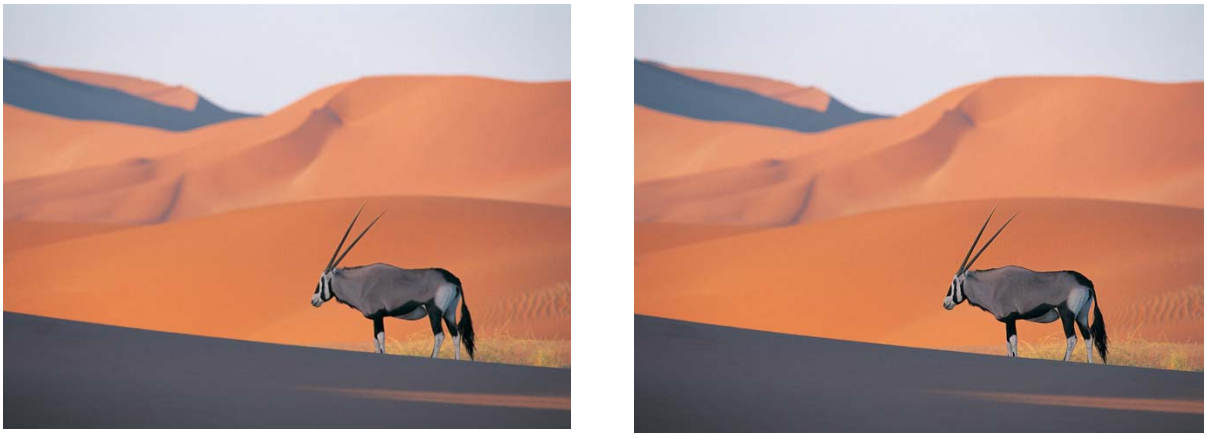


Figure 19: Graph showing PSNR values against increasing data bits in Oryx Antelope Image using Red color Channel

Here using the text in Annex-D, 22,466 characters (1, 79,728 bits) could be saved in the Red channel without using compression. However, if compression is used for random text provided in Annex-D, 24,406 characters (19,5,248 bits) could be used because compression reduced the data size to 17, 9,728 bits which is equal to the capacity of the Red channel in this image.



(a) Original Image **(b) Image after embedding 179728 bits in Red Channel**

Figure 20: Stego-image vs. original image using Red Channel (Oryx Antelope)

Similarly, using the text given in Annex-B, 24,407 characters (19,5,256) bits could be saved using compression. It is because using LZW algorithm, the program could reduce the data size to 17,9,520 bits from 19,5,256 bits which is less than the total capacity of this image when using Red channel as a data embedding Channel. The distortion measured as PSNR is 57,985 dB which is highest in all the tests performed.

Using Blue channel the image has the total capacity to store 1,79,258 bits. The other test results are shown in the graph below.

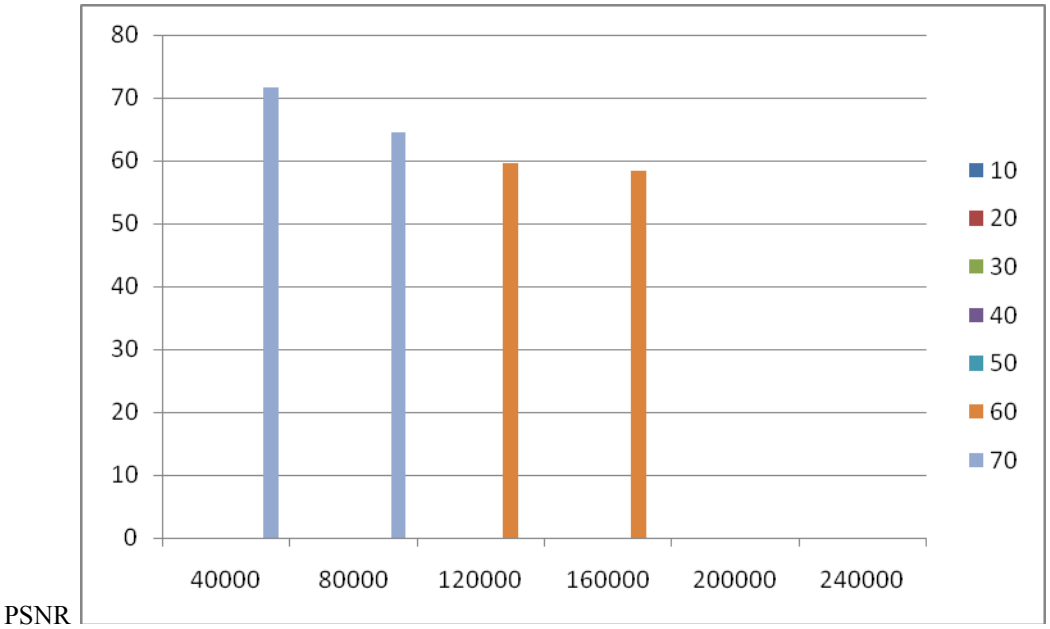


Figure 21: Graph showing PSNR values against increasing data bits in Oryx Antelope Image using Blue color Channel

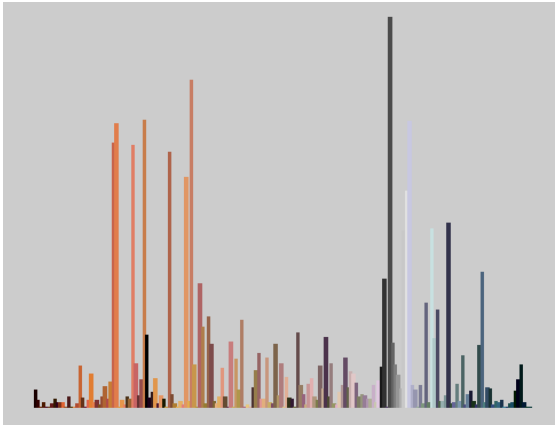
Using random data in Annex-D, the image could store 24,319 characters (1, 94,552 bits) after compression. The total data bits stored in this image were 1, 79,200 bits, the output of the compression algorithm. However, if the compression is not used the image could hold only 22,407 characters which is quite less than the results obtained after using compression. The PSNR value after embedding this amount of data was 58.25 dB.

(a) Original Image

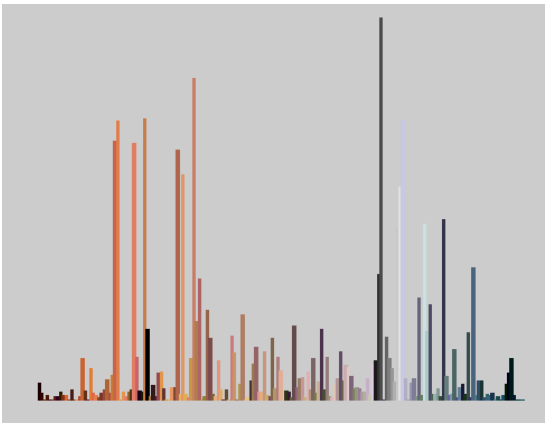
(b) Image after embedding 179258 bits in Blue Channel

Figure 22: Stego-image vs. original image using Blue Channel (Oryx Antelope)

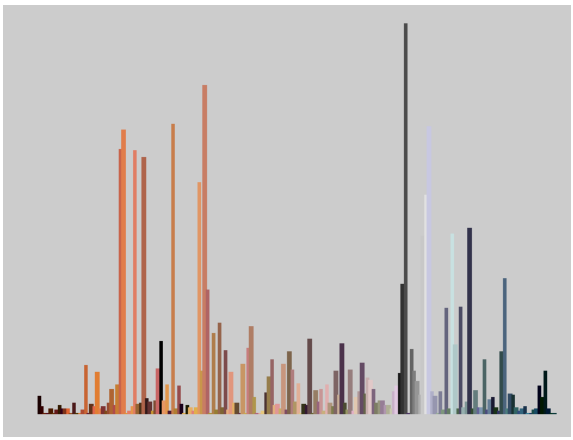
Similarly, using the Green channel, the algorithm was able to store 22,980 characters (1, 83,847 bits) without using compression. However, if compression is used, the image could hide 25,100 characters (2,00,800 bits) using the same Green Channel because the compression could reduce the data set size to 1,83,776 bits which is less than the total capacity of this image using the Green Channel. The PSNR value after storing 1.83,847 bits was 58 dB.



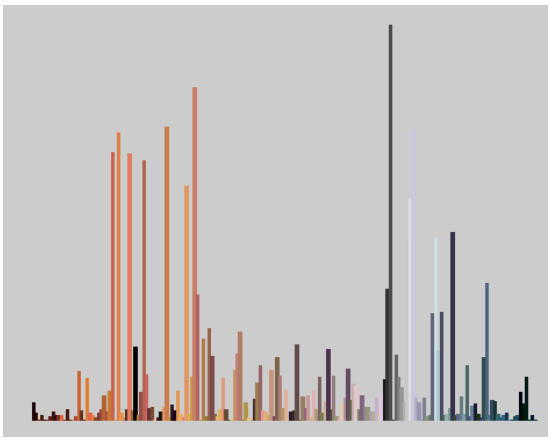
(a) Color bars for the original image



(b) Color bars for the Stego image (using Red Channel)



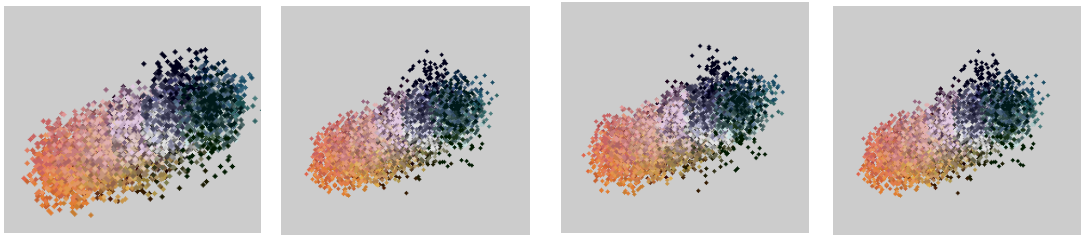
(c) Color bars for the Stego image (using Blue Channel)



(d) Color bars for the Stego image (using Green Channel)

Figure 23: Color Bars of Stego Images (Oryx Antelope)

The above Color bars depict the changes brought about by the algorithm after storing data in Red, Green and Blue data embedding Channels to their maximum capacities.



(a) Original Image (b) After embedding data in Red Channel (c) After embedding data in Blue Channel (d) After embedding data in Green Channel

Figure 24: Color Clouds – Oryx Antelope

The Color clouds in figure 13 (a,b,c and d) show the dispersion achieved after the algorithm hides data in the three Channels. As is evident from the clouds, the changes made by the program are insignificant as compared to the amount of data stored. This is the unique feature of this technique that it does not compromise the image quality and leaves the excess of input data if it happens to exceed the maximum data hiding capacity of any particular test image.

5.1.3 Laden

Another set of tests was performed on the following 'Laden' image. Here using the Red Channel, the image has the capacity to store 31,664 characters (2,53,313 bits). This limit was discovered after performing the tests shown in the graph below.

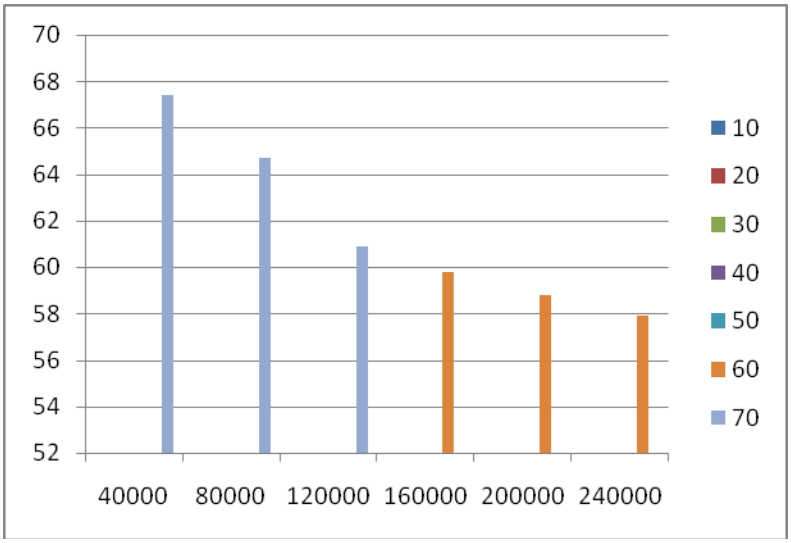


Figure 25: Graph showing PSNR values against increasing data bits in Laden Image using Red color Channel

However, using compression we could achieve higher payload capacity. Using compression, the program could embed 36,020 characters (2,88,160 bits) because the compressed data size was 2,53,248 bits which is less than 2,53,313 bits – the total data hiding capacity of this image in Red Channel. The results of embedding this amount of data are shown below.

(a) Original Image **(b) Image after embedding 253313 bits in Red Channel**

Figure 26: Stego-image vs. original image using Red Channel (Laden)

The distortion produced by Steganography is not discernable by human eye and this marks the success of this algorithm. The PSNR of the original and stego image is 57.9 dbs (one of the highest values considering the test data).

(a) Original Image **(b) Image after embedding 231149 bits in Blue Channel**

Figure 27: Stego-image vs. original image using Blue Channel (Laden)

Using the Blue Channel of the same image, we can embed 28893 characters (2,31,149 bits) without using compression. See the graph below.

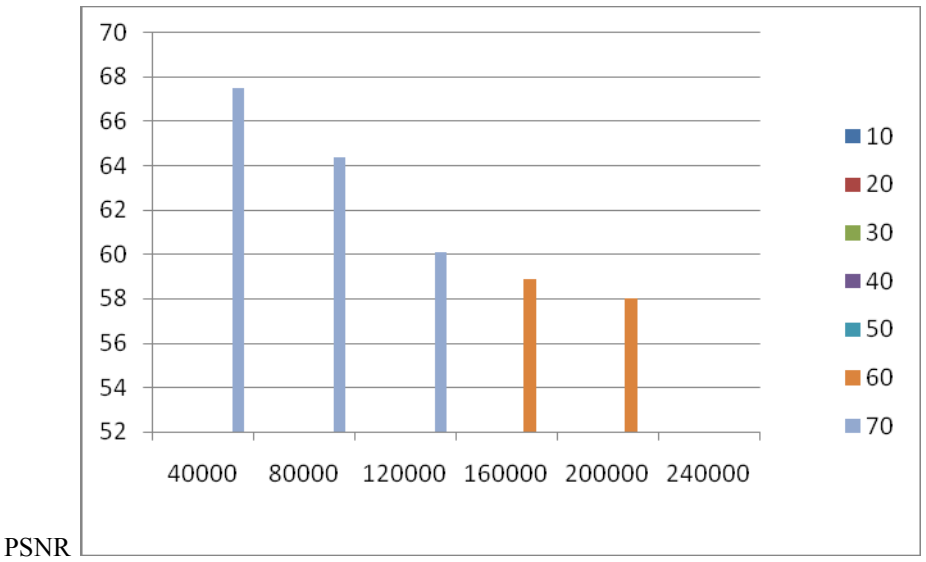
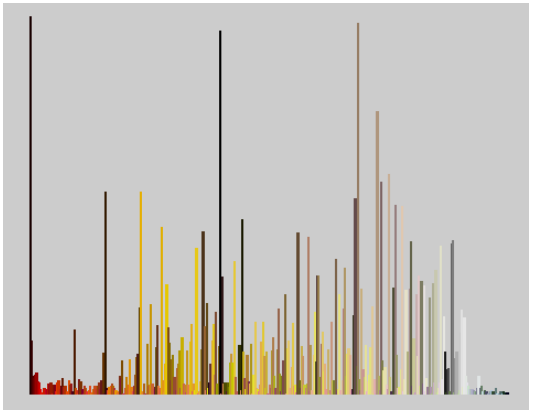


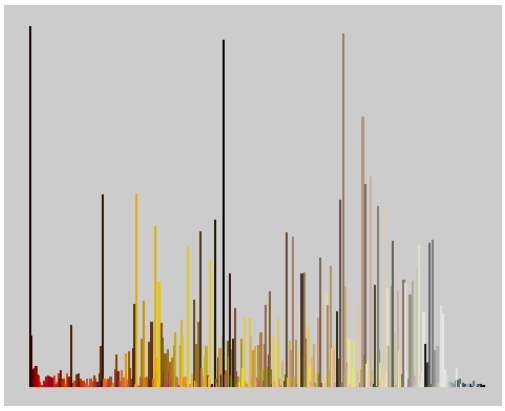
Figure 28: Graph showing PSNR values against increasing data bits in Laden Image using Blue color Channel

However, if compression is used, the image could hide 32,363 characters (2,58,904 bits) compressed to 2,30,944 bits. The PSNR in this case was found to be 57.6 dB. The results are shown in the figures above.

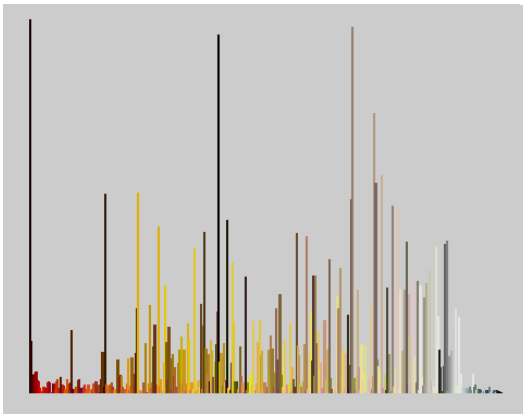
Similarly, using the Green Channel the image could embed 260767 bits.



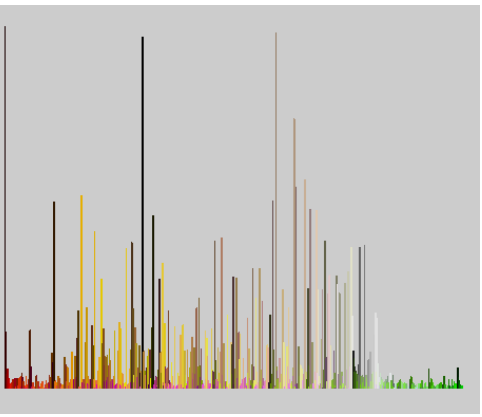
(a) Color bars for the original image



(b) Color bars for the Stego image (using Red Channel)



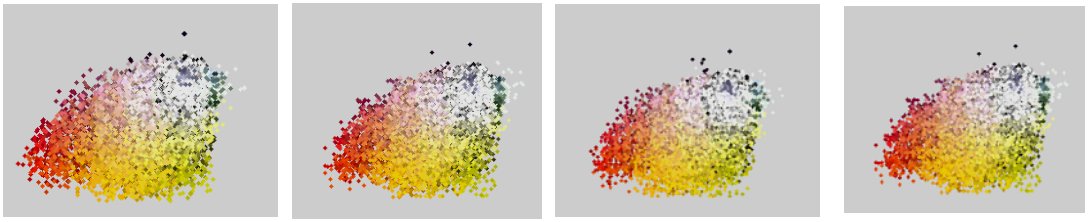
(c) Color bars for the Stego image (using Blue Channel)



(d) Color bars for the Stego image (using Green Channel)

Figure 29: Color Bars of Stego Images (Laden)

These color graphs show the color distribution of the test image before and after embedding secret information**.**



(a) Original **(b) Red Channel** **(c) Blue Channel** **(d) Green Channel**

Figure 30: Color Clouds - Laden

The color clouds also show the dispersion produced in the stego-images after altering their Red, Green or Blue Channels individually. As can be seen only slight variation in the color cloud is produced even after embedding data to the maximum limit of this image.

5.1.4 Frangipani Flowers

The fourth set of tests was performed on the following 'Frangipani Flowers' image. Here, the Red Channel had the capacity to hide 1,55,640 bits which makes 19,455 characters if compression is not used. See the graph below.

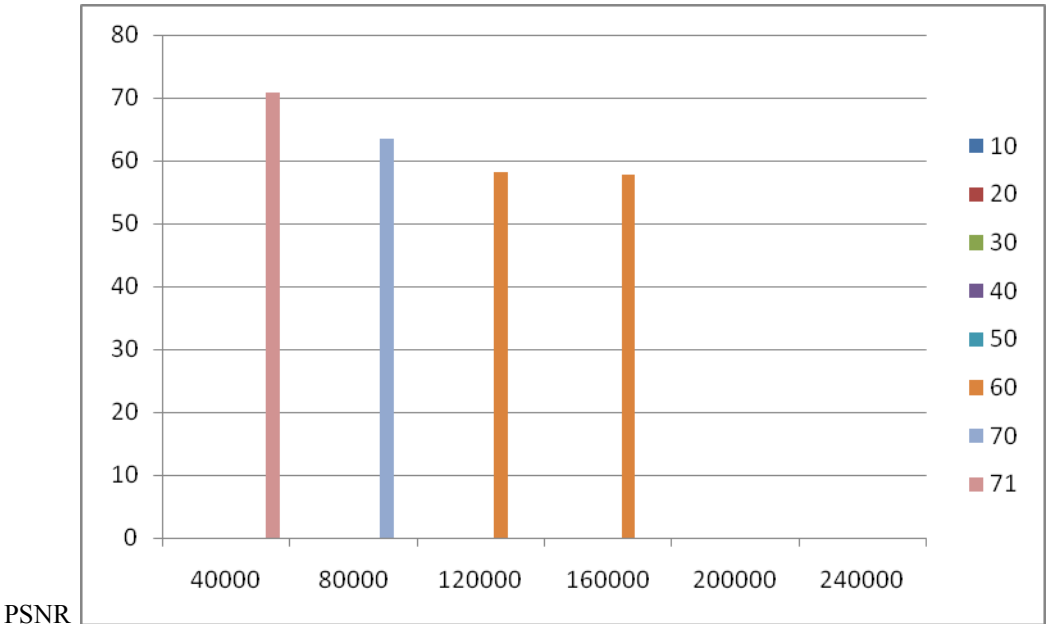
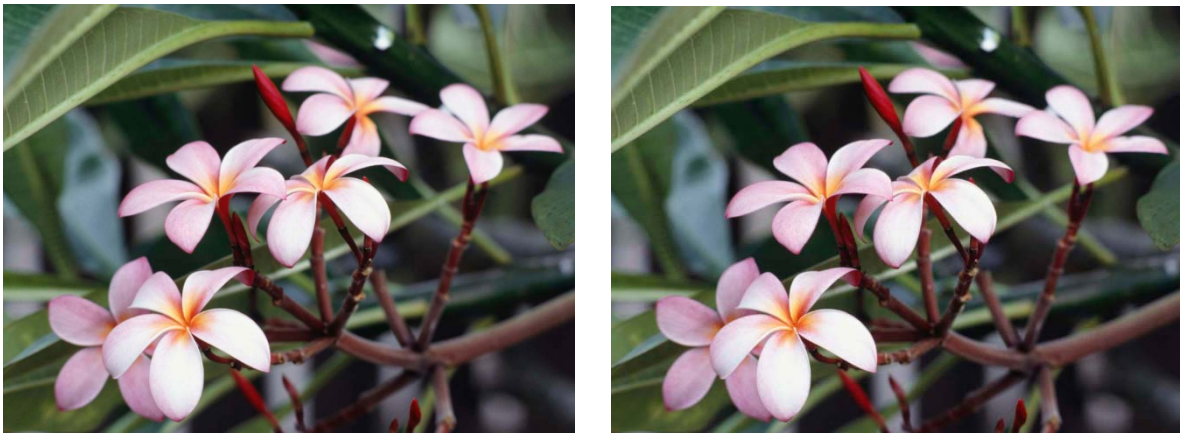


Figure 31: Graph showing PSNR values against increasing data bits in Frangipani Flowers Image using Red color Channel

However, if compression is used, 20800 characters (1,66,400 bits) could be embedded in the same Channel using the same image. This is because the compression reduces the data set size to 1,55,568 bits which is well within the maximum limit of the current embedding Channel.



(a) Original Image **(b) Image after embedding 155640 bits in Red Channel**

Figure 32: Stego-image vs. original image using Red Channel (Frangipani Flowers)

The PSNR in this case was found to be 57.8 dB.

Using the Blue Channel in the same image, we could embed 1,43,857 bits in total. This limit was discovered after performing following tests on this image.

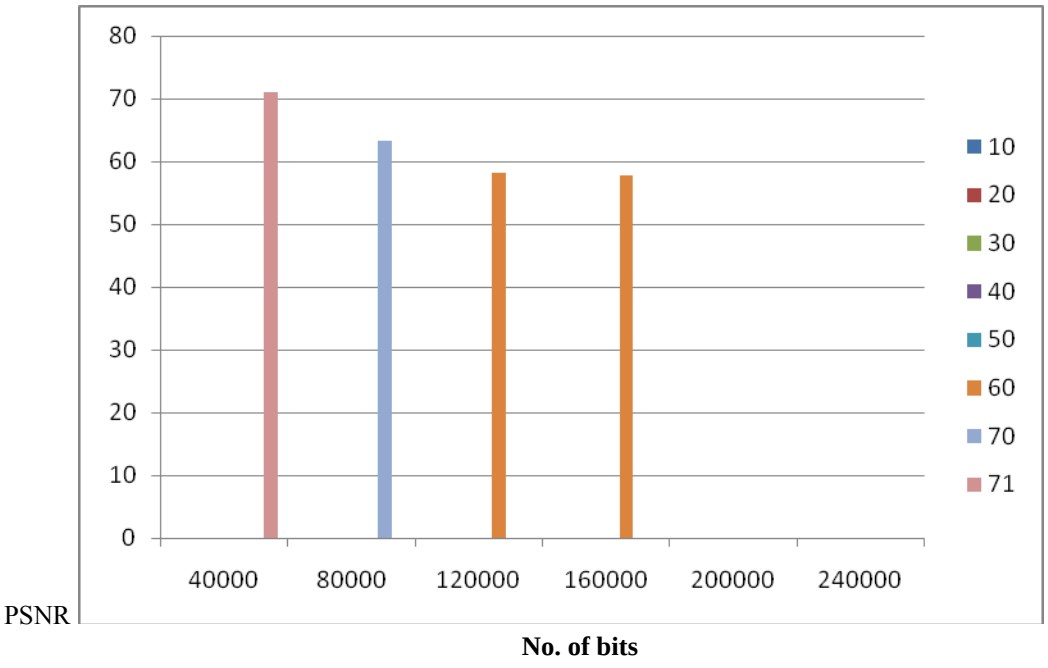
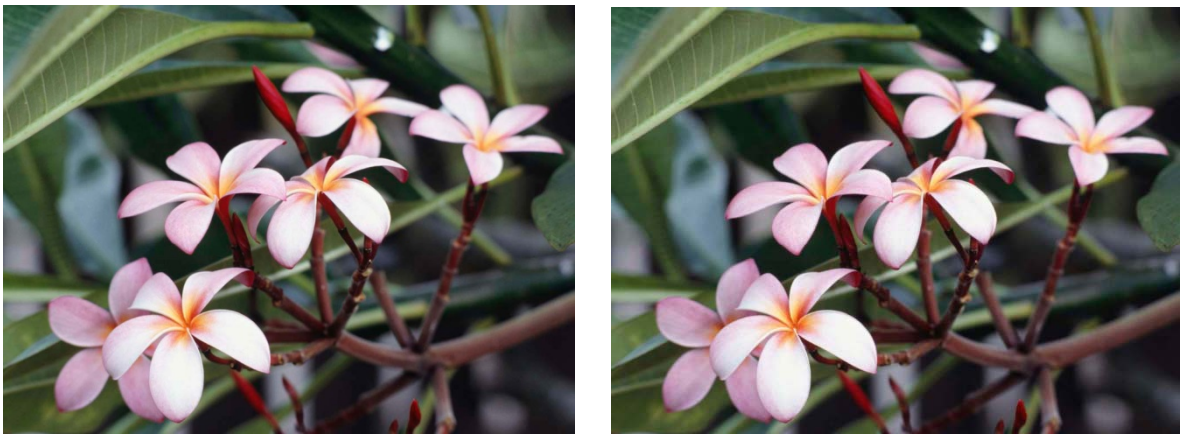


Figure 33: Graph showing PSNR values against increasing data bits in Frangipani Flowers Image using Blue color Channel

If compression is not used, the image can simply hide 17,982 characters. The compression however, enabled us to increase the data hiding capacity of this Channel to 18,854 characters (1,50,832) which makes 14,3088 compressed bits.

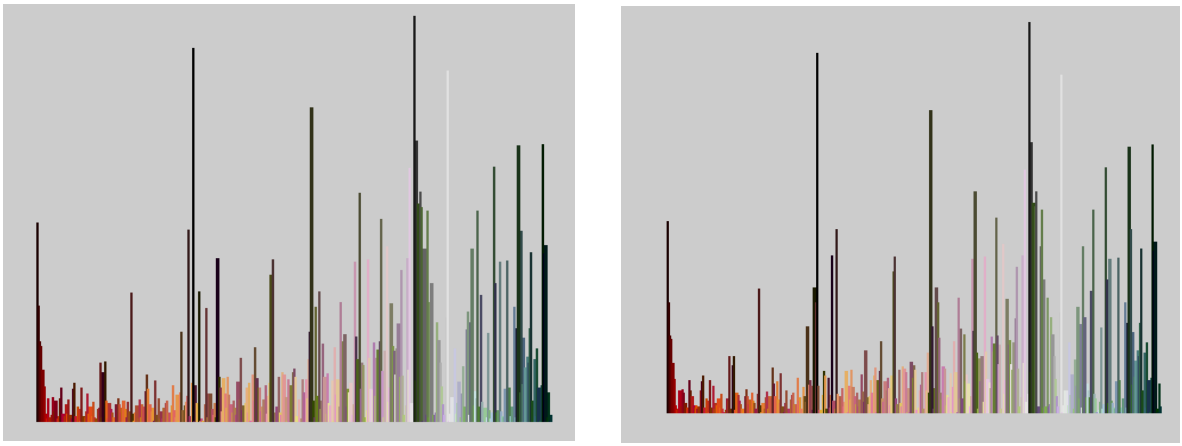


(a) Original Image **(b) Image after embedding 143857 bits in Blue Channel**

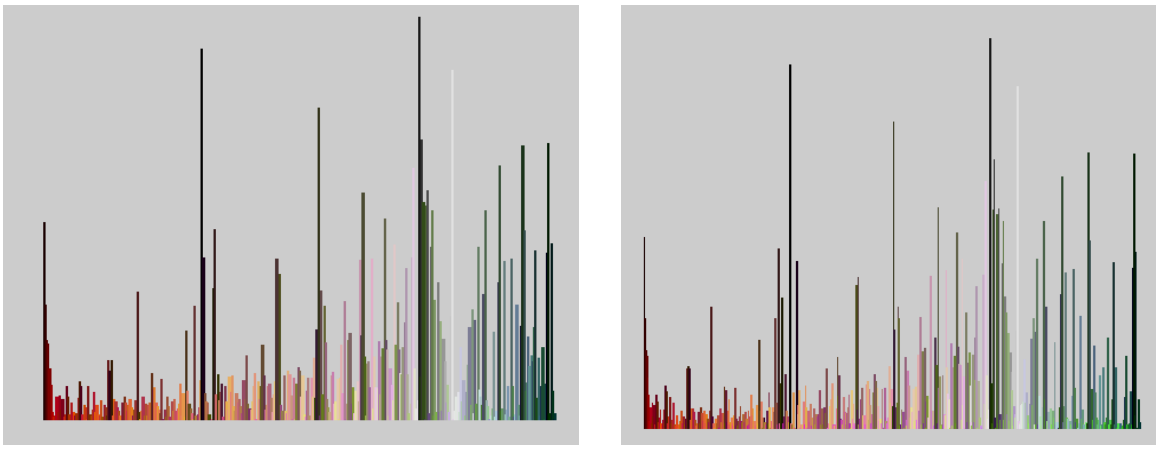
Figure 34: Stego-image vs. original image using Blue Channel (Frangipani Flowers)

The PSNR value in the case of Blue channel is 57.8 dbs.

Similarly the Green Channel of this image has the capacity to hold 1,48,597 bits in total.



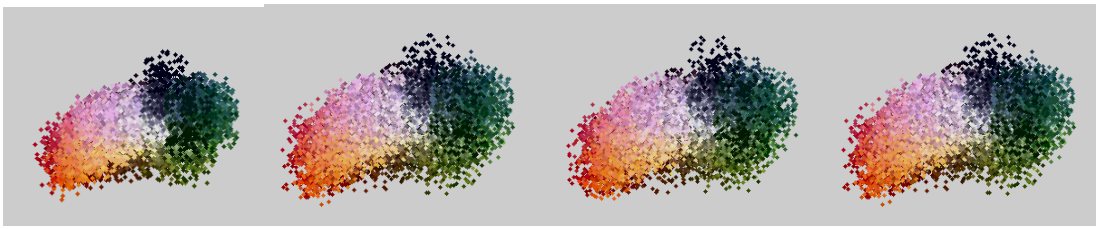
(a) Color bars for the original image **(b) Color bars for the Stego image (using Red Channel)**



(c) Color bars for the Stego image (using Blue Channel) **(d) Color bars for the Stego image (using Green Channel)**

Figure 35: Color Bars of Stego Images (Frangipani Flowers)

These color graphs shows the color distribution of the image pixels in the original and stego images after different color Channels are altered.



(a) Original **(b) Red Channel** **(c) Blue Channel** **(d) Green Channel**

Figure 36: Color Clouds (Frangipani Flowers)

5.1.5 Tuco Toucan

The fifth set of tests was performed on the 'Tuco Toucan' image. The tests results showed that the Red channel in this image has the capacity to hide 1,42,448 bits in total. See the graph below

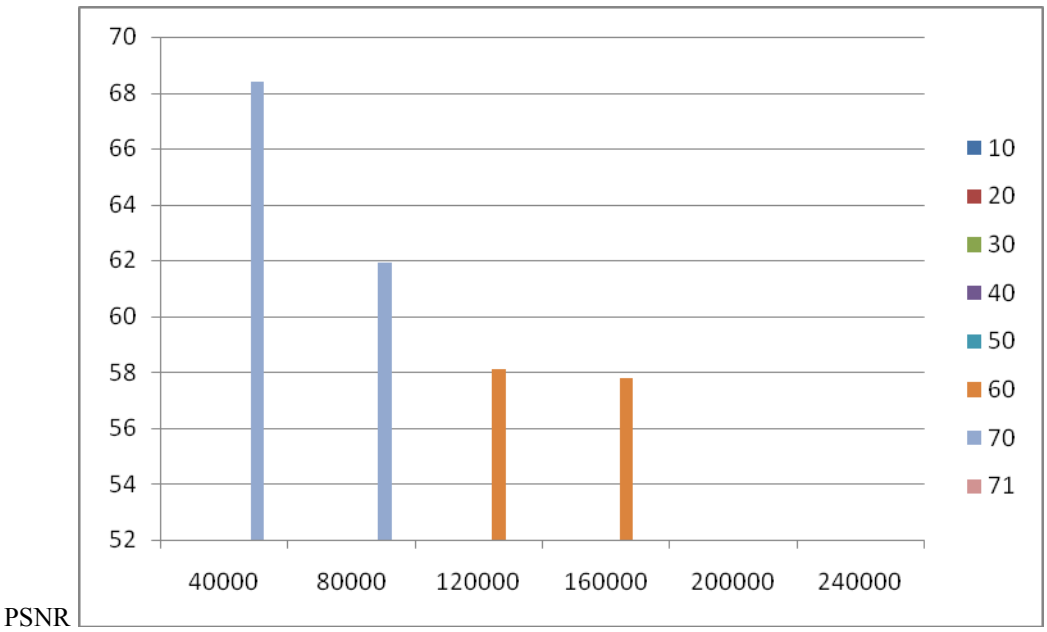


Figure 37: Graph showing PSNR values against increasing data bits in Tuco Toucan Image using Red color Channel

This number of bits makes 17,806 characters if embedded without using compression. Nevertheless, compression if added to the input data, prior to embedding it to the Red channel, can increase its storage capacity. The random, computer generated text provided in Annex-B showed that the LZW algorithm could compress 18,783 characters (1,50,264 bits) to 1,42,400 bits which falls within the maximum data hiding limit of this Channel. See the figures below for the output

(a) Original Image **(b) Image after embedding 142448 bits in Red Channel**

Figure 38: Stego-image vs. original image using Red Channel (Tuco Toucan)

The figures show the results of embedding data in the Red Channel as opposed to the original input image. The PSNR of these two images is 57.8 dB.

Similarly the Blue Channel can embed 13,9336 bits altogether.

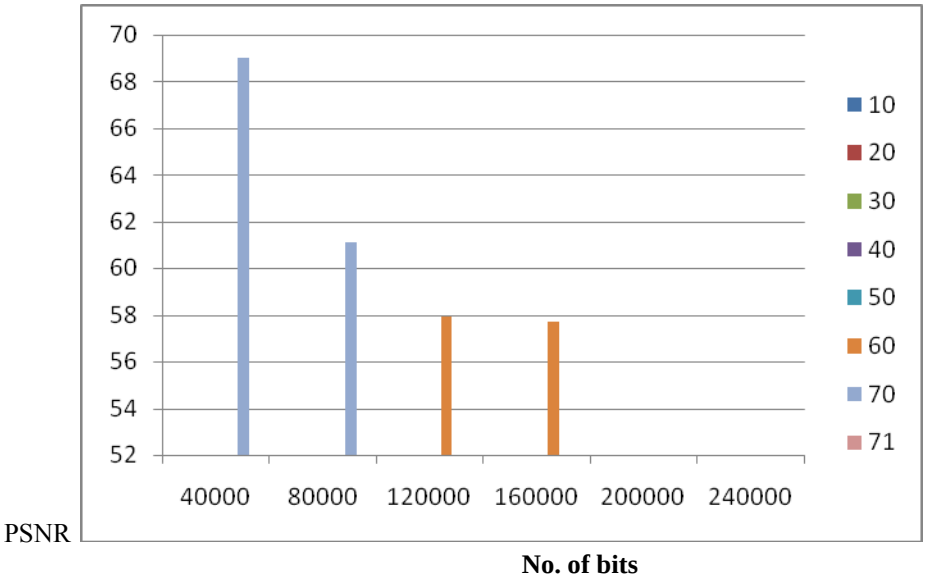
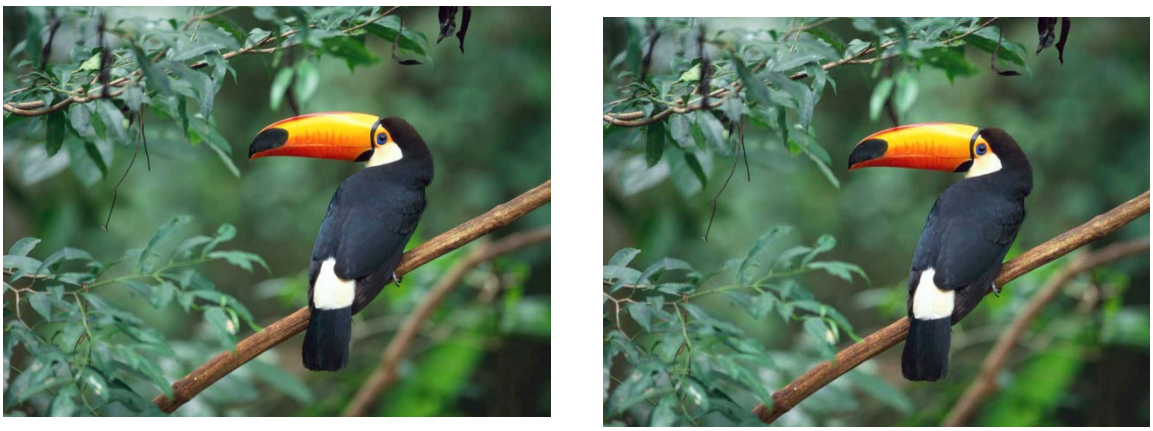


Figure 39: Graph showing PSNR values against increasing data bits in Tuco Toucan Image using Blue color Channel

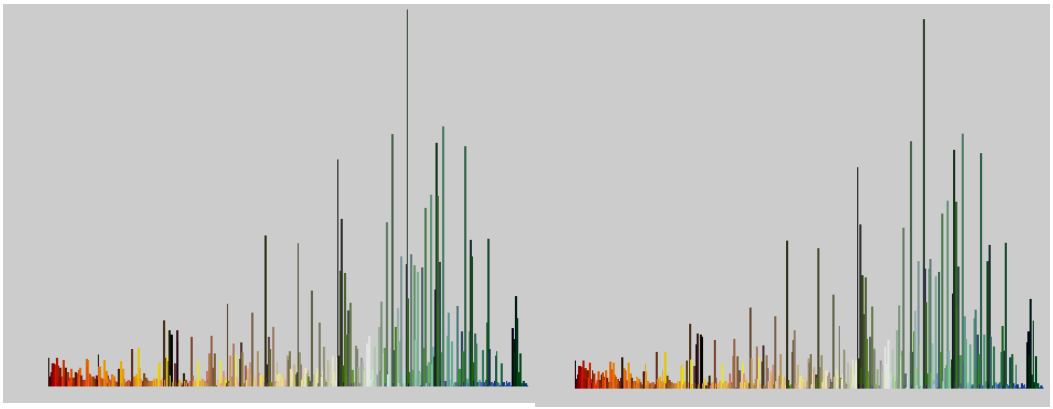
Obviously, if compression is not used, the image can only hide 17,417 characters. However, the experiments show that if LZW is used then the image can hold up to 18,300 characters (1,46,400 Bits) because the compressed data size is 1,39,152 which is less then the maximum limit.



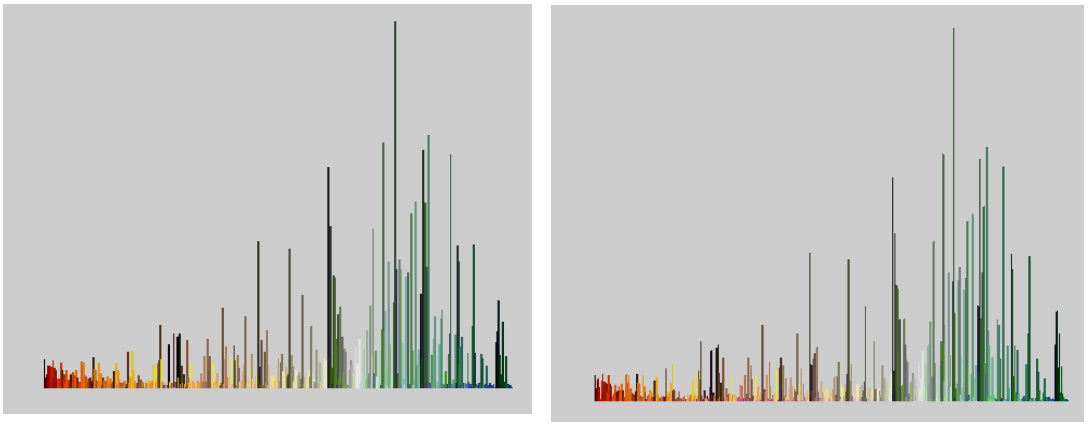
(a) Original Image **(b) Image after embedding 139336 bits in Blue Channel**

Figure 40: Stego-image vs. original image using Blue Channel (Tuco Toucan)

The PSNR of these two images is 57.7 dB.



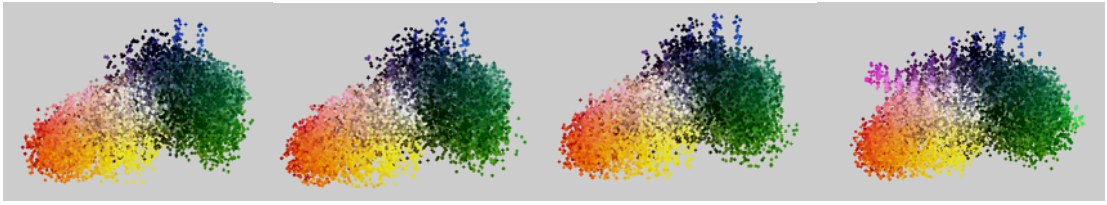
(a) Color bars for the original image **(b) Color bars for the Stego image (using Red Channel)**



(c) Color bars for the Stego image (using Blue Channel) **(d) Color bars for the Stego image (using Green Channel)**

Figure 41: Color Bars of Stego Images (Tuco Toucan)

These color bars show different color content with its frequency in the original and stego test images.



(a) Original **(b) Red Channel** **(c) Blue Channel** **(d) Green Channel**

Figure 42: Color Clouds - Tuco Toucan

The color clouds illustrate that the changes in the color spread are not very significant as compared to the data hiding capacity of these channels.

Comparison with other techniques

Chapter 6

Conclusion and Future Work

Steganography is a relatively new field. The major objective of using Steganography is to hide information in some other medium so as to provide protection against detection. Here, the success of a technique lies in changing the cover medium in a way that makes the change imperceptible for both the human and computer. It differs from cryptography because cryptography aims at concealing the secret message. A major limitation of Steganography is the size of the cover medium used. Therefore, all the Steganographic methods aim to achieve higher hiding capacity in a given medium.

Major media that can be used for Steganography are text, image, audio and video files. Most of the techniques still rely on the agreement between the sender and the receiver on the exchange of the key. There are quite a few ways in which text files could be used for information hiding e.g. moving the text lines up or down, altering or changing the space between the words in the text file etc. Using the audio files, sample quantization, temporal sampling or perceptual sampling could be used to achieve information hiding.

Images however, offer a wide variety of ways in which additional information could be embedded. The major dimensions in which images can be used are spectrum, LSB substitution and transformed domain Steganography. In LSB, the least significant bit of each pixel is replaced with the message bit. In transformed domain Steganography, data hiding is achieved by Fourier transform or discrete cosine transformed. These techniques are more robust and difficult to detect as compared to LSB insertion.

There are few techniques available for color image Steganography. This is because, most of the color image formats like bmp are larger in size and communicating heavy images over open systems like internet can raise suspicions. One of the techniques suggests using edge pixels to hide the message bits. The advantage of using this technique over LSB is that an edge pixel even if altered, will not stand out extraordinarily among its neighboring pixels. Another method uses variable bits from the RGB channels for hiding information. Block difference methods have also been proposed. However, the selection of blocks could be done in a variety of ways.

The technique adopted here also uses block differences to decide whether to embed data in a particular block. A compression has also been used to increase the payload capacity. This method first compressed the input text, and then runs six scans over the input image to calculate the maximum and minimum differences that corresponds to maximum number of pixels. It then uses these differences to decide which blocks to use for embedding data.

The proposed scheme cannot be compared with other techniques since none of the other techniques for color Steganography uses only one Channel at a time to embed secret information. Moreover, there is very little work done in the spatial domain. Most of the techniques, available for color image Steganography, use frequency domain for embedding data into images. All the Spread Spectrum and Frequency based Steganographic methods use all the bits in the image to embed the data.

However, if all the channels are used simultaneously the capacity of the same image, using this technique, could be increased at least 3 times (See table 10).

Image	**Size**	**Bits in Red Channel**	**Bits in Blue Channel**	**Bits in Green Channel**	**Total capacity**
Oryx Antelope	1024 X 786	179732	179258	183847	542837
Forest	1024 X 786	50000	128611	57415	236026
Forest Flower	1024 X 786	148000	143857	145394	437251
Frangipani Flower	1024 X 786	155640	81423	148597	385660
Garden	1024 X 786	69450	125163	123878	318491
Garden Sea turtle	1024 X 786	44320	88686	127122	260128
Humpback Whale	1024 X 786	79000	139336	92604	310940
Tuco Toucan	1024 X 786	142448	68085	143948	354481
Tree	1024 X 786	68000	48472	76982	
Waterfall	1024 X 786	49000	76660	50804	176464
Winter Leaves	1024 X 786	86128	148504	88396	323028
Lineage	1024 X 786	155745	231149	95179	482073

Laden	1024 X 786	253313	141179	140679	535171
Cricket	1024 X 786	137801	86791	145751	369623
Island	1024 X 786	58963	90599	88663	238225
Bike	1024 X 786	93017	137447	260767	491231
Car	1024 X 786	143347	44553	163154	351054
Truck	1024 X 786	143644	986036	1031126	216080

Table 8: Cumulative Capacities of test images

It has been shown in chapter 4 that the algorithm inherently preserves the image quality and operates well above 50 dB PSNR in all the test cases. Currently, only one channel is being used. However, the algorithm can be modified to use all the three channels simultaneously. It is more difficult to detect Steganography using this scheme because only one channel is used at a time. Moreover, a single image can be sued for transmitting three independent text files. Another important point to note is that it is the lossless data hiding scheme. All the text that is embedded can be successfully recovered. Furthermore, the computational complexity of this method is quite less as compared to transformed domain techniques.

6.1 Limitations of the Current Technique

Currently, LZW algorithm is being used to compress the user data. The major drawback of this technique is that if the compressed message exceeds the test image capacity, the dictionary is not completely saved. This creates problem while extracting the embedded message.

6.2 Future Work

Currently the scheme takes three consecutive pixels as a block and uses the difference value of any one channel to decide whether to embed data in the current block. However, this algorithm could be further improved to use single pixel's three channel (Red, Green, and Blue) values as a block. This could result in significant improvement in the current techniques capacity. However, this needs to be researched.

Moreover, the implementation of LZW algorithm could be further improved to overcome the above stated limitation.

References

[1]. http://www.tech-faq.com/steganography.shtml

[2]. http://wiki.answers.com/Q/What_is_the_difference_between_stegenography_and_watermarking

[3]. http://en.wikipedia.org/wiki/Security_through_obscurity

[4]. http://www.jjtc.com/stegdoc/sec202.html

[5]. http://itslab.csce.kyushu-u.ac.jp/sakurailab/research/fingerprint.html

[6]. http://en.wikipedia.org/wiki/Digital_signature

[7]. http://www.scribd.com/doc/20529/Seminar-on-Steganography?autodown=doc

[8]: Petitcolas, Fabien A.P., "Information Hiding: Techniques for Steganography and Digital Watermarking.", 2000.

[9] Sellars, D., "An Introduction to Steganography",
URL: http://www.cs.uct.ac.za/courses/CS400W/NIS/papers99/dsellars/stego.html

[10]: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA349102

[11] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding", *IBM Systems Journal*, vol. 35, Issues 3&4, 1996, pp. 313-336.

[12] D. Huang and H. Yan, "Interword Distance Changes Represented by Sine Waves for Watermarking Text Images", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, December 2001, pp.1237-1245

[13] K. Rabah, "Steganography-The Art of Hiding Data", *Information Technology Journal*, vol. 3, Issue 3, 2004, pp. 245-269.

[14] S.H. Low, N.F. Maxemchuk, J.T. Brassil, and L. O'Gorman, "Document marking and identification using both line and word shifting", *Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '95)*, vol.2, 2-6 April 1995, pp. 853 - 860.

[15] Y. Kim, K. Moon, and I. Oh, "A Text Watermarking Algorithm based on Word Classification and Inter-word Space Statistics", *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR'03)*, 2003, pp. 775–779.

[16] S.H. Low, N.F. Maxemchuk, J.T. Brassil, and L.O'Gorman, "Document marking and identification using both line and word shifting", *Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '95), vol.2, 2-6 April 1995, pp. 853 - 860.*

[17] M. Kharrazi, H. T. Sencar, N. Memon, Image Steganography: Concepts and Practice, *Lecture Note Series, Institute for Mathematical sciences, National University of Singapore, 2004.*

[18] "Issues in Information Hiding Transform Techniques," NRL/MR/5540-02-8621 http://chacs.nrl.navy.mil/publications/CHACS/2002/ 2002chang-NRL-MR-5540-02-8621.pdf #search='2D%20DFT%20in%20steganography

[19] W. Pennebaker and J. Mitchell. "JPEG STILL IMAGE DATA COMPRESSION STANDARD". van Nostrand Reinhold, 1993.

[20] L. Marvel "Image Steganography for Hidden Communication". *Ph.D. Dissertation, Univ. of Delaware, Dept of EE, 1999.*

[21]. Robert T. McKeon "Strange Fourier Steganography in Movies", *Proceedings of IEEE EIT 2007*

[22] Mohammad Shirali-Shahreza, M. Hassan Shirali-Shahreza "Text Steganography in SMS", *International Conference on Convergence Information Technology 2007*

[23]: H. Motameni, M. Norouzi, M. Jahandar, and A. Hatami "Labeling Method in Steganography" *PROCEEDINGS OF WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY VOLUME 24 OCTOBER 2007 ISSN 1307-6884*

[24]: M. Chaumont and W. Puech, " A Color Image Hidden in a Grey-Level Image" *Laboratory LIRMM, UMR CNRS 5506, University of Montpellier II 161, rue Ada, 34392 MONTPELLIER CEDEX 05, FRANCE*

[25]: Elvin M. Pastorfide and Giovanni A. Flores, " An Image Steganography Algorithm for 24-bit Color Images Using Edge-Detection Filter ", *INSTITUTE OF COMPUTER SCIENCE, CMSC 190 SPECIAL PROBLEM*

[26]: Mohammad Tanvir Parvez and Adnan Abdul-Aziz **"**RGB Intensity Based Variable-Bits Image Steganography**"** *Gutub College of Computer Sciences & Engineering King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia*

[27] Wen-Yuan Chen, "Color image steganography scheme using DFT, SPIHT codec, and modified differential phase-shift keying techniques**",** *Department of Electronic Engineering, National Chin-Yi University of Technology, 35 Lane 215, Section 1, Chung-Shan Road, Taiping City, Taichung County 411, Taiwan, ROC, 2007*

[28]: Ran-ZanWang∗,Yao-De Tsai, "An image-hiding method with high hiding capacity based on best-block matching and *k*-means clustering", *Pattern Recognition Society. Published by Elsevier 2006176464*