New Method for Image Inside Image Steganography

طريقة جديدة لاختزال صورة داخل صورة

By

Ahmed Faleh

Supervisor

Dr.Hebah Nasereddin

A Thesis Submitted in Partial Fulfillment of the Requirements for the Master Degree in Computer Science

Faculty of Information Technology

Middle East University

May 2014

اقرار تفويض

اني احمد فالح داود افوض جامعه الشرق الاوسط بتزويد نسخ من رسالتي للمكتبات اوالمؤسسات او الهئات او الافراد عنده طلبها

التوقيع :

التاريخ 201/5/26

Authorization statement

I Ahmed Falih Dawood Authorize the Middle East University

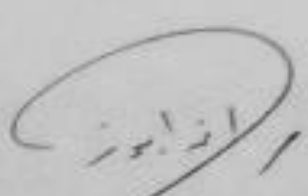
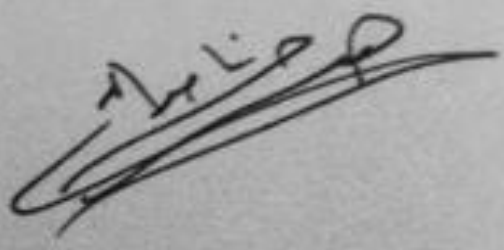
To supply a copy my Thesis to libraries, establishment or individuals

Signature : 

Date:2014-5-26

Examination Committee Decision

This is to certify that the thesis entitled **"New Method For Image Inside Image Steganograhpy"** was successfully defended and approved on.26/5/2014

Examination Committee Member	Signature
1- Dr. Raed Abu Zitar	
2- Dr. Madhafar M. Al-Jarrah	
3- Dr. Hebah H. O. Nasereddin	

DEDICATION

To my father who taught me how to find my way through man's most challenging hardships

To my mother who dedicated her life to watch over me

To my sister with whom I shared with the joy of life

To my sister's husband supported me in this thesis

To the one who guided me toward success in my academic advancement Dr. Hebah Nasereddin

To my friend, my lifetime companion; Ahmad Jala

AKNOWLEDGEMENT

I would like to thank my father and my mother for their continuous support during my study. I also would like to thank my great supervisor Dr. Hebah H. O. Nasereddin for her support, encouragement, proofreading of thesis drafts, and for helping me throughout my studies, putting me in the right step of scientific research. I would like to thank the Information Technology Faculty members at the Middle East University. I would also like to thank my best friend Ahmad Jalal and my friends for their support throughout my academic journey and all of my family members.

Table of Contents

List of Tables

List of Figures

Abstract

Steganography is the art and science of invisible communication, this is accomplished through hiding information in other objects in such a way that no one apart from the intended recipient knows of the information existence, thus hiding the existence of the communicated information.

In this thesis, a new method is proposed to hide a colored image of any type inside 4 channel PNG image.

The method suggests that the selected secret image is converted to an indexed image with custom color palette to reduce its size, the custom color palette is constructed using Lempel-Ziv-Welch (LZW) compression which is applied to the secret image to reduce its size so that the amount of data needed to be embedded with the secret image is reduced.

The proposed method used the least 2 significant bits and achieved better results than the true color method, this is due to the fact that the secret image size was lower after conversion than the converted image in the true color method.

الخلاصة

إخفاء المعلومات هو فن وعلم الاتصالات الغير مرئية، ويتم إنجاز ذلك من خلال إخفاء معلومات في الكائنات الأخرى بحيث لا يمكن لأحد غير المتلقي يعرف عن وجود تلك المعلومات، وبالتالي اخفاء وجود المعلومات، في هذه الأطروحة، يتم اقتراح طريقة جديدة لإخفاء صورة ملونة من أي نوع داخل الصورة اخرى. يقترح هذا الأسلوب تحويل الصورة سرية إلى صورة مفهرسة مع لوح ألوان مخصص لتقليل حجمها، لوحة الألوان مخصصة تبنى باستخدام طريقة (LZW) للضغط التي سيتم تطبيقها على الصورة السرية لتقليل حجمها بحيث يتم تقليل كمية البيانات المطلوبة لتكون جزءا لا يتجزأ مع صورة سرية، وتستخدم الطريقة المقترحة البتات الاول والثاني الأقل وحققت نتائج أفضل، وهذا يرجع إلى حقيقة أن حجم الصورة سرية كان أقل بعد التحويل.

Abbreviations

CO: Cover Object.

DCT: Discrete Cosine Transform.

DFT: Direct Fourier Transform.

DH: Data Hiding.

GIF: Graphics Interchange Format.

LSB: Least Significant Bit.

LZW: Lempel-Ziv-Welch.

RGBA: Red, Green, Blue and Alpha.

RMSE: Root Mean Squared Error.

PNG: Portable Network Graphics.

PNSR: Peak Signal to Noise Ratio

SD: Standard Deviation.

SI: Stego Image.

CHAPTER 1: INTRODUCTION

Introduction

Maintaining privacy in personal communications is something everyone desires, so the security of information and data is one of the most important factors of communication technology. Cryptography is the field of technologies for protecting information by transforming it to unreadable form. It tries to encrypt the information in such a way that a third party who has access to the encrypted data cannot read it and cannot reconstruct, decrypt, the original information.

Steganography is the art and science of invisible communication (Morkel T. et al 2006). This is accomplished through hiding information in other objects in such a way that no one apart from the intended recipient knows of the information existence, thus hiding the existence of the communicated information (Morkel T. et al 2006). The word steganography is derived from the Greek words "stegos" meaning "cover" and "grafia" meaning "writing" (Moerland, T. 2001) defining it as "covered writing". In image steganography the information is hidden exclusively in images.

The idea of hiding information has long history. In Histories the Greek historian Herodotus writes of a nobleman, Histaeus, who needed to communicate with his son-in-law in Greece. He shaved the head of one of his most trusted slaves and tattooed the message onto the slave's scalp. When the slave's hair grew back the slave was dispatched with the hidden message (Silman, J. 2001).

The categories of steganography classified depending on the cover format which hides the secret data. The best file formats that are more suitable for steganography those with a high degree of redundancy. Redundancy can be defined as data or bits that could be

omitted without loss of meaning or function; repetition or superfluity of information. Figure (1.1) shows the four main categories of file formats that can be used for steganography as a cover for hiding any appropriate digital data as a secret.

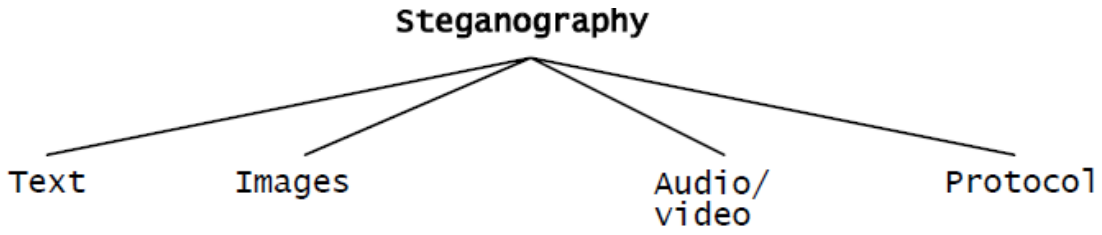


Figure 1.1: Categories of steganography (Morkel T. et al.2006)

Digital images have been chosen in this proposed work because they are the most popular cover objects for steganography, because of their proliferation especially on the Internet, and the large amount of redundant bits present in the digital representation. In the domain of digital images many different image file formats exist, most of them for specific applications. For these different image file formats, different steganographic algorithms exist.

Image steganography techniques can be divided into two groups: those in the Image Domain and those in the Transform Domain (Silman, J. 2001). Image – also known as spatial – domain techniques embed messages in the intensity of the pixels directly, while for transform – also known as frequency – domain, images are first transformed and then the message is embedded in the image (Morkel T. et al. 2006).

Image domain techniques apply bit insertion. The image formats that are most suitable for image domain steganography are lossless and the techniques are typically dependent on the image format.

Transform domain techniques and methods hide messages in more significant areas of the cover image making it more robust. Many transform domain methods are independent of the image format, those methods is suitable for lossy and lossless compression.

Least significant bit (LSB) insertion is a common, simple approach to embed information in a cover image on the image domain (Johnson, N.F. & Jajodia, S. 1998). The least significant bit (in other words, the 8th bit) of some or all of the bytes inside an image is changed to a bit of the secret message. When using a 24-bit image, a bit of each of the red, green and blue (RGB) color components can be used, since they are each represented by a byte. In other words, one can store 3 bits in each pixel. An 800 × 600 pixel image, can thus store a total amount of 1,440,000 bits or 180,000 bytes of embedded data. For example a grid for 3 pixels of a 24-bit image can be as follows: (Johnson, N.F. & Jajodia, S. 1998)

(00101101 00011100 11011100)

(10100110 11000100 00001100)

(11010010 10101101 01100011)

When the number 200, which binary representation is 11001000, is embedded into the least significant bits of this part of the image, the resulting grid is as follows :

(00101101 00011101 11011100)

(10100110 11000101 00001100)

(11010010 10101100 01100011)

List Significant bit (LSB) is the simplest form of bit insertion, it could be improved to List Significant bits (LSBs) to increase the capacity, And a several methods can be used to improve the security.

Image domain methods or Transform domain methods have different strong and weak points. Image domain methods are simple approach than transfer domain methods, and image domain least significant bit (LSB) technique in (RBG) colored images like PNG images and BMP images is most suitable for applications where the focus is on the amount of information to be transmitted and not on the secrecy of that information.(Morkel T. et al2006).

The following table compares least significant bit (LSB) insertion in RBG colored image and in GIF files, JPEG compression steganography: (Morkel T. et al2006)

	LSB in RBG colored images	**LSB in GIF**	**JPEG compression**
Invisibility	High*	Medium*	High
Payload capacity	High	Medium	Medium
Robustness against statistical attacks	Low	Low	Medium
Robustness against image manipulation	Low	Low	Medium
Independent of file format	Low	Low	Low
Unsuspicious files	Low	Low	High

* - Depends on cover image used

Table 1.1:Comparison of image steganography algorithms (Morkel T. et al2006)

On LSB principle there are many methods and techniques developed to increase the capacity and the quality of the cover image and to increase the security of the hidden data. This data could be anything, text or pixels of a secret image etc.

As shown on table 1.1, the highest payload capacity with high Invisibility can be achieved by using image domain cryptography, specifically by using LSBs Substitution in RGB colored images. This thesis proposed method will use LSBs to achieve high capacity.

When working especially with colored images as a secret message, which contains huge data, and in the same time it contains redundant data, it will be a good idea to reduce the unwanted or the not useful data in a way that doesn't affect the purpose that is needed from sending the secret image.

1.1 Problem Statement

The main goal of steganography is to hide the existence of any secret data from the eye of a third party, so the resultant stego-image must appear normal and not suspicious after it embeds the secret data. The stego-image must have an acceptable quality in comparison with its size. These problems will be harder to solve when the secret data is a colored image, because there is more data to embed.

Hiding the data is not enough to be sure that your data is safe, to improve the security it is very important to use some encryption with the hidden secret data.

Embedding the secret colored image data within the cover image will affect the quality of the stego-image. To increase the quality of the stego-image in general it is important to manipulate fewer bits of the cover with the secret data. Another important thing in image inside image steganography is how to retrieve the data in a right way to reconstruct the secret image. To reconstruct the secret image in a right way some information is needed more than the data of the image itself like the length of data.

1.2 Objectives

In order to send and retrieve a colored secret image safely on the Internet, it is important that the size of stego-image is not very big, the quality is good, and that it is to be sure that no third party can notice the existence of the secret image. It is very important to use a propitiate cryptography method to be sure that your data is safe even a third party knew about the existence of the secret data. In order to achieve that, the following goals should be completed:

- Reduce the secret image data by transfering it from RGB colored image to 8-bit or less indexed image.
- Discard sending any information about the resolution of the secret, and using a cover image has the same resolution as the secret image instead. That can be done by using the alpha channel to embed the extra data without using a cover with bigger resolution
- Encrypt the secret image while embedding it by dividing the secret data to blocks and using a portion of every block data (no extra bits) as an indicator and XORing it with bits of key to shuffle the data.

1.3 Motivation

The motivation behind developing image Steganography methods according to its use in various organizations to communicate between its members, as well as, it can be used for communication between members of the military or intelligence operatives or agents of companies to hide secret messages or in the field of espionage.

The main goal of using the Steganography is to avoid drawing attention to the transmission of hidden information. If suspicion is raised, then this goal that has been planned to achieve the security of the secret messages, because if the hackers noted any change in the sent message then this observer will try to know the hidden information inside the message.

Chapter 2: Literature Review

Literature Review

2.1 Image steganography: Data in Image

This section describes some previous works in LSB, many techniques have been produced in LSB principle to improve The capacity of the cover image without affecting the quality of the stego-image, or to improve the security of the embedded data.

(Nitin J. et al 2012) introduced new techniques for improving the stego-image, those techniques prevent the human eye from noticing the changes which happened on the cover image. That will help to avoid the detection of secret existence.

(Chi-Kwong & L.M. Cheng 2002) discuss how to improve the simple LSB substitution method of hiding secret messages into 8-bit "grayscale" images as the cover media, by using an optimal pixel adjustment process (OPAP) to enhance the image quality of the stego-image obtained by the simple LSB substitution method.

(Zhang, X & Zhou, S 2008) proposed an approach called multi bit assignment steganography for palette images, in which each gregarious color that possesses close neighboring color in the palette is used to represent several secret bits.

(Gandharba Swain, & S. K. Lenka, 2010) have discussed a double substitution algorithm for encrypting at sender side and decrypting at receiver side and the embedding process was at 7th and 8th bit positions alternatively

Mei-Yi Wu, Yu-Kun Ho, & Jia-Hong Lee (2004) showed that an image steganography with palette based images is suggested. The method is based on a palette adjustment scheme, which can iteratively embed one message bit into each pixel in a palette based

image. In each iteration, both the cost of removing an entry color in a palette and the benefit of generating a new one to replace it are calculated. If the maximal benefit exceeds the minimal cost, an entry color is replaced.

It is found that the fundamental statistics of natural images are altered by the hidden non-natural information (Alvaro Martin, Guillermo Sapiro, & Gadiel Seroussi, 2005). if they do not touch the bytes those carry the image features and embed in the other bytes then the problem can be solved. As LSB embedding is very common, many steganalysis tools are available for it (Sorina Dumitrescu, & Xiaolin, 2005). So LSB embedding is not secured now-a-days. New embedding techniques are to be proposed to the steganographic world. There is a large number of steganographic tools available on the Internet, a particular threat exists when criminals use steganography to conceal their activities within digital images in cyber space.

(Nitin J. et al.2012) chose to hide the data on edges inside the cover image, because altering the pixels in areas of the image where there are pixels that are most like their neighbors is much more noticeable by human eye. In objects boundaries there is a contrast between the object and the background, so altering those bits is safer than altering bits with no big contrast.

(Ashfaaq M. et al 2010) use the pixel indicator technique on RGB colored cover image that uses the same LSB principle to increase the security, and improve it. The technique uses least significant bits of one of the channels Red, Green or Blue as an indicator of data existence or data size in the other two channels.

(Juneja M. et al 2009) show a method for converting 24-bit RBG cover image to Palette-Based 8-bit image while simultaneously encoding the desired hidden information. The goal is to use 8-bit colored image as a cover because there size is small.

2.2 Image steganography: Image in Image

(Rawat D. & Bhandari V. 2013) discuss how to hide a 8-bit colored image (often called 8-bit truecolor) in 8-bit colored image, then how to improve the stego-image quality by hiding the secret image in 24-bit that is totally indistinguishable from the original image by the human eye.

Converting the secret colored image to the 8 bit color system will reduce the size of the secret image and reduce the bit number which needed to represent the colored image. This changes will not be suitable to all images and will affect the quality of some images more than others depending on what colors is the image consist of and if that colors have a close equivalents in the truecolor system.

In this method, the principle of indexed image to reduce the secret image data is being used. Using 8-bit indexed image instead of truecolor system for the secret image will improve the quality of it; choosing a 256 color of the original secret image colors will give a bitter quality than using the colors of the truecolor system to represent the secret image.

"*8 bit color system is very limited but true direct color system, it uses 8 bits to represent the R (red), G (green), and B (blue) components of every color, 3 bits for R, 3 bits for G, and 2 bit for B, which produce* ($2^3 \times 2^3 \times 2^2 = 256$) *256 color. 2 bits for B component has been chosen because the human eye is less sensitive to the blue component. This form of*

color graphics is often called 8-bit truecolor, as it does not use a palette at all, and is more similar to the 15-bit, 16-bit, and 24-bit truecolor modes." (Rawat D. & Bhandari V. 2013)

(Jassim F. 2012) whole idea is a about reducing the images data by (FMM) compression to find enough places to embed the secret image. In the beginning the Five Module Method (FMM) is used to compress the cover and the secret images. To ensure that no third party can get the secret, a stego-key produced by forward shifting either horizontally or vertically.

"*The main idea of FMM is to compress the image by transform the whole image into multiplies of five. It is known that, each of the R, G, and B channels in the color image are consist of pixel values varying from 0 to 255, after transform the whole pixels inside the image into numbers divisible by five then the new range of each channel will be from 0 to 51. FMM decrees the number of information without affecting the Human Visual System (HVS).*" (Jassim F. 2012)

(Shuchi Sharma, 2013) Proposed a technique to use the YIQ color space model and RGBA based cover image. RGB value's of the secret image pixel's is first converted into YIQ color space, which are then embedded into the least significant bits of the color pixel's of the cover image, as well as in the alpha channel. their results demonstrates that it is practically possible to hide an image in another image while maintaining acceptable image quality.

Chapter 3: Proposed Work

Proposed work

This chapter proposes a new method to hide a colored image inside 4 channel PNG image. The method suggests that the selected secret image is converted to an indexed image with custom color palette to reduce its size, the custom color palette is constructed using Lempel-Ziv-Welch (LZW) compression which is applied to the secret image to reduce its size so that the amount of data needed to be embedded with the secret image is reduced, after that the image is read as stream of bits, each byte is permutated according to the result of XORing the bits of a key with the 7^{th} and 8^{th} bit of each byte, the key is 128 bits in length, the byte is then embedded inside the 4 channels cover image that contains at least the same amount of pixel as the secret image after compression, figure (3.1) on the next page show the flow of the proposed method.

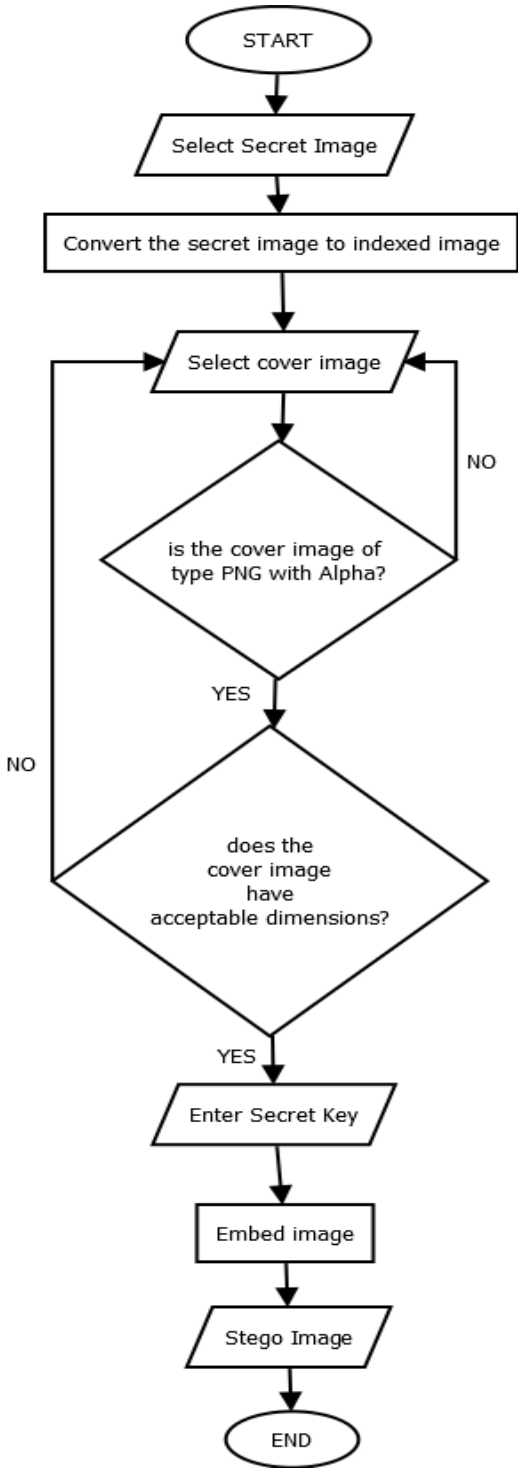


Figure 3.1. Chart of proposed model

3.1 Hiding secret image inside cover image

As described in Figure 3.1 and explained in the previous paragraph, the proposed methodology will go through the following steps:

- Choose the secret image:

Secret image could be of any image type, because it will be converted later to indexed image.

```
private void BrowseSecret_Click(object sender, EventArgs e)
{
    OpenFileDialog SecretFile = new OpenFileDialog();
    SecretFile.Filter = "Images Files|*.bmp;*.png;*.jpeg;";
    if (SecretFile.ShowDialog() == System.Windows.Forms.DialogResult.Cancel)
    {
        return;
    }
    SecretImage = Image.FromFile(SecretFile.FileName);
```

Figure 3.2. Code for Choosing secret image

After choosing the secret image, it will be converted to GIF indexed image, the size of the image will be reduced because GIFs are compressed using LZW compression algorithm.

- Converting and compressing the chosen secret image:

  To convert to GIF images the Lempel-Ziv-Welch compression algorithm is needed to build a color table for the image so that each color value is matched to a pixel. The LZW performs by scanning the image horizontally from top to left,

the pixel values are then encoded and mapped to segments for storage, each segment is a byte, the encoded pixel values doesn't always match the 8-bit size of the byte, so, the encoded values are inserted in the byte using the Little-Endian scheme, the beginning of each subsequent value is inserted in the least significant bit of the segment, these segments are stored as sub-blocks of maximum size 255 bytes, these sub-blocks contain a prefixed with a byte indicating the number of data bytes in the sub-block. The series of sub-blocks is terminated by an empty sub-block.

Images characterized of large areas of one color will be compressed better than those that do not have such characteristic. Thus, simple images like line art and logos are better saved as Gifs as they are likely to contain areas of single flat color which result in more compression efficiency. Photographs on the other hand contain much more colors and the LZW algorithm cannot compress them efficiently, figure 3.3 show the code for conversion.

```
SecretImage.Save(@"D:\Secret.gif", System.Drawing.Imaging.ImageFormat.Gif);
SecretImage = Image.FromFile(@"D:\Secret.gif");
SecretImageBox.Image = SecretImage;
```

Figure 3.3. Code for converting secret image

- Choose the cover image:

Choose cover image of the same size of the secret image or more in terms of number of pixels, it must be of type PNG with alpha channel; PNG image with alpha channel has four channels; Red, Green, Blue, and Alpha, it represent every pixel with 32-bit, 8-bit for every channel, these channels will be used to store the

permutated byte of the secret image, for example, in order to hide a secret image of size 512 × 512, a cover image of at least 512 × 512 is needed.

Each pixel in the cover image is used to embed a secret byte of the secret image intended to hide, each channel in the pixel will hold 2 bits of the secret byte regardless of the size of the secret image.

- Prepare the secret image to be embedded:

  To embed the secret image, the image is read as a blocks of bytes that contain the image as stream of bytes, each byte is called a secret block, and an additional 3 bytes are added to the front of the series to specify the size of the series which is going to be used during the extraction process.

- Embed the secret data within the cover as the following:
  - After reading the secret image as blocks of bytes, the three bytes are embedded completely in the first pixel which is going to contain the length of the stream, the rest of the bytes blocks is embedded with 1 byte payload per pixel, the embedding process is performed by replacing the least 2 significant bits of every channel (RGBA) with 2 bits of the stream, the decision of which 2 bits of the block go to which channel is made by the result of XORing 2 bits of the secret key determined by the index of the secret block with 2 bits from the stream. Figure 3.4 show the hiding algorithm.

```
public void Hide_Message()
{
    WSImages myImage = new WSImages();
    byte[] ImageBytes = myImage.GetImage("D:\\Secret.gif");
    SetLength(ImageBytes.Length);
    int[] KeyBits = GetKeyBits(SecretKey.Text);
    int EmbeddingOrder;
    Color tmpColor;
    int iHeight = CoverImageBitmap.Height;
    int iWidth = CoverImageBitmap.Width;

    int index, index1;
    bool completed = false;
    bool FirstKey, SecondKey, FirstImageBit, SecondImageBit;
    int[] ImageBits = new int[8];
    int w = 0;
    for (int i = 1; i < iWidth; i++)
    {
        for (int j = 1; j < iHeight; j++)
        {
            if (w == ImageBytes.Length)
            {
                completed = true;
                break;
            }

            ImageBits = Util.ConvertToBits(ImageBytes[w++]);
            tmpColor = CoverImageBitmap.GetPixel(i, j);
            tmpColor = Color.FromArgb(tmpColor.A, tmpColor.R, tmpColor.G, tmpColor.B);

            index = ((w-1) % KeyBits.Length);
            index1 = ((w) % KeyBits.Length);
            FirstKey = (KeyBits[index]==1)?true:false;
            SecondKey = (KeyBits[index1]==1)?true:false;

            FirstImageBit = Convert.ToBoolean(ImageBits[6]);
            SecondImageBit = Convert.ToBoolean(ImageBits[7]);

            EmbeddingOrder = GetEmbeddingOrder(FirstKey, SecondKey,
                FirstImageBit, SecondImageBit);
            if (EmbeddingOrder == 1)
            {
                tmpColor = ChangeColorValue(tmpColor, (ImageBits[0]), (ImageBits[1]),
                    (ImageBits[2]), (ImageBits[3]), (ImageBits[4]), (ImageBits[5]),
                    (ImageBits[6]), (ImageBits[7]));
```

Figure 3.4. Code for Hiding Algorithm

- To improve the security, a simple cryptography will be done while embedding the data, every secret block will be shuffled over the 4 channels depending on a 2 bit indicator (7^{th} and 8^{th} bits of each block).

The indicator will not use any extra bits, it will use the most 2 significant bits of the secret block, the indicator bits are XORed with bits of secret key and the result will decide the shuffle order for the block whose indicator bits are being used, figure 3.5 show the process to determine the embedding order.

```
public int GetEmbeddingOrder(bool KeyBit1, bool KeyBit2, bool ImageBit1, bool ImageBit2)
{
    if ((KeyBit1 ^ ImageBit1) && (KeyBit2 ^ ImageBit2)) //if result is 11
    {
        return 1;
    }
    else
    {
        if (!(KeyBit1 ^ ImageBit1) && !(KeyBit2 ^ ImageBit2)) // if result is 00
        {
            return 2;
        }
        else
        {
            if (!(KeyBit1 ^ ImageBit1) && (KeyBit2 ^ ImageBit2)) //if result is 01
            {
                return 3;
            }
        }
    }

    return 4; //base case if result is 10

}
```

Figure 3.5. Code for Embedding Order

Here is a screen shot of the application developed for simulation and testing the proposed method in this thesis.

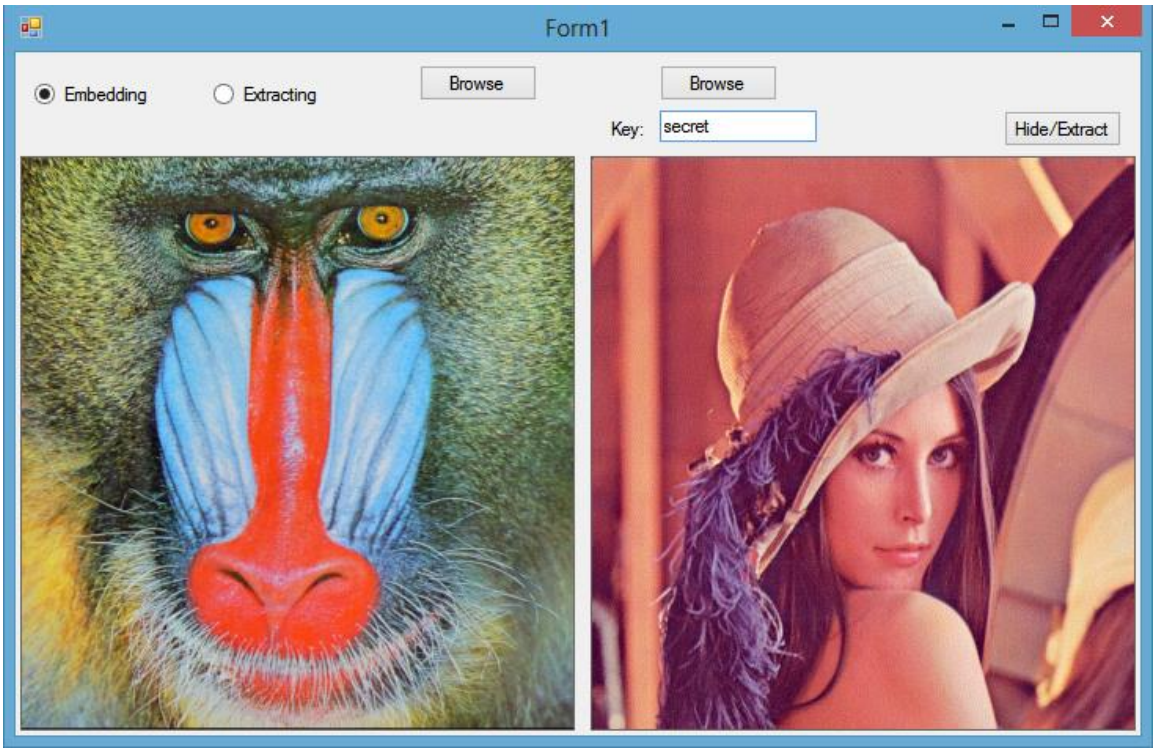


Figure 3.6. Application interface

3.2 Extracting secret image from stego image

After embedding the secret image and creating a stego image, the following steps are required in order to extract the secret image:

- Reading secret data length:

  The length of the secret data is extracted from the first pixel, the length determines the number of pixel to iterate through to extract a complete

image, this length is embedded in the first length using Red, Green and Blue only.

- Reading image pixels:

The first step toward extracting the secret image is to read the second pixels of the stego image and then separating each channel in a variable.

- XORing bits of the key with the 2 LSBs of alpha

To determine the order of extracting from each pixel, the 2 LSBs of the alpha channel in each pixel are xored with 2 bits of the key, the result determine which channel's LSBs should be extracted first.

- Building the bytes of secret image

After deciding the shuffle of each pixel, start building the bytes of the secret image to form a series of bytes which later converted to image.

- Converting bytes to image

After the secret series of bytes is ready, create an object of type bytes stream to create a file on the hard desk using this stream, the format of the file is GIF, given that the stego image was intact and data was extracted correctly using the correct key, the file should be properly read as GIF Image which is the secret image.

The following figure shows the extracting process.

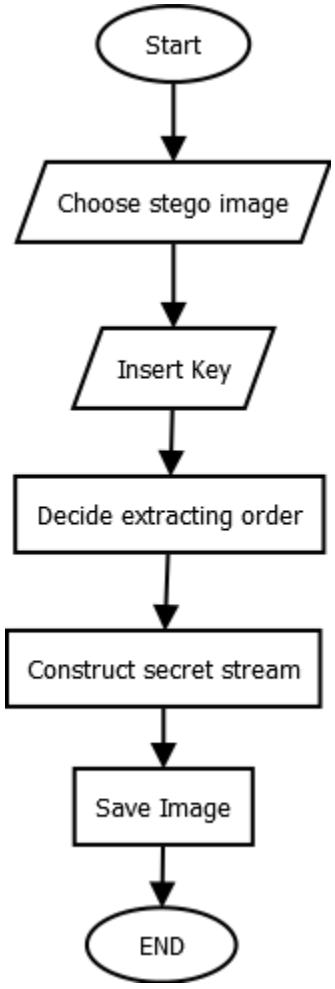


Figure 3.7. Extracting Secret Image Flowchart

3.3 Evaluation plan

As a result of the proposed enhanced methodology , we plan to evaluate the work as following :

1. A comparison between the cover image and the stego-image to calculate the noise on the secret image.
2. Comparing between the quality of the resultant image when converting it by using indexed method and truecolor system method.

Chapter 4: Experimental Results

Experimental Results

This chapter studies the results of the proposed method and compares them with the characteristics of truecolor (rawat,d.2013) method, in this regard, the tool was developed using Visual C# 2010, however, variety of tools been used to evaluate the performance of the proposed method on PNG images and the effect of conversion process on the secret images, the images of the truecolor are constructed by a software developed to performed the procedures as instructed in the paper discussed in the related work. The study is performed on 3 different images size 512 × 512 images using 3 different secret images for each.

4.1 Image Quality Test

This test measures the quality of the images compared to the original cover image, several samples been used to evaluate the performance of the proposed method and their relation to different color values, this test was done using a tool called DiffImg version 2.0.1 that compares the original cover image and the stego image.

Here are some statistical differences between the proposed method and the Truecolor method for each of the images used for testing after storing full capacity in each image, data is exposed in terms Standard Deviation (SD) of Root Mean Squared Error (RMSE) and Peak Signal to Noise Ratio (PSNR) using the following formulas:

Standard Deviation (SD):

The standard deviation is a statistical parameter used to describe how much variation or dispersion is there in a set of values from the average, in the following tables, the standard deviation is being calculated for images after converting the secret images.

It is calculated according to the following formula

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X - X`)^2} \quad \ldots\ldots (1)$$

Where:

X: the observed values of the sample items.

X`: the mean value of these observations.

N: the size of the sample.

This is the square root of the sample variance, which is the average of the squared deviations about the sample mean.

Root Mean Squared Error (RMSE):

The root-mean-square deviation or root-mean-square error (RMSE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed, in this section the RMSE values are discussed for secret images after conversion.

$$RMSE = \sqrt{\frac{1}{m \times n}\sum_{i=0}^{m}\sum_{j=0}^{n}(A_{ij} - B_{ij})^2} \quad \ldots\ldots (2)$$

Where:

A_{ij}: represents one pixel in the original image

B_{ij} represents one pixel in the stego-image

M*N represent the height and width of the image

Peak Signal to Noise Ratio:

PSNR in images is used as a quality metric that describes the quality of the image subjectively, in this section, the PSNR values are being reviewed for secret images.

$$PSNR = 20 \cdot \log_{10}\left(\frac{255}{RMSE}\right) \quad \text{.....(3)}$$

4.1.1 Quality Test on Secret Images

The study is performed on 3 different images size 512 × 512 images using 2 groups of secret images, each group is made of 3 different secret images, for each image in the first group which is used in the experiments, the following tables show the effect of conversion on secret images after being converted, the change in quality is measured using 3 statistical parameters, Standard Deviation (SD), Root Mean Squared Error (RMSE) and Peak Signal to Noise Ratio (PSNR).

The following figures 4.1 and 4.2 are sample images of the secret images before and after conversion.

Figure 4.1. Baboon after conversion using proposed method

Figure 4.2. Lena after conversion using Truecolor method

The degradation in quality after the conversion was relatively low using the proposed method, however, effects of conversion using the true color was relatively great.

Standard Deviation

The standard deviation is a statistical parameter used to describe how much variation or dispersion is there in a set of values from the average, in the following tables, the standard deviation is being calculated for images after converting the secret images.

Table 4.1 shows the SD after converting the secret image.

	GIF	Truecolor
Baboon	6.03402	4.71313
Lena	6.49100	4.86168
Peppers	6.19795	4.56489

Table 4.1. Standard Deviation after converting the secret image

SD values are preferred to be low which indicates that values are more similar, however, the SD for Truecolor method was lower which provided a better quality after hiding the secret image than the proposed method.

Root Mean Squared Error

The Root-Mean-Square Error (RMSE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed, in this section the RMSE values are discussed for secret images after conversion.In Table 4.2, the RMSE is calculated for each secret image in both methods

	GIF	Truecolor
Baboon	10.14378	8.17262
Lena	10.98884	8.41312
Peppers	10.29922	8.41214

Table 4.2. RMSE after converting the secret image

(RMSE) values are preferred to be low which indicates that values are more similar, however, the RMSE for Truecolor method was lower which provided a better quality after hiding the secret image than the proposed method.

Peak Signal To Noise Ratio:

PSNR in images is used as a quality metric that describes the quality of the image subjectively, in this section, the PSNR values are being reviewed for secret images. Table 4.3 show the PSNR values of the secret images after conversion using the proposed method and the true color method.

	GIF	Truecolor
Baboon	28.006	29.883
Lena	27.312	30.3098
Peppers	27.874	29.632

Table 4.3. PSNR of the secret images after conversion.

Peak Signal to Noise Ratio (PSNR) values are preferred to be higher which indicates that image quality is better, however, the PSNR for Truecolor method was similar to the proposed method.

4.1.2 Quality Test on Cover Pictures

The change in quality is measured using three statistical parameters, Standard Deviation (SD), Root Mean Squared Error (RMSE) and Peak Signal to Noise Ratio (PSNR), these parameters describe the amount of change by comparing the change of pixels in the cover image and the stego-image of both methods.

These are sample images of the results after hiding Baboon using the proposed method.

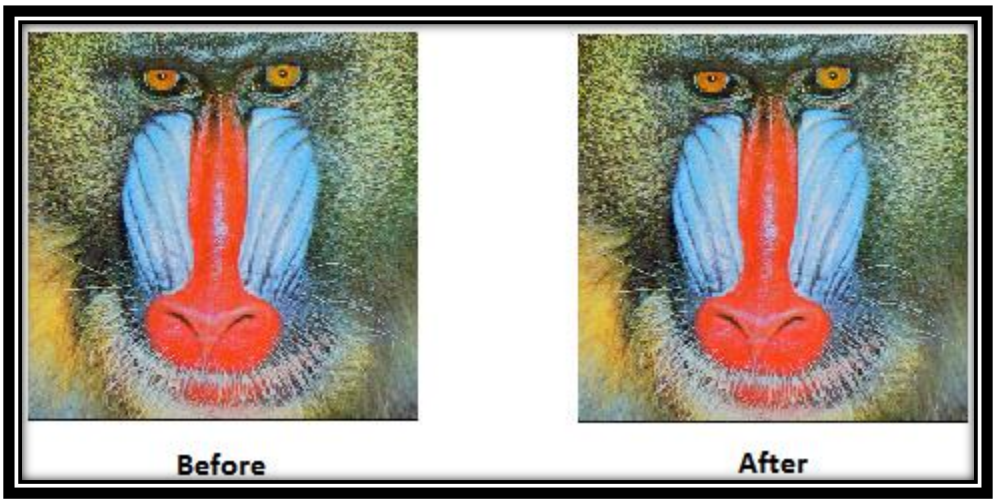


Figure 4.3. Image before and after embedding Lena.

Figure 4.4. Lena image before and after embedding Peppers

Figure 4.5. Peppers image before and after embedding Baboon

Standard Deviation

The standard deviation is a statistical parameter used to describe how much variation or dispersion is there in a set of values from the average, in the following tables, the standard deviation is being calculated for images after embedding.

The Standard Deviation values in table 4.4 are calculated after hiding Baboon inside the cover images.

	Proposed Method	Truecolor Method
Baboon	0.93979	1.36903
Lena	0.93504	1.38217
Peppers	0.92572	1.36026

Table 4.4. Standard Deviation of stego images after hiding Baboon

SD values are preferred to be low which indicates that values are more similar, however, the SD for the proposed method was lower which provided a better quality after hiding the secret image than the Truecolor method.

The Standard Deviation values in table 4.5 are calculated after hiding Lena inside the cover images.

	Proposed Method	Truecolor Method
Baboon	0.86760	1.30887
Lena	0.86146	1.31765
Peppers	0.85222	1.30901

Table 4.5. Standard Deviation of stego images after hiding Lena

SD values are preferred to be low which indicates that values are more similar, however, the SD for the proposed method was lower which provided a better quality after hiding the secret image than the Truecolor method.

The SD values in table 4.6 are calculated after hiding Peppers inside the cover images.

	Proposed Method	Truecolor Method
Baboon	0.88083	1.43834
Lena	0.87361	1.46199
Peppers	0.86359	1.40955

Table 4.6. Standard Deviation of stego images after hiding Peppers

Standard Deviation (SD) values are preferred to be low which indicates that values are more similar, however, the SD for the proposed method was lower which provided a better quality after hiding the secret image than the Truecolor method.

Root Mean Squared Error

The root-mean-square deviation or root-mean-square error (RMSE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed, in this section the RMSE values are discussed for stego images before and after embedding of secret images.

The Root Mean Squared Error values in table 4.7 are calculated after hiding Baboon inside the cover images.

	Proposed Method	Truecolor Method
Baboon	1.21285	2.16282
Lena	1.20370	2.16620
Peppers	1.19048	2.17664

Table 4.7. Root Mean Squared Error after hiding Baboon

Root Mean Squared Error (RMSE) values are preferred to be low which indicates that values are more similar, however, the RMSE for the proposed method was lower which provided a better quality after hiding the secret image than the Truecolor method

The Root Mean Squared Error values in table 4.8 are calculated after hiding Lena inside the cover images.

	Proposed Method	Truecolor Method
Baboon	1.04225	2.07197
Lena	1.03394	2.06536
Peppers	1.02231	2.10625

Table 4.8. Root Mean Squared Error after hiding Lena

Root Mean Squared Error (RMSE) values are preferred to be low which indicates that values are more similar, however, the RMSE for the proposed method was lower which provided a better quality after hiding the secret image than the Truecolor method

The Root Mean Squared Error values in table 4.9 are calculated after hiding Peppers inside the cover images.

	Proposed Method	Truecolor Method
Baboon	1.06390	2.25486
Lena	1.05365	2.27558
Peppers	1.04184	2.24333

Table 4.9. Root Mean Squared Error after hiding Peppers

Root Mean Squared Error (RMSE) values are preferred to be low which indicates that values are more similar, however, the RMSE for the proposed method was lower which provided a better quality after hiding the secret image than the Truecolor method

Peak Signal to Noise Ratio

PSNR in images is used as a quality metric that describes the quality of the image subjectively, in this section, the PSNR values are being reviewed for stego images.

The Peak Signal to Noise Ratio values in table 4.10 are calculated after hiding Baboon inside the cover images.

	Proposed Method Size of secret Image 168KB	Truecolor Method Size of secret Image 260KB
Baboon	46.454	41.430
Lena	46.520	41.417
Peppers	46.616	41.375

Table 4.10. Peak Signal to Noise values after hiding Baboon.

Peak Signal to Noise Ratio (PSNR) values are preferred to be higher which indicates that image quality is better, however, the PSNR for the proposed method was higher than the Truecolor method which indicates better quality.

The Peak Signal to Noise Ratio values in table 4.11 are calculated after hiding Lena inside the cover images.

	Proposed Method Size of secret Image 132KB	Truecolor Method Size of secret Image 260KB
Baboon	47.771	41.803
Lena	47.840	41.830
Peppers	47.939	41.660

Table 4.11. Peak Signal to Noise values after hiding Lena.

Peak Signal to Noise Ratio (PSNR) values are preferred to be higher which indicates that image quality is better, however, the PSNR for the proposed method was higher than the Truecolor method which indicates better quality.

The Peak Signal to Noise Ratio values in table 4.12 are calculated after hiding Peppers inside the cover images.

	Proposed Method Size of secret Image 136KB	Truecolor Method Size of secret Image 260KB
Baboon	47.592	41.068
Lena	47.676	40.989
Peppers	47.774	41.112

Table 4.12. Peak Signal to Noise values after hiding Peppers.

Peak Signal to Noise Ratio (PSNR) values are preferred to be higher which indicates that image quality is better, however, the PSNR for the proposed method was higher than the Truecolor method which indicates better quality.

The process of encrypting and embedding secret image (Baboon) inside these images left no perceptual effect that can be noticed with naked eyes; however, they will also study the effect of embedding other secret images for the same testing set.

4.1.3 Histograms for Proposed Method

The following histograms represent the distribution difference between stego-images before and after the embedding of the indexed secret images.

The numbers on the left (y-axis) represent the number of recurrence of a color value, the values on the x-axis are the values of the colors, each bar indicate the color value and how much it is repeated, the shadowed curve present the amount of changes that happened in the color value for the image after embedding.

Figure 4.6 Show the number of recurrence of a color value, the values of the colors, and how much it is repeated between the original image and stego image.

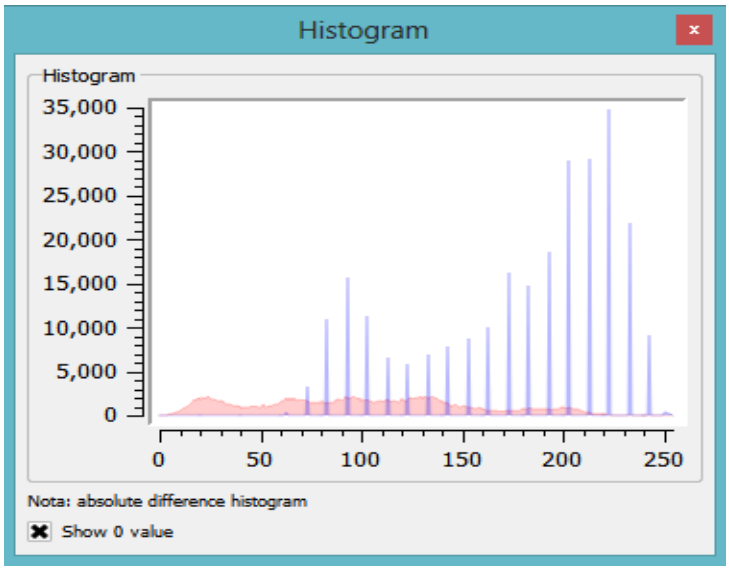


Figure 4.6. Histogram for Baboon inside Lena using the proposed method.

Figure 4.7 Show the number of recurrence of a color value, the values of the colors, and how much it is repeated between the original image and stego image.

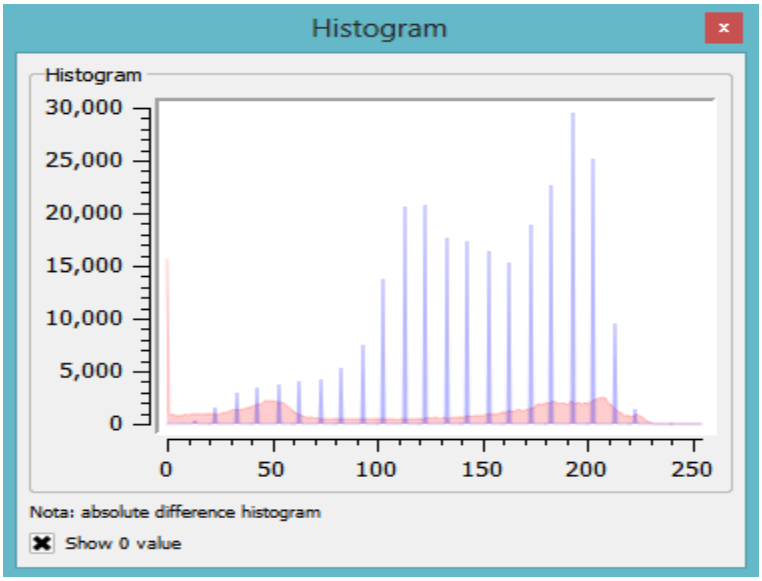


Figure 4.7. Histogram for Baboon inside Peppers using the proposed method.

Figure 4.8 Show the number of recurrence of a color value, the values of the colors, and how much it is repeated between the original image and stego image

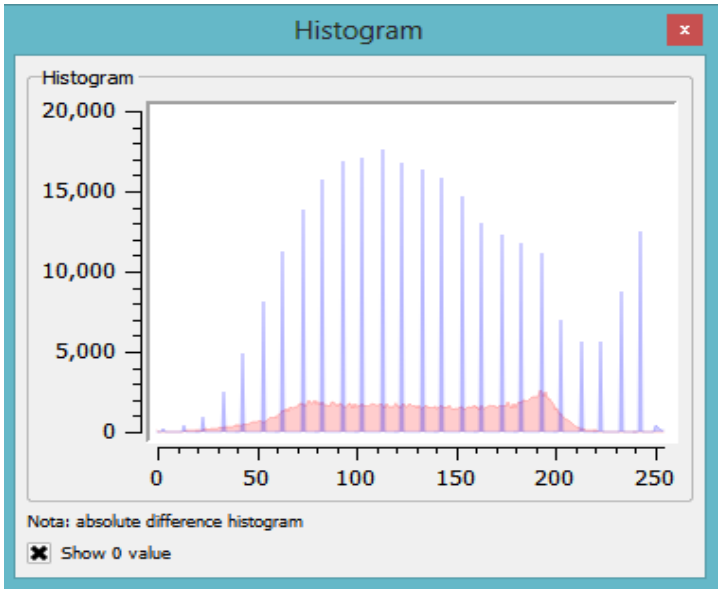


Figure 4.8. Histogram for Lena inside Baboon using the proposed method.

Figure 4.9 Show the number of recurrence of a color value, the values of the colors, and how much it is repeated between the original image and stego image.

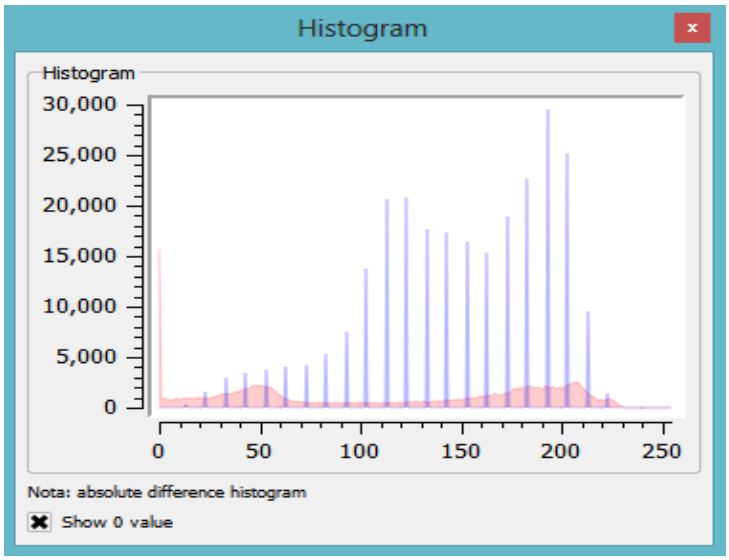


Figure 4.9. Histogram for Lena inside Peppers using the proposed method.

Figure 4.10 Show the number of recurrence of a color value, the values of the colors, and how much it is repeated between the original image and stego image

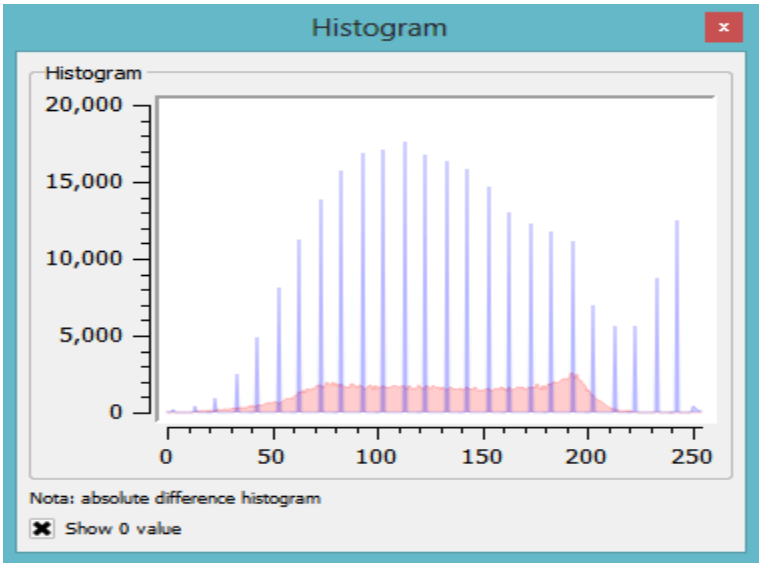


Figure 4.10. Histogram for Peppers inside Baboon using the proposed method.

Figure 4.11 Show the number of recurrence of a color value, the values of the colors, and how much it is repeated between the original image and stego image.

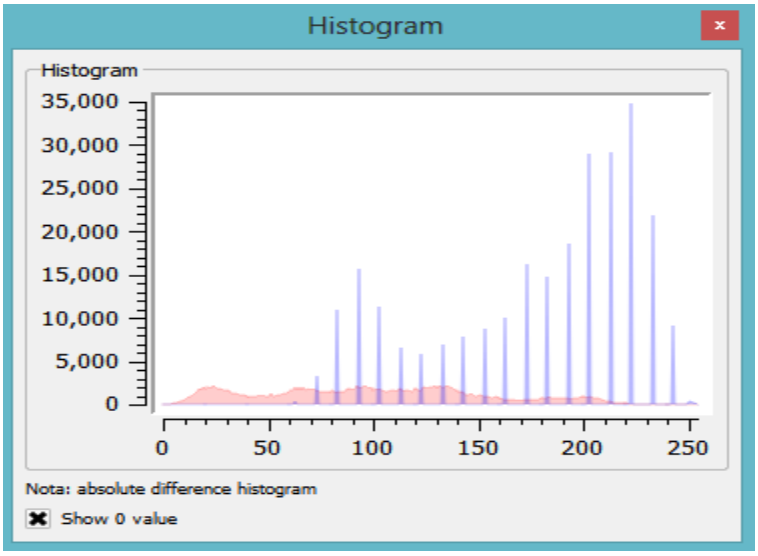


Figure 4.11. Histogram for Peppers inside Lena using the proposed method.

The following tables are direct comparison between the proposed method and the truecolor method for hiding lena, the results of the proposed method was better than the truecolor method.

	Proposed Method	Truecolor Method
Standard Deviation	0.86760	1.30887
RMS Error Deviation	1.04225	2.07197
PSNR	47.771	41.803

Table 4.13. Effects of embedding Lena inside Baboon

	Proposed Method	Truecolor Method
Standard Deviation	0.86146	1.31765
RMS Error Deviation	1.03394	2.06536
PSNR	47.840	41.830

Table 4.14. Effects of embedding Lena inside Lena

	Proposed Method	Truecolor Method
Standard Deviation	0.85222	1.30901
RMS Error Deviation	1.02231	2.10625
PSNR	47.939	41.660

Table 4.15. Effects of embedding Lena inside Beppers

After studying the results of the proposed method and comparing it with the true color method for the process of converting the secret image and embedding inside the stego image one can note that the perceptual difference in converted secret image in the proposed method was lower than the true color method but higher RMSE levels, however, the statistics for stego images in comparison to cover ones indicate that the proposed method was superior to true color method, this is due to the fact the reduction in size of converted secret image was greater in the proposed method than in the true color method.

Chapter 5: Conclusions

Conclusions

5.1 Overview

In this thesis a new method is proposed which suggests that the selected secret image is converted to an indexed image with custom color palette to reduce its size and then encrypted using key of 128 bits, the custom color palette is constructed using Lempel-Ziv-Welch (LZW) compression which is applied to the secret image to reduce its size so that the amount of data needed to be embedded with the secret image is reduced.

5.2 Conclusion

In conclusion, the results of the proposed method when compared to the results of the truecolor method indicate that the lower size of the secret image caused by converting it to indexed image provided better quality, this can be viewed in higher PSNR values and lower RMSE, this observation is caused by the fact that lower amount of bits are needed to be replaced in order to hide the secret image.

5.3 Future Work

In this thesis a method is proposed for image inside image steganography, this method can be the base for which other studies can emerge, the following list present some ideas for further studies:

1- Test the proposed method using larger secret images with the same size for cover image used in this thesis.

2- Use the proposed method to test the results of conversion and hiding using gray scale images as cover ones.

3- Combine the method with other techniques for compression to reduce the size of the secret image before embedding.

References

- Morkel T., Eloff J.H.P., & Olivier M.S., (2006)"An Overview of Image Steganography", University of Pretoria.
- Moerland, T. (2001), "Steganography and Steganalysis", Leiden Institute of Advanced Computing Science.
- Silman, J. (2001), "Steganography and Steganalysis: An Overview", SANS institure.
- Johnson, N.F. & Jajodia, S. (1998), "Exploring Steganography: Seeing the Unseen", Computer Journal.
- Nitin Jain, Sachin Meshram, &, Shikha Dubey,(2012). Image Steganography Using LSB and Edge Detection Technique.
- Chi-Kwong & L.M. Cheng (2002), Hiding data in images by simple LSB substitution.
- Xinpeng Zhang, Shuozhong Wang, & Zhenyu Zhou. (2008). Multibit Assignment Steganography in Palette Images. *IEEE Signal Processing Transactions*, 15, 553-556. http://dx.doi.org/10.1109/LSP.2008.2001117
- Gandharba Swain, & S.K.Lenka. (2010). Steganography-Using a Double Substitution Cipher. *International Journal of Wireless Communications and Networking*, 2(1), 35-39.
- Mei-Yi Wu, Yu-Kun Ho, & Jia-Hong Lee. (2004). An iterative method of palette-based image steganography. *Pattern Recognition Letters*, 25, 301 309. http://dx.doi.org/10.1016/j.patrec.2003.10.013.

- Alvaro Martin, Guillermo Sapiro, & GadielSeroussi. (2005). Is steganography natural. *IEEE Transactions on Image Processing*, 14(12), 2040-2050. http://dx.doi.org/10.1109/TIP.2005.859370
- Sorina Dumitrescu, & Xiaolin. (2005). A New Framework of LSB Steganalysis of Digital Media. *IEEE Transactions on Signal Processing*, 53(10), 3936-3947. http://dx.doi.org/10.1109/TSP.2005.855078.
- Ashfaaq M., Bosco J. & Rayappan B. (2010), Color Guided color Image Steganography, R. Amirtharajan, Sandeep Kumar Behera, Motamarri Abhilash Swarup.
- Juneja M., Sandhu P., & Walia E. (2009), Application of LSB Based Steganographic Technique for 8-bit color Images.
- Rawat D. & Bhandari V. (2013), A Steganography Technique for Hiding Image in an Image using LSB Method for 24 Bit Color Image.
- Jassim F.,(2013) Hiding Image in Image by Five Modulus Method for Image Steganography.
- Shuchi Sh , Uma K (2103) A High Capacity data – Hiding Technique Using Steganography
- Mandal P., Modern (2012) Steganographic technique: A survey on image steganography
- Dutta, P., Bhattacharyya, D., & Kim, T.-h. (2009). Data Hiding in Audio Signal: A Review. Kim International Journal of Database Theory and Application, 1-8.

- E. W. Weisstein, " MathWorld--A Wolfram Web Resource," Primitive Root, 2014. [Online]. Available: http://mathworld.wolfram.com/PrimitiveRoot.html. [Accessed 2014].
- M.Tommila,"Apfloat,"2013.[Online].Available:http://www.apfloat.org/prim.html. [Accessed 2014].
- K. F. M. Khodaei, "A New Adaptive Steganographic Method Using Optimal LSB Substitution and Pixel-Value Differencing," IET-Image Processing, pp. 1751-9659, 2012.

Appendix A: Software code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;
using System.Drawing.Imaging;

namespace Image_Steganography
{
    public partial class Form1 : Form
    {

        Image CoverImage;
        BitmapIP CoverImageIP;
        Bitmap CoverImageBitmap;

        Image SecretImage;
        Bitmap SecretImageBitmap;
        BitmapIP SecretImageIP;


        public Form1()
        {
            InitializeComponent();
        }

        private void BrowseCover_Click(object sender, EventArgs e)
        {
            OpenFileDialog CoverFile = new OpenFileDialog();
            CoverFile.Filter = "Images Files|*.bmp;*.png;*.jpeg; *.*;";
            if                    (CoverFile.ShowDialog()                    ==
System.Windows.Forms.DialogResult.Cancel)
```

```
            {
                return;
            }
            CoverImage = System.Drawing.Image.FromFile(CoverFile.FileName);
            CoverImageBox.Image = CoverImage;
            CoverImageBitmap = new Bitmap(CoverFile.FileName);
            CoverImageIP = new BitmapIP(CoverImageBitmap);

        }

        private void BrowseSecret_Click(object sender, EventArgs e)
        {
            OpenFileDialog SecretFile = new OpenFileDialog();
            SecretFile.Filter = "Images Files|*.bmp;*.png;*.jpeg; *.*;";
            if                    (SecretFile.ShowDialog()                    ==
System.Windows.Forms.DialogResult.Cancel)
            {
                return;
            }
            SecretImage = Image.FromFile(SecretFile.FileName);
            SecretImage.Save(@"D:\Secret.gif",
System.Drawing.Imaging.ImageFormat.Gif);
            SecretImage = Image.FromFile(@"D:\Secret.gif");
            SecretImageBox.Image = SecretImage;
        }

        public bool ValidateInputs()
        {
            if (SecretKey.Text.Length == 0)
            {
                MessageBox.Show("Please enter the key");
                return false;
            }
            if (CoverImageBox.Image == null)
            {
                MessageBox.Show("Please select cover image");
                return false;
            }
            if (SecretImageBox.Image == null && !Extracting.Checked)
```

```
            {
                MessageBox.Show("Please select secret image");
                return false;
            }


            return true;
        }

        private void Hide_Click(object sender, EventArgs e)
        {
            if (Embedding.Checked)
            {
                if (!ValidateInputs())
                    return;
                Hide_Message();
            }
            else
            {
                if (Extracting.Checked)
                {
                    if (SecretKey.Text.Length == 0)
                    {
                        MessageBox.Show("Please enter the key");
                        return;
                    }
                    Extract_Message();
                }
            }
        }

        public int GetEmbeddingOrder(bool KeyBit1, bool KeyBit2, bool ImageBit1,
bool ImageBit2)
        {
            if ((KeyBit1 ^ ImageBit1) && (KeyBit2 ^ ImageBit2)) //if result is 11
            {
                return 1;
            }
            else
```

```
            {
                if (!(KeyBit1 ^ ImageBit1) && !(KeyBit2 ^ ImageBit2)) // if result
is 00
                {
                    return 2;
                }
                else
                {
                    if (!(KeyBit1 ^ ImageBit1) && (KeyBit2 ^ ImageBit2)) //if
result is 01
                    {
                        return 3;
                    }
                }
            }

            return 4; //base case if result is 10

        }

        public int GetLength()
        {
            int iWidth = CoverImageIP.GetBitmap().Width;
            int iHeight = CoverImageIP.GetBitmap().Height;
            Color[,] ImageArray = CoverImageIP.GetImage2DArray();
            int Length;
            Color tmpColor = new Color();
            tmpColor = ImageArray[0, 0];
            tmpColor = Color.FromArgb(tmpColor.R, tmpColor.G, tmpColor.B);
            int RColor = tmpColor.R;
            int GColor = tmpColor.G;
            int BColor = tmpColor.B;
            int x = new int();
            int y = new int();
            int z = new int();
            x = GColor % 10;
            y = BColor % 10;
            z = RColor % 10;
```

```
            Length = Convert.ToInt32((RColor * 10000 + GColor * 100 +
BColor).ToString());

            return Length;
        }

        public void Extract_Message()
        {

            Color[,] ImageArray = CoverImageIP.GetImage2DArray();

            Color[] ImageArray1D = Convert2Dto1D(ImageArray);




            int iLen = GetLength();


            int iHeight = CoverImageBitmap.Height;
            int iWidth = CoverImageBitmap.Width;
            bool completed = false;
            int[] ImageBits = new int[8];
            int w = 0;
            StringBuilder sHiddenData = new StringBuilder();
            int[] KeyBits = GetKeyBits(SecretKey.Text);
            for (int i = 1; i < iWidth; i++)
            {
                for (int j = 1; j < iHeight; j++)
                {
                    if ((w) == iLen)
                    {
                        completed = true;
                        break;
                    }

                    sHiddenData.Append(GetStoredValue(CoverImageBitmap.GetPixel(i,
j), KeyBits, w));
```

```
                    w++;
                }
                if (completed)
                    break;
            }



            byte[] ImageBytes = GetBytesFromString(sHiddenData.ToString());

            WSImages myImage = new WSImages();


            MemoryStream memStream = new MemoryStream(ImageBytes);
            // Convert memory stream to a Bitmap
            try
            {
                Bitmap bm = new Bitmap(memStream);
                bm.Save("d:\\ExtractedSecret.gif",
System.Drawing.Imaging.ImageFormat.Gif);
                MessageBox.Show("Extracted Successfuly!");
                SecretImageBox.Image = bm;
            }
            catch
            {
                MessageBox.Show("Unable to extract image, please check key");
            }


        }

        public static byte[] GetBytesFromString(string bitString)
        {
            return Enumerable.Range(0, bitString.Length / 8).
                Select(pos => Convert.ToByte(
                    bitString.Substring(pos * 8, 8),
                    2)
                ).ToArray();
        }
```

```
string GetStoredValue(Color tmpColor, int[]KeyBits, int w)
{
    int index = ((w) % KeyBits.Length);
    int index1 = ((w + 1) % KeyBits.Length);
    bool FirstKey = (KeyBits[index] == 1) ? true : false;
    bool SecondKey = (KeyBits[index1] == 1) ? true : false;

    string sStoredValue = "";
    byte bRed, bGreen, bBlue, bAlpha;

    bRed = tmpColor.R;
    bGreen = tmpColor.G;
    bBlue = tmpColor.B;
    bAlpha = tmpColor.A;
    int[] iRedBits, iGreenBits, iBlueBits, iAlphaBits;

    iRedBits = Util.ConvertToBits(bRed);
    iGreenBits = Util.ConvertToBits(bGreen);
    iBlueBits = Util.ConvertToBits(bBlue);
    iAlphaBits = Util.ConvertToBits(bAlpha);

    int iFirstBit;
    int iSecondBit;
    int iThirdBit;
    int iFourthBit;
    int iFifthBit;
    int iSixthBit;
    int iSeventhBit;
    int iEightethBit;

    bool first, second;
    if (iAlphaBits[7] == 1)
        first = true;
    else
        first = false;

    if (iAlphaBits[6] == 1)
        second = true;
```

```
else
    second = false;


int EmbeddingOrder=GetEmbeddingOrder(FirstKey,SecondKey,first,second);


    iFirstBit = iRedBits[7];
    iSecondBit = iRedBits[6];


    iThirdBit = iGreenBits[7];
    iFourthBit = iGreenBits[6];

    iFifthBit = iBlueBits[7];
    iSixthBit = iBlueBits[6];

    if (EmbeddingOrder == 1)
    {

        iFirstBit = iRedBits[7];
        iSecondBit = iRedBits[6];


        iThirdBit = iGreenBits[7];
        iFourthBit = iGreenBits[6];

        iFifthBit = iBlueBits[7];
        iSixthBit = iBlueBits[6];
    }
if (EmbeddingOrder == 2)
{

    iFirstBit = iGreenBits[7];
    iSecondBit = iGreenBits[6];


    iThirdBit = iRedBits[7];
    iFourthBit = iRedBits[6];
```

```
            iFifthBit = iBlueBits[7];
            iSixthBit = iBlueBits[6];
        }
        if (EmbeddingOrder == 3)
        {

            iFirstBit = iRedBits[7];
            iSecondBit = iRedBits[6];


            iThirdBit = iBlueBits[7];
            iFourthBit = iBlueBits[6];

            iFifthBit = iGreenBits[7];
            iSixthBit = iGreenBits[6];
        }
        if (EmbeddingOrder == 4)
        {

            iFirstBit = iBlueBits[7];
            iSecondBit = iBlueBits[6];


            iThirdBit = iGreenBits[7];
            iFourthBit = iGreenBits[6];

            iFifthBit = iRedBits[7];
            iSixthBit = iRedBits[6];
        }

        iSeventhBit = iAlphaBits[7];
        iEightethBit = iAlphaBits[6];


        sStoredValue = iFirstBit.ToString() + iSecondBit.ToString() +
iThirdBit.ToString() + iFourthBit.ToString() + iFifthBit.ToString() +
iSixthBit.ToString() + iSeventhBit.ToString() + iEightethBit.ToString();
```

```
            return sStoredValue;
        }

        public void SetLength(int TxtLength)
        {

            int iWidth = CoverImageIP.GetBitmap().Width;
            int iHeight = CoverImageIP.GetBitmap().Height;
            Color[,] ImageArray = CoverImageIP.GetImage2DArray();
            Color tmpColor = new Color();
            tmpColor = ImageArray[0, 0];
            tmpColor = Color.FromArgb(tmpColor.R, tmpColor.G, tmpColor.B);
            int RColor = tmpColor.R;
            int GColor = tmpColor.G;
            int BColor = tmpColor.B;

            int[] x = new int[6];

            for (int ii = 0; ii < 6; ii++)
                x[ii] = 0;


            string TxtLength12 = "000000";
            TxtLength12 = TxtLength12 + TxtLength.ToString();
            x[0]                                                    =
Convert.ToInt32(TxtLength12.ToString().ElementAt(TxtLength12.Length     -
1).ToString());
            x[1]                                                    =
Convert.ToInt32(TxtLength12.ToString().ElementAt(TxtLength12.Length     -
2).ToString());
            x[2]                                                    =
Convert.ToInt32(TxtLength12.ToString().ElementAt(TxtLength12.Length     -
3).ToString());
            x[3]                                                    =
Convert.ToInt32(TxtLength12.ToString().ElementAt(TxtLength12.Length     -
4).ToString());
```

```
            x[4]                                                    =
Convert.ToInt32(TxtLength12.ToString().ElementAt(TxtLength12.Length     -
5).ToString());
            x[5]                                                    =
Convert.ToInt32(TxtLength12.ToString().ElementAt(TxtLength12.Length     -
6).ToString());
            RColor = Convert.ToInt32(x[5] + "" + x[4]);
            GColor = Convert.ToInt32(x[3] + "" + x[2]);
            BColor = Convert.ToInt32(x[1] + "" + x[0]);
            tmpColor = Color.FromArgb(RColor, GColor, BColor);
            CoverImageBitmap.SetPixel(0, 0, tmpColor);



        }

        public Color[] Convert2Dto1D(Color[,] ImageArray)
        {
            int iWidth = ImageArray.GetUpperBound(0) + 1;
            int iHeight = ImageArray.GetUpperBound(1) + 1;

            Color[] ConvertedArray = new Color[iWidth * iHeight];
            int iIndex = 0;

            for (int i = 0; i < iWidth; i++)
            {
                for (int j = 0; j < iHeight; j++)
                {
                    ConvertedArray[iIndex] = ImageArray[i, j];
                    iIndex++;
                }
            }
            return ConvertedArray;

        }

        Color ChangeColorValue(Color tmpColor, int iFirstBit, int iSecondBit, int
iThirdBit, int iFourthBit, int iFifthBit, int iSixthBit, int iSeventhBit, int
iEightethBit)
        {
```

```
        byte bRed, bGreen, bBlue, bAlpha;

        bRed = tmpColor.R;
        bGreen = tmpColor.G;
        bBlue = tmpColor.B;
        bAlpha = tmpColor.A;
        int[] iRedBits, iGreenBits, iBlueBits, iAlphaBits;

        iRedBits = Util.ConvertToBits(bRed);
        iGreenBits = Util.ConvertToBits(bGreen);
        iBlueBits = Util.ConvertToBits(bBlue);
        iAlphaBits = Util.ConvertToBits(bAlpha);

        iRedBits[7] = iFirstBit;
        iRedBits[6] = iSecondBit;

        iGreenBits[7] = iThirdBit;
        iGreenBits[6] = iFourthBit;

        iBlueBits[7] = iFifthBit;
        iBlueBits[6] = iSixthBit;

        iAlphaBits[7] = iSeventhBit;
        iAlphaBits[6] = iEightethBit;

        bRed = Convert.ToByte(ReturnBitStr(iRedBits), 2);
        bGreen = Convert.ToByte(ReturnBitStr(iGreenBits), 2);
        bBlue = Convert.ToByte(ReturnBitStr(iBlueBits), 2);
        bAlpha = Convert.ToByte(ReturnBitStr(iAlphaBits), 2);

        Color EncodedColor = Color.FromArgb(bAlpha, bRed, bGreen, bBlue);

        return EncodedColor;
    }

    private string ReturnBitStr(int[] iBits)
    {
        string s = "";
```

```
    for (int i = 0; i <= iBits.GetUpperBound(0); i++)
    {
        s = s + iBits[i];
    }

    return s;
}

public Color[,] Conver1DTo2D(Color[] ImageArray, int iColNumber)
{
    int iWidth = ImageArray.Length / iColNumber;

    Color[,] ConvertedArray = new Color[iWidth, iColNumber];


    int x = 0;
    int y = 0;
    for (int i = 0; i < ImageArray.Length; i++)
    {
        if ((i % iColNumber == 0) && (i != 0))
        {
            y = 0;
            x++;
        }

        ConvertedArray[x, y] = ImageArray[i];
        y++;
    }

    return ConvertedArray;

}

public int[] GetKeyBits(string key)
{
    System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();

    byte[] KeyBytes = encoding.GetBytes(key);
```

```
    StringBuilder sb = new StringBuilder();
    foreach (byte b in Encoding.Unicode.GetBytes(key))
    {
        sb.Append(Convert.ToString(b, 2));
    }

    string temp = sb.ToString();
    int[] KeyBits = new int[temp.Length];
    for (int i = 0; i < temp.Length; i++)
    {
        if (temp.ElementAt(i) == '1')
            KeyBits[i] = 1;
        else
            KeyBits[i] = 0;
    }


    return KeyBits;
}

public void Hide_Message()
{
    WSImages myImage = new WSImages();
    byte[] ImageBytes = myImage.GetImage("d:\\Secret.gif");
    SetLength(ImageBytes.Length);
    int[] KeyBits = GetKeyBits(SecretKey.Text);
    int EmbeddingOrder;
    Color tmpColor;
    int iHeight = CoverImageBitmap.Height;
    int iWidth = CoverImageBitmap.Width;

    int index, index1;
    bool completed = false;
    bool FirstKey, SecondKey, FirstImageBit, SecondImageBit;
    int[] ImageBits = new int[8];
    int w = 0;
    for (int i = 1; i < iWidth; i++)
    {
```

```
                for (int j = 1; j < iHeight; j++)
                {
                    if (w == ImageBytes.Length)
                    {
                        completed = true;
                        break;
                    }

                    ImageBits = Util.ConvertToBits(ImageBytes[w++]);
                    tmpColor = CoverImageBitmap.GetPixel(i, j);
                    tmpColor = Color.FromArgb(tmpColor.A, tmpColor.R, tmpColor.G,
tmpColor.B);

                    index = ((w-1) % KeyBits.Length);
                    index1 = ((w) % KeyBits.Length);
                    FirstKey = (KeyBits[index]==1)?true:false;
                    SecondKey = (KeyBits[index1]==1)?true:false;

                    FirstImageBit = Convert.ToBoolean(ImageBits[6]);
                    SecondImageBit = Convert.ToBoolean(ImageBits[7]);

                    EmbeddingOrder = GetEmbeddingOrder(FirstKey, SecondKey,
FirstImageBit, SecondImageBit);
                    if (EmbeddingOrder == 1)
                    {
                        tmpColor = ChangeColorValue(tmpColor, (ImageBits[0]),
(ImageBits[1]), (ImageBits[2]), (ImageBits[3]), (ImageBits[4]), (ImageBits[5]),
(ImageBits[6]), (ImageBits[7]));

                    }
                    else
                    {
                        if (EmbeddingOrder == 2)
                        {
                            tmpColor = ChangeColorValue(tmpColor, (ImageBits[2]),
(ImageBits[3]), (ImageBits[0]), (ImageBits[1]), (ImageBits[4]), (ImageBits[5]),
(ImageBits[6]), (ImageBits[7]));
                        }
                        else
```

```
                    {
                        if (EmbeddingOrder == 3)
                        {
                            tmpColor = ChangeColorValue(tmpColor,
(ImageBits[0]), (ImageBits[1]), (ImageBits[4]), (ImageBits[5]), (ImageBits[2]),
(ImageBits[3]), (ImageBits[6]), (ImageBits[7]));
                        }
                        else
                        {
                            if (EmbeddingOrder == 4)
                            {
                                tmpColor = ChangeColorValue(tmpColor,
(ImageBits[4]), (ImageBits[5]), (ImageBits[2]), (ImageBits[3]), (ImageBits[0]),
(ImageBits[1]), (ImageBits[6]), (ImageBits[7]));
                            }
                        }
                    }
                    CoverImageBitmap.SetPixel(i, j, tmpColor);
                }
                if (completed)
                    break;
            }

            CoverImageBitmap.Save("d:\\Stegno.png");


        }
    }
}
```