MEU جـامـعـة الـشـرق الأوسـط MIDDLE EAST UNIVERSITY

Text in Image Steganograghy Using LSD Method

نص داخل صورة باستخدام المنزلة العشرية الاصغر

By

Ahmad Jalal Ahmad

Supervisor

Dr. Oleg Victorov

Submitted in Partial Fulfillment of the Requirements for

the Master Degree in Computer Science

Department of Computer Science

Faculty of Information Technology

Middle East University

Amman, Jordan

April 2014

إقرار تفويض

اناأحمد جلال احمد أفوض جامعة الشرق الاوسط بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الافراد عند طلبها.

التوقيع: [illegible]

التاريخ: ٧/٤/ ٢٠١٤

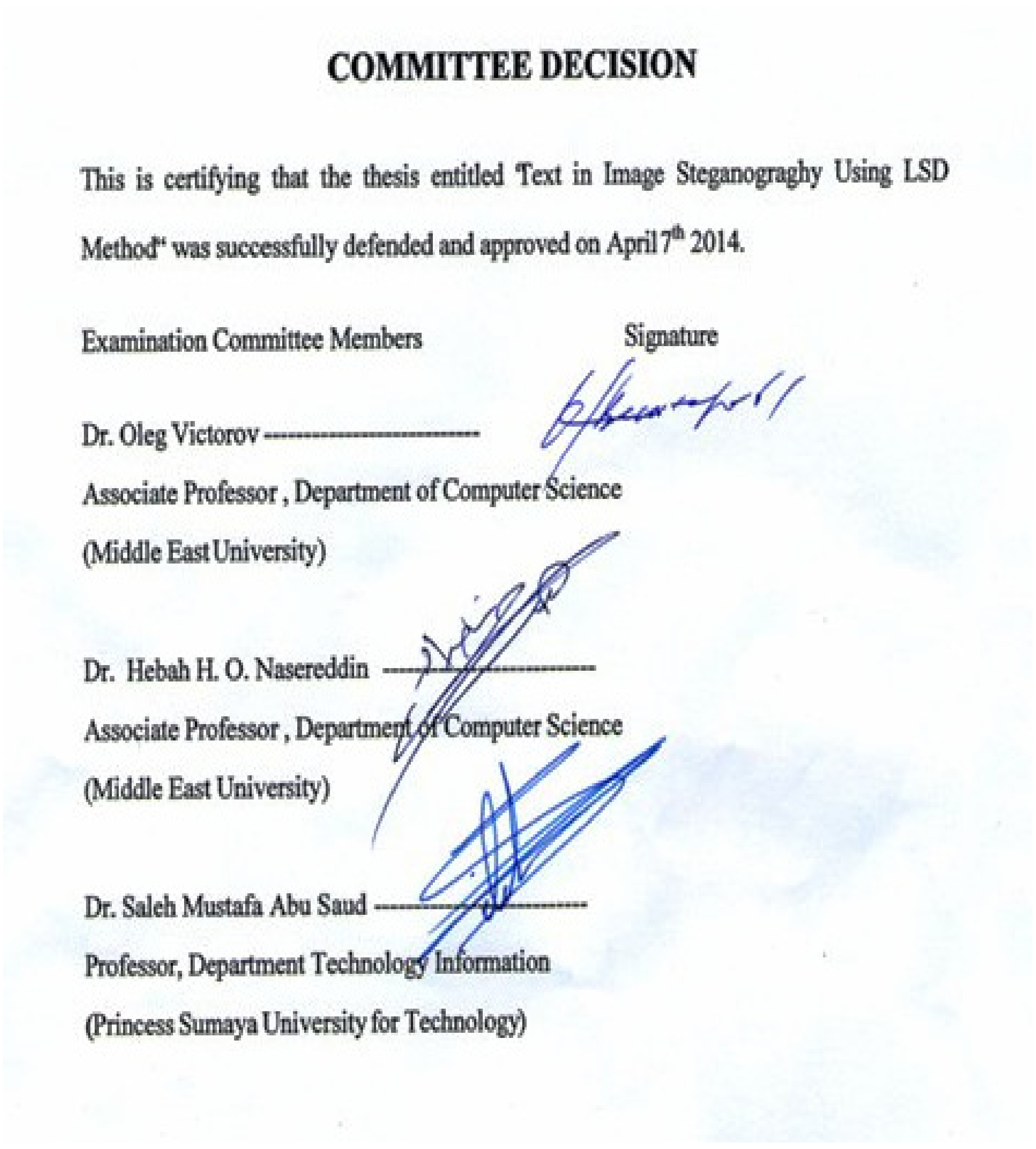
Authorization statement

I am Ahmad JalalAhmad; I Authorize the Middle East University to supply

A copy of my Thesis to libraries, establishments or individuals upon their request.

Signature: [illegible]

Date: 2011/4/7



COMMITTEE DECISION

This is certifying that the thesis entitled 'Text in Image Steganograghy Using LSD Method" was successfully defended and approved on April 7th 2014.

Examination Committee Members	Signature
Dr. Oleg Victorov --------------------------	[illegible]
Associate Professor , Department of Computer Science	
(Middle East University)	
Dr. Hebah H. O. Nasereddin --------------------------	[illegible]
Associate Professor , Department of Computer Science	
(Middle East University)	
Dr. Saleh Mustafa Abu Saud --------------------------	[illegible]
Professor, Department Technology Information	
(Princess Sumaya University for Technology)	

DEDICATION

This thesis is dedicated to all the people who never stopped believing in me

To my greatfather who never stopped supporting me during the journey of my life, to the father that made me the man I am.

To my greatmother who raised me with passion.

To my brotherand sister.

To my lovely wife, who cheered up my life.

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 Oleg Victorov for his 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 Falih and my friends for their support throughout my academic journey and all of my family members.

Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	**Meaning**
AES	Advanced Encryption Standard
CO	Cover Object
ASCII	American Standard Code for Information Interchange.
DCT	Discrete Cosine Transform
DH	Data Hiding
LSB	Least Significant Bit
LSD	Least Significant Digit
LSHD	Least Significant Hexadecimal Digit
PSNR	Peak Signal to Noise Ratio
RMSE	Root Mean Squared Error
SD	Standard Deviation
SO	Stego Object

Abstract

Steganography is the art of concealing messages or information within other non-secret text or data. Various steganography techniques have been proposed in literature. The Least significant Bit (LSB) steganography is one such technique in which least significant bit of the image is replaced with data bit. As this method is exposed to steganalysis, efforts are being put together in order to make it more secure. This study propose a new technique for image steganography, that uses the Least Significant Digit (LSD) to store text data inside colored images, each character is presented using its ASCII code which is made of three digits, the proposed technique use the RGB values of each pixel which enhance the ability of storing more data than previous known techniques, the embedding is achieved by replacing the LSD of each color of the pixel by a digit of the character ASCII representation, the results show that the value of colors effect the amount of data one is able to store in the image, the obtained results of the proposed study shows that the use of encryption in steganography using the proposed method does not affect the image quality. It is also worth to mention that the storage capacity is better than LSB and DCT.

الخلاصة

الاختزال هو فن إخفاء الرسائل أو المعلومات السرية داخل النص أو بيانات غير سرية أخرى تقنيات اختزال المعلومات المقترحة في هذا المجال متعددة، من أبرز التقنيات المستخدمة هي اختزال المعلومات في الجزء الاقل اهمية من الصورة بما يعرف باسم .Least Significant Bit (LSB) في مثل هذه التقنية يتم استبدال بعض من الاجزاء الأقل أهمية من الصورة مع جزء من البياتات المراد اخفاءها داخل الصورة. من الممكن ان يتعرض هذا الأسلوب للكشف اذا كان ضعيفا او بدائيا، وبناءً على ذلك، يجري حاليا وضع الجهود معا من أجل جعلها أكثر أمانا و اقل عرضة للكشف من قبل الجهات الخارجية. احد الطرق هو تشفير البيانات الخام قبل تضمينها في الصورة وعلى الرغم من أن عملية التشفير يزيد من تعقيد الوقت، ولكن في نفس الوقت يوفر أمان أعلى كذلك .تقترح هذه الدراسة تقنية جديدة لإخفاء المعلومات داخل صورة، حيث تم استخدام أقل الأرقام أهمية في الصورة (LSD) لتخزين بيانات نصية داخل الصور الملونة الحاملة للنص السري، ويقدم كل حرف باستخدام رمز ASCIIوالتي هي مصنوعة من ثلاثة أرقام ، والتقنية المقترحة استخدامت قيم RGB حيث ان كل بكسل تعزز القدرة على تخزين بيانات أكثر من التقنيات المعروفة سابقاً، ويتحقق التضمين عن طريق استبدال LSD من كل لون من بكسل بتمثيل ASCIIالخاص به ، فقد بينت النتائج أن قيمة الألوان تؤثر على كمية أحد البيانات قادرة على تخزين في الصورة.

كما اظهرت النتائج التي تم الحصول عليها من الدراسة المقترحة أن استخدام التشفير في اختزال المعلومات باستخدام الطريقة المقترحة لا يؤثر على جودة الصورة ومن الجدير بالذكر أن LSB توفر سعة تخزين أفضل من طرق اخرى مذكورة في الدراسة.

CHAPTER ONE

Chapter 1 : Introduction

1.1 Preface

Information security has two approaches. The first approach is called data encryption and it deals with changing data and making it unreadable by performing some data transformation. In such case one knows that a secret data exists but he is not able to read it (Guillermo, 2005).

The second approach is called steganography; it refers to the technique used to hide data or secret messages inside some other object called the cover. This can be done by changing some of the object's properties. The purpose of steganography is to set up a secret communication between sender and receiver so that any person in the middle cannot find out about the connection existence. An attacker is not able to detect any information about the hidden data only by looking at the cover file (Morkel T. et al 2006). This puts some limitations on the amount of data to hide inside a certain object and this is simply because the noise done to the cover file has to be minimal. The word steganography, derived from Greek language and literally means covered writing. It includes secret communications methods that make it hard to detect hidden data (Moerland, T. 2001).

The first known use of steganography dates back to 440 B.C, a famous story of Herodotus when he shaved the head of one of his slaves and tattooed it with a secret message, after the hair grew again the message was invisible so the slave could carry the message easily and it can be retrieved by shaving his head again (Yilmaz, 2003).

1.2 Classifications of Steganography

Almost all digital file formats can be used for steganography, but the formats that are more suitable are those which have a high degree of redundancy. Redundancy can be defined as the bits of an object that provide accuracy far greater than necessary for the object's use and display (Currie, D.L. et al., 1996).

The redundant bits of an object are those bits that can be altered without the alteration being detected easily. Image and audio files especially comply with this requirement, while research has also uncovered other file formats (MPEG, GIF….) that can be used for information hiding (Anderson, et al., 1998).

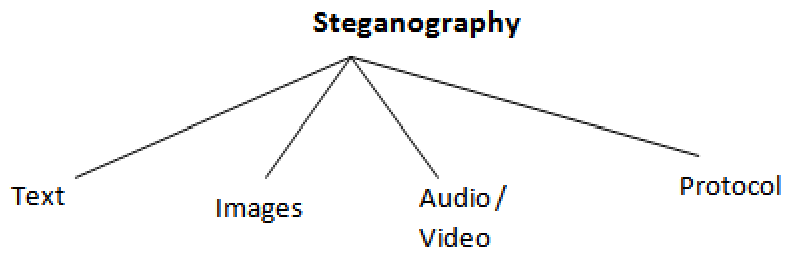


Figure 1.1. Main Categories of Steganography (Morkel T. et al.2006)

1.2.1 Text Steganography

Hiding secret information in text is historically the most important method of steganography. An obvious method was to hide a secret message in every *n-th* letter of every word of a text message. However, the usage of Internet and the different digital file formats have decreased text-in-text method importance (Nasereddin & Al Farzaeai, 2010).

1.2.2 Images Steganography

Hiding data, or, more appropriately, the art of hiding secret information within images is going to be major concern of this proposed study. It is basically achieved by replacing unused bits of a file with bits of hidden messages. Steganography takes advantage of these areas, replacing them with information, the files can then be exchanged without anyone knowing what really lies inside them (Morkel T. et al 2006).

An image might contain a private letter to a friend, a secret message, or a very dangerous terrorist hit list. A recording of a short sentence might contain company's plans for a secret new product. In modern steganography images are represented in computers as an array of light intensities at various points (or pixels). If a colored image is used, then there is such an array for each of the three primary colors: red, green and blue (RGB). Colored image is obtained by superposing these three arrays; each pixel is the combination of these three colors (Ashfaaq M. et al 2010).

Since computer files, images, etc. do not use all of the bits inside the file to store the data. One could replace the least significant bit of the original image with the secret bits and the image will not be distorted. For example, using most popular image size (640×480 and 256 colors) you can hide 300 KB data (Rabah K., 2004).

1.2.3 **Audio Steganography**

To hide information in audio files similar techniques are used as for image files. One different technique unique to audio steganography is masking, which exploits the properties of the human ear to hide information unnoticeably. A faint, but audible, sound becomes inaudible in the presence of another louder audible sound (Currie, D.L. & Irvine, C.E., 1996).

Such property create a channel in which to hide information. Although nearly equal to images in steganographic potential, the larger size of meaningful audio files makes them less popular than images (Artz, D., 2001).

1.3 Background of the Problem

Hiding data, or more appropriately, the art of hiding information within images is going to be the major concern in this study, it is basically achieved by replacing unused bits of a file or by manipulating bits that has low impact on the file with bits of hidden messages. Steganography takes advantage of these areas, replacing them with information (Currie, D.L. & Irvine, C.E., 1996).

The files can then be exchanged without anyone knowing what really lies inside of them. An image of might contain a private letter to a friend, a secret message, a very dangerous terrorist hit list. A recording of a short sentence might contain your company’s plans for a secret new product. In modern steganography images are represented in computers as an array of numbers that represent light intensities at various points (or pixels). If a color image is used, then there is such array for each of the three primary colors, red, green and blue (RGB). Colored image is obtained by superposing these three arrays; each pixel is the sum of these three colors. Since computer files, images… etc. do not use all of the bits inside the file to store the data, an idea of data hiding comes about. One could replace the least significant bit of the original image with the secret bits and the image will not be distorted. For example, using most popular image size (640×480 and 256 colors) can hide 300KB (Kefa Rabah, 2004), in the proposed method we used 512 × 512 and 256 colors to hide 256KB.

There are many algorithms today used to encode text into images, a famous one uses the concept of the Least Significant Bit and it's widely used even in commercial applications. Usually gray colors are used as special case in order to bind it with a certain message and still make it invisible for normal people who see it, these algorithms vary in terms of different aspects like time needed to encode a certain message into the image or the time needed to decode/extract a message from image. Also, the noise done to the message has to be minimal. The following points should be considered when designing an algorithm:

- The number of pixels that are used to hide a character.
- The maximum length that message should have if cover image has $n \times m$ pixels.
- Flag settings to indicate that which pixel contains a message.

1.4 Problem Significance and Motivation

In images steganography, proposed steganographic solutions so far have a limitation on the embedding capacity for colored images without effecting the quality of the images, the thesis will propose a solution to increase the image capacity with keeping the image quality relatively accepted

1.5 Limitations

Limitations of this approach are:

1. Specific format of images have to be used.
2. Some places of an image have distortions.
3. The size of the embedded text is limited.
4. Embedding and extracting time have limitations.

1.6 Goals

The goals of the thesis are to provide the following:

1- Propose a technique that provides more embedding capacity than available techniques without using text compression.

2- Keep the overall quality of the image intact.

1.7 Thesis Outline

Chapter two represents the theoretical background and related works of data hiding concepts, data hiding techniques and types including data hiding in image, audio and video.

Chapter three represents the proposed model, the proposed solution, the supported techniques used in the proposed solution, the specification of this supported techniques and the effectiveness of this supported techniques in the proposed solution.

Chapter four represents the experimental results and evaluates the proposed solution in many cases, also compare it with another technology of hide text in text by using open space method.

Finally in Chapter five the results are being discussed, it contains the conclusions and the future works.

CHAPTER TWO

Chapter 2 : Literature Surveys

2.1 Related Work

Least Significant Bit (LSB) method is usually used for hiding data in image. In the LSB method the 8th bit of every byte of the carrier file is substituted by one bit of every bit of the secret information. The LSB technique typically doesn't increase the image size, however looking on the dimensions of the data that's to be hidden within the image, the image will become distorted (Mohammad Y., et al., 2008).

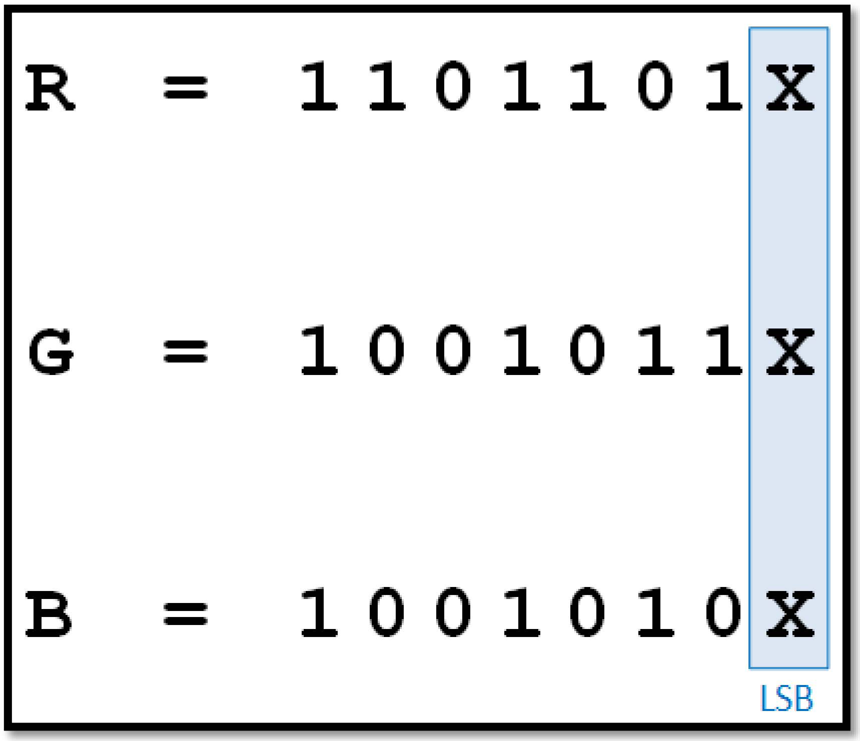


Figure 2.1. LSB Approach

The advantages of LSB are its simplicity to embed the bits of the message directly into the LSB plane of cover-image.

Another approach used for hiding data inside images is the Least Significant Hexadecimal Digit (LSHD), this method embeds in the spatial domain by converting the color value and the ASCII code of the character to hexadecimal values and then substituting each digit of the ASCII in the least digit of the color (imbedding Text in an image).

This approach provide higher payload but do relatively big harm to the image quality.

Ross J. Anderson and Fabien A.P. Petitcolas (2007) argued that every steganographic approach have its limitations; they proposed an approach using Shannon’s theory for achieving perfect secrecy. In the methods that are proposed by H. Motameni and his colleagues one can embed at the dark corners of an image (Mot, 2007). These methods reduces the perceptual noticed by the attacker.

Also, embedding the secret information can be done in frequency domain by using Discrete Wavelet Transform method (Po Yuch Chen, & Hung Ju Lin, 2006). With this technique the embedding ought to be done at high frequency coefficients. P. Mohan Kumar and D. Roopa (2007) urged that one will apply block matching technique to go looking for similarity between blocks of the secret image and embed in LSBs of the cover image.

(Hus, 2009) used completely different strategy in image steganography art by mapping the pixels of image to English letters and special characters. Lisa M. Marvel and Charles G. Boncelet (1999) urged to cover at the inherent noise places.

(Wang, 2006) also did the two way block matching for image-in-image steganography. But this approach is suspicious to the hackers.

(Zhang, 2008) and his colleagues proposed an approach called multibit 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 (Xinpeng Zhang, Shuozhong Wang, & Zhenyu Zhou, 2008).

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). But 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.

Hideki Noda (2006) presents two JPEG steganographic methods using Quantization Index Modulation (QIM) in the Discrete Cosine Transform (DCT) domain. Figure 2.2 show the two methods approximately preserve the histogram of quantized DCT coefficients, aiming at secure steganography against histogram-based attacks.

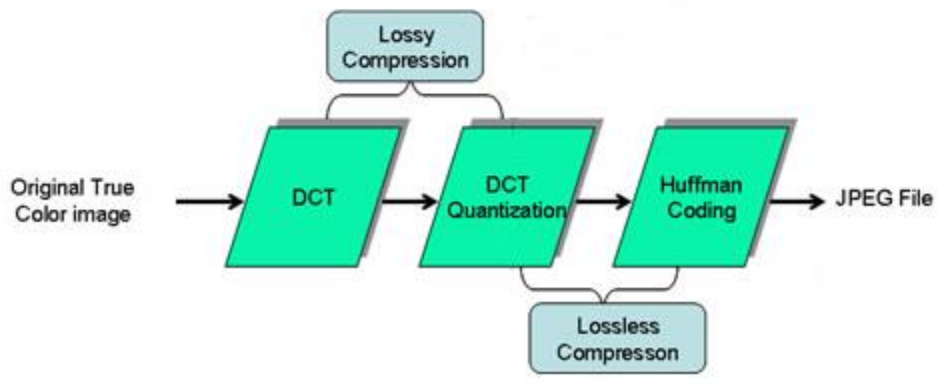


Figure 2.2. Steganography in DCT Domain (Hideki Noda, 2006).

Dutta, Bhattacharyya, & Kim (2009) introduced a robust method of imperceptible audio data hiding. This system provides good and efficient method for data hiding from hackers and sends it to destination safely. This proposed system does not change the size of the file even after encoding, it is also suitable for any type of audio file format. The idea is to hide secret message within audio signal using with a steganographic key, to retrieve the embedded message the same steganographic key should be used along with the extractor. The paper concluded that audio data hiding techniques can be used for a number of purposes other than covert communication or deniable data storage, information tracing and finger printing and tamper detection.

CHAPTER THREE

Chapter 3 : The Proposed Model

Model for the proposed method for data hiding – text in image is presented here, the proposed method will hide text data inside 24 bits colored images using the LSD of the colors, the image that contains the hidden data will be stored in PNG format that uses lossless compression to avoid losing data in the lousy compression process.

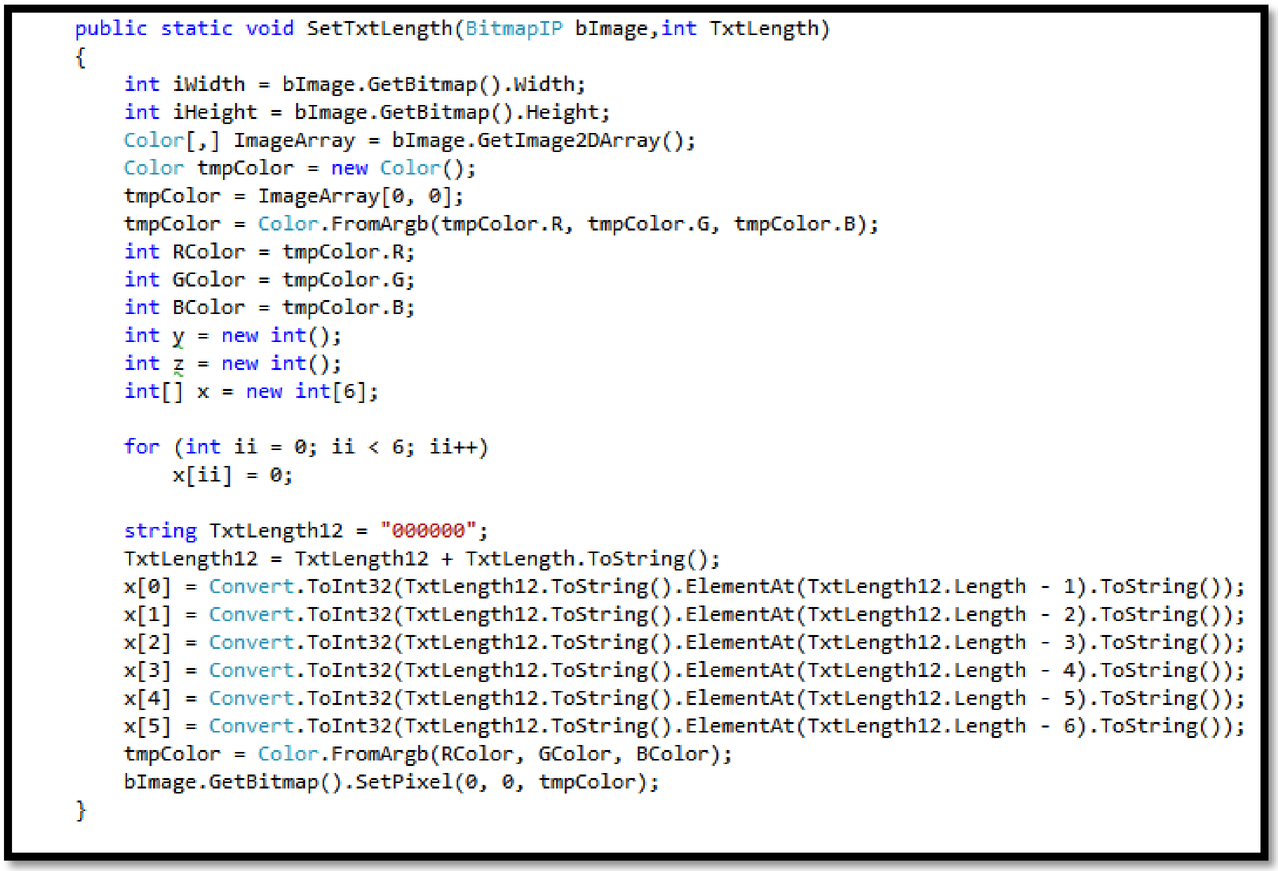
3.1 Data Hiding

Method based on RGB model has been used to hide relatively large messages while keeping the image quality acceptable.

Each color of RGB has three digits to represent its value, take the least significant digit and replace it by a digit of the octal ASCII presentation code of the character to hide, the hiding process is performed as the following:

- Storing the message length

  First pixel of the colored image will have the length of the message, the length of the message is stored by replacing the 2 least significant digits of each color of the first pixel by 2 digits of the length.

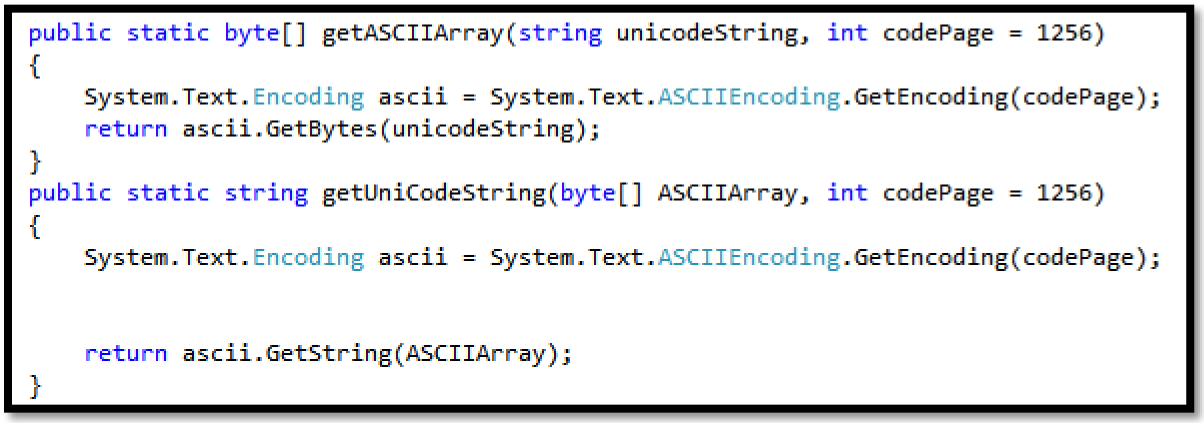


```
public static void SetTxtLength(BitmapIP bImage,int TxtLength)
{
    int iWidth = bImage.GetBitmap().Width;
    int iHeight = bImage.GetBitmap().Height;
    Color[,] ImageArray = bImage.GetImage2DArray();
    Color tmpColor = new Color();
    tmpColor = ImageArray[0, 0];
    tmpColor = Color.FromArgb(tmpColor.R, tmpColor.G, tmpColor.B);
    int RColor = tmpColor.R;
    int GColor = tmpColor.G;
    int BColor = tmpColor.B;
    int y = new int();
    int z = new int();
    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());
    tmpColor = Color.FromArgb(RColor, GColor, BColor);
    bImage.GetBitmap().SetPixel(0, 0, tmpColor);
}
```

Figure 3.1. Code of storing the length of the message.

- Storing each character in the message

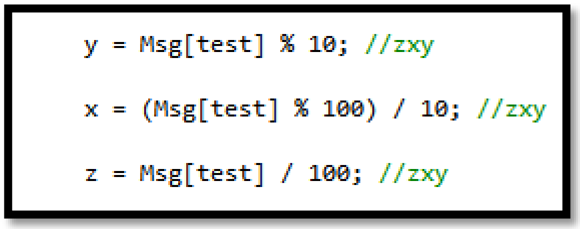
To store the character inside a pixel, get the octal ASCII code of the character in the message because octal only uses 3 bits to represent, the digits of the ASCII codes are later separated and stored in 3 variables, each variable will be stored in the least significant digit of a color.



```
public static byte[] getASCIIArray(string unicodeString, int codePage = 1256)
{
    System.Text.Encoding ascii = System.Text.ASCIIEncoding.GetEncoding(codePage);
    return ascii.GetBytes(unicodeString);
}
public static string getUniCodeString(byte[] ASCIIArray, int codePage = 1256)
{
    System.Text.Encoding ascii = System.Text.ASCIIEncoding.GetEncoding(codePage);


    return ascii.GetString(ASCIIArray);
}
```

Figure 3.2. Code of getting the ASCII code.

- Separating each digit of the ASCII code of the any character, for example '**c**' whose ASCII octal code is 143 using X = (int)(ASCII(**c**)/100), Y = (int)((ASCII(**c**)%100)/10) and Z = ASCII(**c**) % 10 as the following: '**c**' = 143: X = 1, Y = 4, Z = 3.



```
    y = Msg[test] % 10; //zxy

    x = (Msg[test] % 100) / 10; //zxy

    z = Msg[test] / 100; //zxy
```

Figure 3.3. Separating Digits

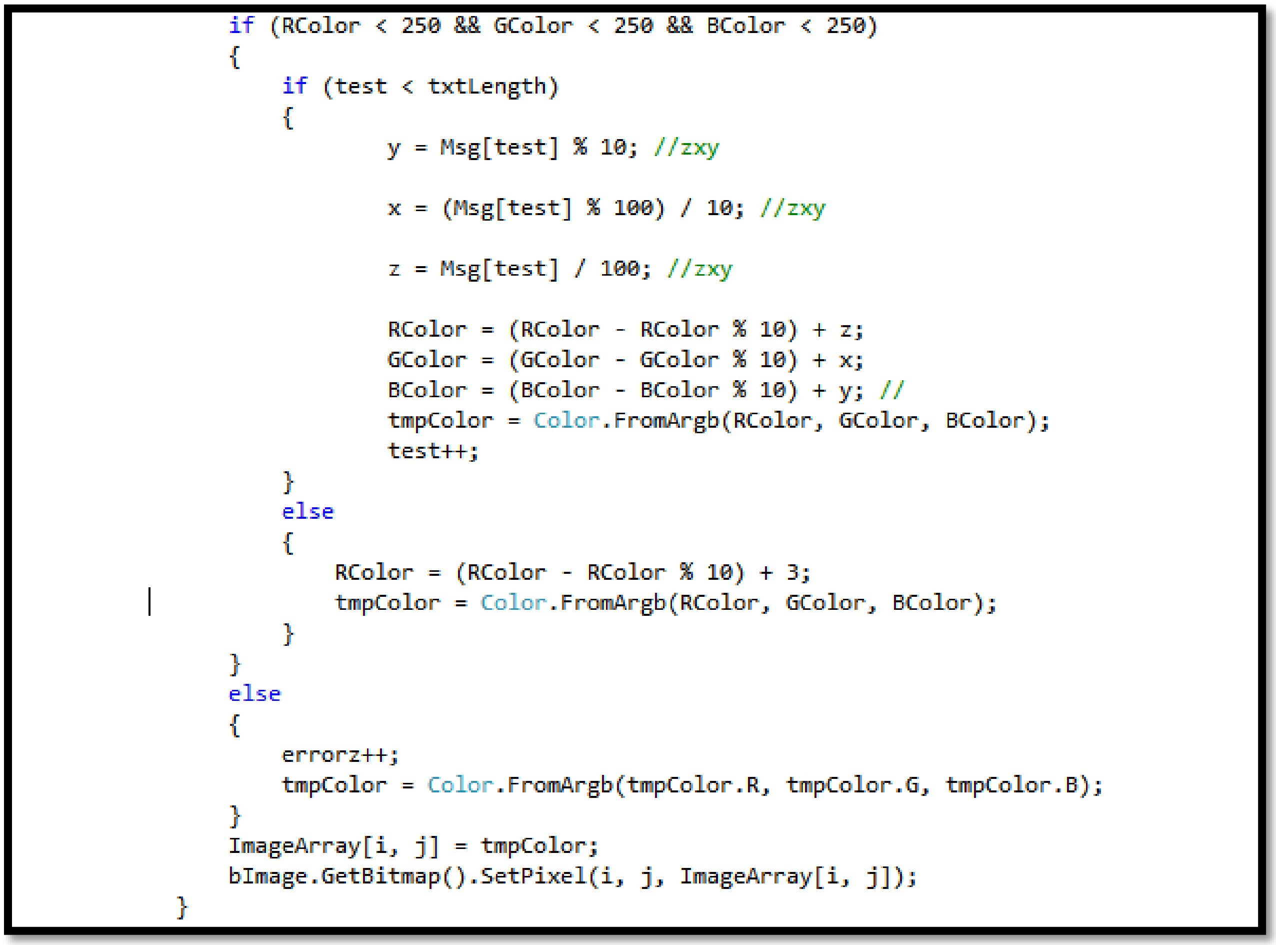
- Get each pixel in the picture and take the values of each color in it (RGB values)

  The colors of each pixel will be separated into 3 variables that hold the values of the colors inside the pixel.

- Check If the value of the color

If any of the colors components used is more than 249, this pixel will not have a character in it, if not leave that pixel.

- Take the pixel and modify its red, green and blue colors to store the the digits of the ASCII code of the character to be hidden (if any of the value of Red, Green or Blue is more than 249 the pixel will not be used to store any data).



```
          if (RColor < 250 && GColor < 250 && BColor < 250)
          {
              if (test < txtLength)
              {
                      y = Msg[test] % 10; //zxy

                      x = (Msg[test] % 100) / 10; //zxy

                      z = Msg[test] / 100; //zxy

                      RColor = (RColor - RColor % 10) + z;
                      GColor = (GColor - GColor % 10) + x;
                      BColor = (BColor - BColor % 10) + y; //
                      tmpColor = Color.FromArgb(RColor, GColor, BColor);
                      test++;
              }
              else
              {
                  RColor = (RColor - RColor % 10) + 3;
                  tmpColor = Color.FromArgb(RColor, GColor, BColor);
              }
          }
          else
          {
              errorz++;
              tmpColor = Color.FromArgb(tmpColor.R, tmpColor.G, tmpColor.B);
          }
          ImageArray[i, j] = tmpColor;
          bImage.GetBitmap().SetPixel(i, j, ImageArray[i, j]);
      }
```

Figure 3.4. Embedding Algorithm

Example 1:

To embed the word “Hello” inside a colored image whose the (R, G, B) color values for the first 5 pixels are { (192,078,171), (213, 155, 175), (099, 170, 010), (123, 213, 222), (232, 097, 043) }

After storing the length of the message, the following steps are performed to hide the word “Hello” we take the octal ASCII of each character:

H	e	l	l	o
110	145	154	154	157

Table 3.1 Octal ASCII of characters.

Hiding ‘H’:

To hide ‘H’ in the first pixel whose value is (192,078,171), do the following:

‘H’ = 110, Separate each digit of the ASCII code using integer division as follows:

x = (ASCII(**H**)/100),

y = ((ASCII(**H**)%100)/10)

z = ASCII(**H**) % 10

So that x = 1, y = 1 and z = 0.

Then replace the least significant digit of R,G,B values of the pixel (192,078,171) with x, y and z so that the new value of the pixel is (191, 071, 170).

Hiding ‘e’:

To hide ‘e’ in the first pixel whose value is (213, 155, 175), do the following:

‘e’ = 145, Separate each digit of the ASCII code using integer division as follows:

x = (ASCII(**e**)/100),

y = ((ASCII(**e**)%100)/10)

z = ASCII(**e**) % 10

So that x = 1, y = 4 and z = 5.

Then replace the least significant digit of R,G,B values of the pixel (213, 155, 175) with x, y and z so that the new value of the pixel is (211, 154, 175).

Hiding ‘l’:

To hide ‘l’ in the first pixel whose value is (099, 170, 010), do the following:

‘l’ = 154, Separate each digit of the ASCII code using integer division as follows:

x = (ASCII(**l**)/100),

y = ((ASCII(**l**)%100)/10)

z = ASCII(**l**) % 10

So that x = 1, y = 5 and z = 4.

Then replace the least significant digit of R,G,B values of the pixel (099, 170, 010) with x, y and z so that the new value of the pixel is (091, 175, 014).

Hiding ‘l’:

To hide ‘l’ in the first pixel whose value is (123, 213, 222), do the following:

‘l’ = 154, Separate each digit of the ASCII code using integer division as follows:

x = (ASCII(**l**)/100),

y = ((ASCII(**l**)%100)/10)

z = ASCII(**l**) % 10

So that x = 1, y = 5 and z = 4.

Then replace the least significant digit of R,G,B values of the pixel (123, 213, 222) with x, y and z so that the new value of the pixel is (121, 215, 224).

Hiding ‘o’:

To hide ‘o’ in the first pixel whose value is (232, 097, 043), we the following:

‘o’ = 157, Separate each digit of the ASCII code using integer division as follows:

x = (ASCII(**o**)/100),

y = ((ASCII(**o**)%100)/10)

z = ASCII(**o**) % 10

So that x = 1, y = 5 and z = 7.

Then replace the least significant digit of R,G,B values of the pixel (232, 097, 043) with x, y and z so that the new value of the pixel is (231, 095, 047).

3.2 Data Retrieving

When want to extract the message from the steganographic image first thing to do is extracting the message length from the first pixel.

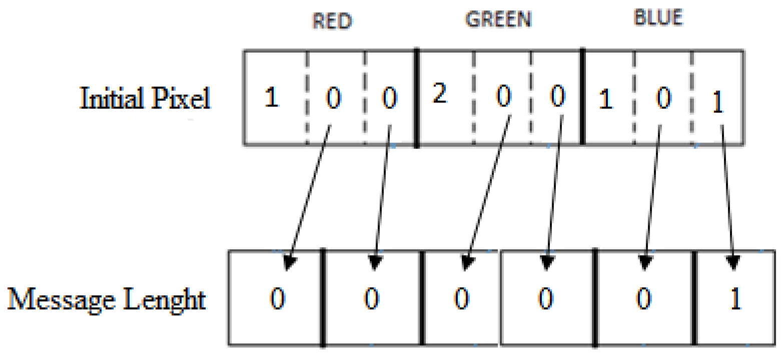


Figure 3.5. Retrieval of Message Length.

After that, the following steps need to be done to get the message from the steganographic image:

1) Checking on the first pixel that is visited (after the initial pixel), if its Red value are less than 250 then the process gets the value of the ‘X’ variable from it, otherwise skip that pixel.

2) Checking on the first pixel that is visited (after the initial pixel), if its Green value are less than 250 then the process gets the value of the 'Y' variable from it, otherwise skip that pixel.

3) Checking on the first pixel that is visited (after the initial pixel), if its Blue value are less than 250 then the process gets the value of the 'Z' variable from it, otherwise skip that pixel.

4) Putting the XYZ values together to have the complete ASCII code of the character which equals to ":".

5) Repeat the previous steps until the end of the message.

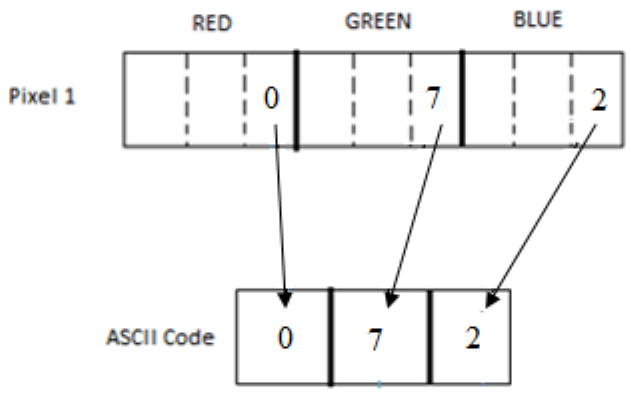


Figure 3.6. Retrieval of Stored Character

3.3 Discussion

From the above operations one can record the following points about the new proposed image steganography method:

- The length of the message to be stored inside an image depends on the size of the image and the pixel colors so 2 different images of the same size can have different capacity.

- The proposed method can provide extra security by modifying the ASCII code using a certain key before embedding it inside the image.
- This method allows the storage of longer textual data to be stored inside an image in comparison to other methods.
- Encoding and decoding operations will operate faster than well-known methods like LSB.

CHAPTER FOUR

Chapter 4 : Experimental Results

This chapter reviews the characteristics of the proposed method and compares them with the characteristics of different known algorithms in the field, in this regard, the programming language is C#, the tool was developed using Visual Studio 2010, however, variety of tools been used to evaluate the performance of the proposed method on PNG images, the testing was done on 64 Bit Core I3 Intel processor operating in frequency of 2.53 GHz and 4.00 GB of RAM.

4.1 Image Quality Test

This test measures the quality of the stego-image 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 both LSB and DCT based methods for each of the images used for testing after storing full capacity in each image, data is exposed in terms of root mean squared error and peak signal to noise ratio using the following formulas as stated by (Mohammad Ali Bani Younes, & Aman Jantan, 2008):

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{n}(x1,t-x2,t)^2}{n}} \text{.. (1).}$$

Where:

N: is the number of pixels in the image,

$X_{1,t}$: is the value of the pixel in the original image before embedding.

$X_{2,t}$: is the value of the pixel in the image after embedding.

$$\mathrm{PSNR} = 20 \cdot \log_{10}\left(\frac{255}{RMSE}\right) \qquad (2).$$

Higher values of PSNR indicate better quality where higher values of RMSE indicate lower quality.

The study is performed on a dataset of 4 images, each image is of size 512 × 512.

Table 4-1 shows statistics after storing ~260883 characters which is the maximum capacity when using DCT, the embedding is performed using the proposed method.

	RMSE	**PSNR**
Lena	2.06375	41.8376
Baboon	2.04356	41.9230
Peppers	2.02228	42.0139

Table 4.1. Statistical Data after storing ~260883 characters Using Proposed Method.

In spite of the relatively large text embedded, the PSNR values remain within accepted range for all images, a value of 41 is considered to be relatively good.

Table 4-2 shows statistics after storing 98300 characters which is the maximum capacity when using LSB, the embedding is performed using the proposed method.

	RMSE	**PSNR**
Lena	1.18440	46.6608
Baboon	1.18722	46.6401
Peppers	1.18687	46.6427

Table 4.2 . Statistical Data After Embedding 98300 Characters Using Proposed Method

After storing 98300 character inside each image, the proposed method yielded a relatively good results with PSNR being higher than 46.

Table 4-3 shows statistics after storing 34816 characters which is the maximum capacity when using DCT-Based, the embedding is performed using the proposed method.

	RMSE	**PSNR**
Lena	0.70933	51.1138
Baboon	0.74583	50.6780

Peppers	0.71856	51.0015

Table 4.3. Statistical Data After Embedding 34816 Characters Using Proposed Method.

The results of the proposed method after storing different amount of character inside each image of the testing dataset yielded a relatively good results with the lowest PSNT being above 41, such readings mean that the quality is still within the acceptable range.

Table 4-4 shows statistics after storing 98300 characters using LSB.

	RMSE	**PSNR**
Lena	0.70500	51.1670
Baboon	0.71228	51.0777
Peppers	0.69774	51.2569

Table 4.4. Statistical Data After Embedding 98300 Characters Using LSB

The quality readings of the LSB technique after hiding 98300 characters present a relatively good quality, better quality than the proposed method.

Table 4-5 shows statistics after storing 34816 characters using DCT-Based.

	RMSE	**PSNR**

Lena	1.29884	45.8596
Baboon	1.32570	45.6818
Peppers	1.48897	44.6730

Table 4.5. Statistical Data After Embedding 34816 Characters Using DCT-Based.

Table 4-6 shows statistics after storing 327680 characters which is the maximum capacity when using Least significant hexadecimal digit (LSHD).

	RMSE	**PSNR**
Lena	2.97122	38.672
Baboon	2.82120	39.122
Peppers	2.89156	38.908

Table 4.6 . Statistical Data After Embedding 327680 Characters Using LSHD.

PSNR values after embedding using LSHD were significantly low, indicating lower quality, these low values compromise the security of the secret data when exposed to statistical analysis.

With keeping the capacity in perspective, The results of the proposed method is more satisfying than the known LSB and DCT methods as it reduces the number of pixels changed in comparison to the data stored with keeping the standard deviation in low levels, this reduction allows us to embed relatively larger text sizes in comparison with other known methods while keeping the overall view of the image intact.

4.2 Payload

Payload is the amount of text that can be hidden inside an image, tables (4-7) to (4-9) show the actual data load that can be embedded inside the 4 cover images using the proposed method and an estimated amount of data that can be embedded inside the 4 cover images using the LSB method.

All cover images in the first set are 512 x 512 colored images and a character is represented by 8-bits.

	LSB Hiding Capacity (Bytes)	**Proposed Method Hiding Capacity (Bytes)**
Baboon	98300	260883
Lena	98300	260928
Peppers	98300	262142

Table 4.7. Capacity of Different Images.

This is the maximum storage capacity with text compression disabled when using DCT-based method.

	DCT-Based and Quantization (Bytes)	Proposed Method Hiding Capacity (Bytes)
Baboon	34816	260883
Lena	34816	260928
Peppers	34816	262142

Table 4.8. Capacity of DCT-Based and LSD for Different Images.

This is the maximum storage capacity with text compression disabled when using LSHD method.

	LSHD (Bytes)	Proposed Method Hiding Capacity (Bytes)
Baboon	327680	260883
Lena	327680	260928
Peppers	327680	262142

Table 4.9. Capacity of LSHD and LSD for Different Images.

The following figures represent the hiding capacity of the proposed method (LSD) in comparison to DCT for the images used in this study.

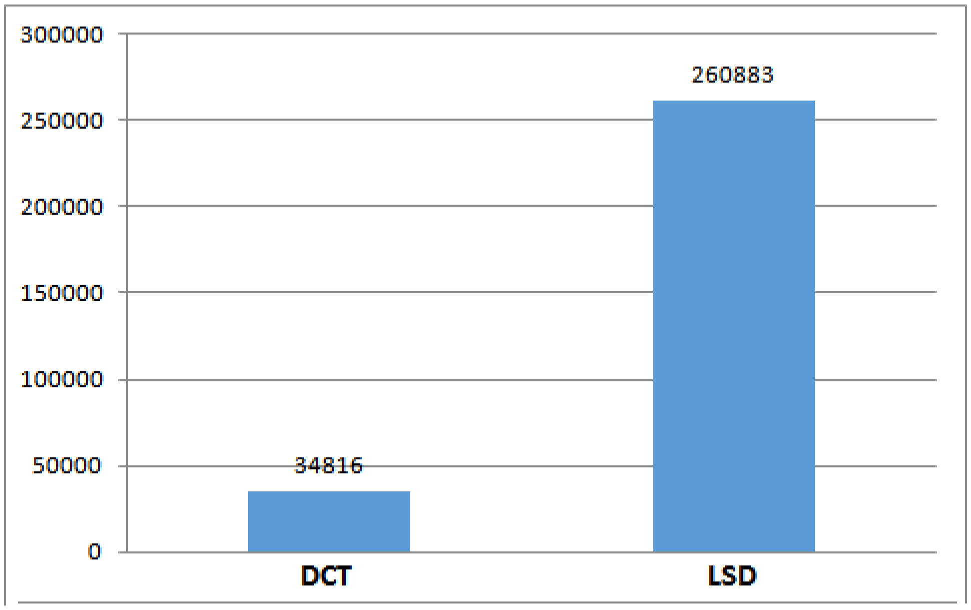


Figure 4.1. Payload of DCT/LSD in bytes for Baboon.

The proposed method (LSD) allowed a payload of 260883, DCT allowed 34816 characters, the proposed method provided a higher payload.

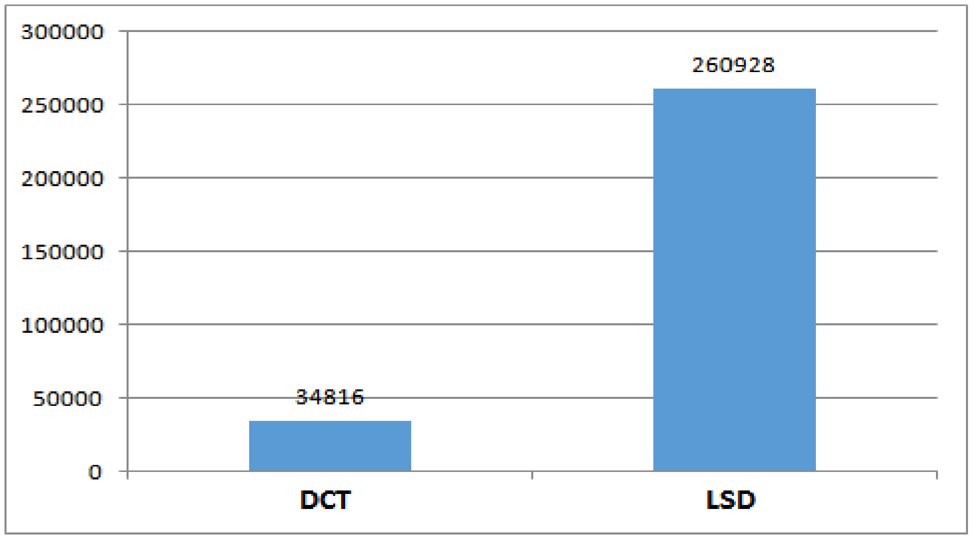


Figure 4.2.Capacity of DCT/LSD in bytes for Lena.

The proposed method (LSD) allowed a payload of 360928, DCT allowed 34816 characters, the proposed method provided a higher payload.

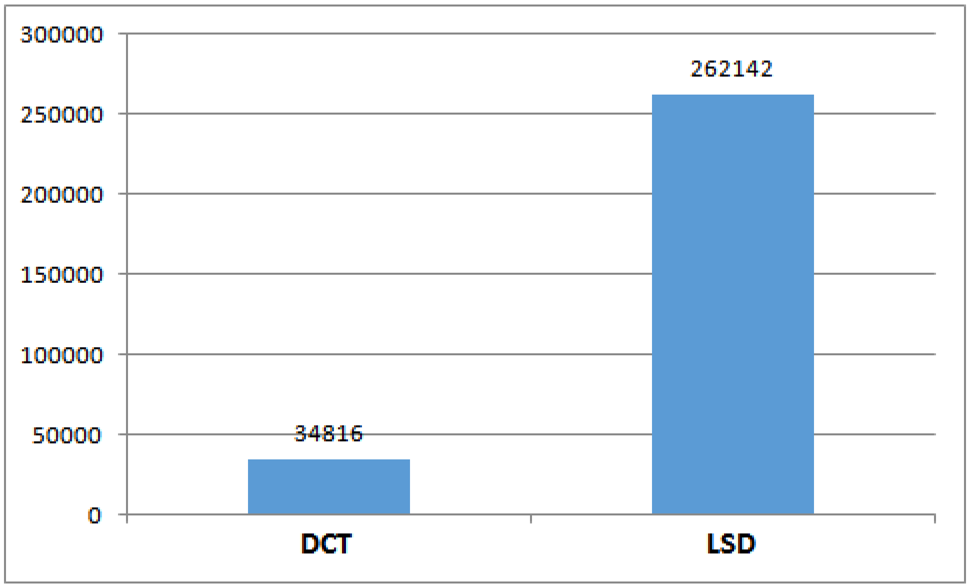


Figure 4.3.Capacity for DCT/LSD in bytes for Peppers.

The proposed method (LSD) allowed a payload of 262142, DCT allowed 34816 characters, the proposed method provided a higher payload.

The following figures represent the hiding capacity of the proposed method (LSD) in comparison to LSB for the images used in this study.

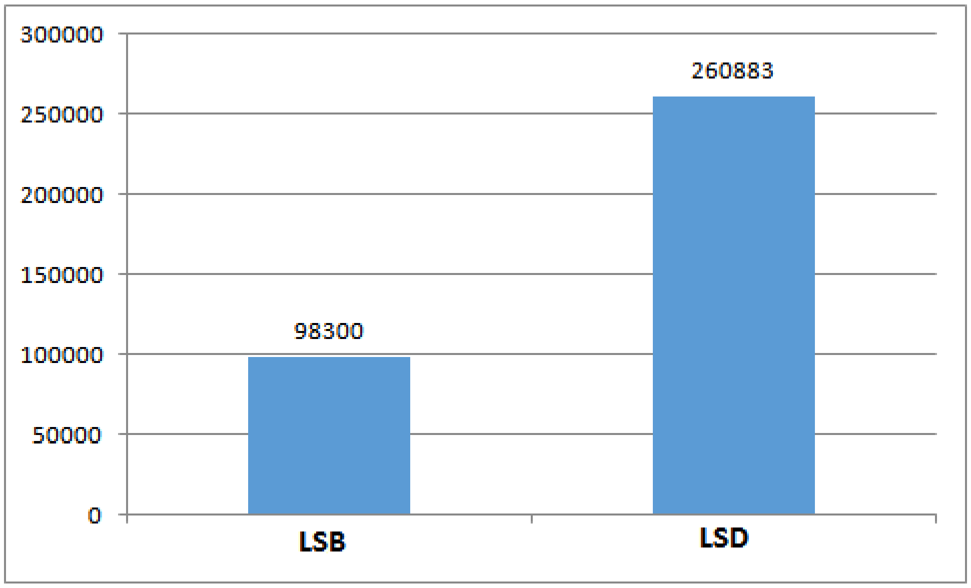


Figure 4.4.Capacit of LSB/LSD in bytes for Baboon.

The proposed method (LSD) allowed a payload of 260883, LSB allowed 98300 characters, the proposed method provided a higher payload.

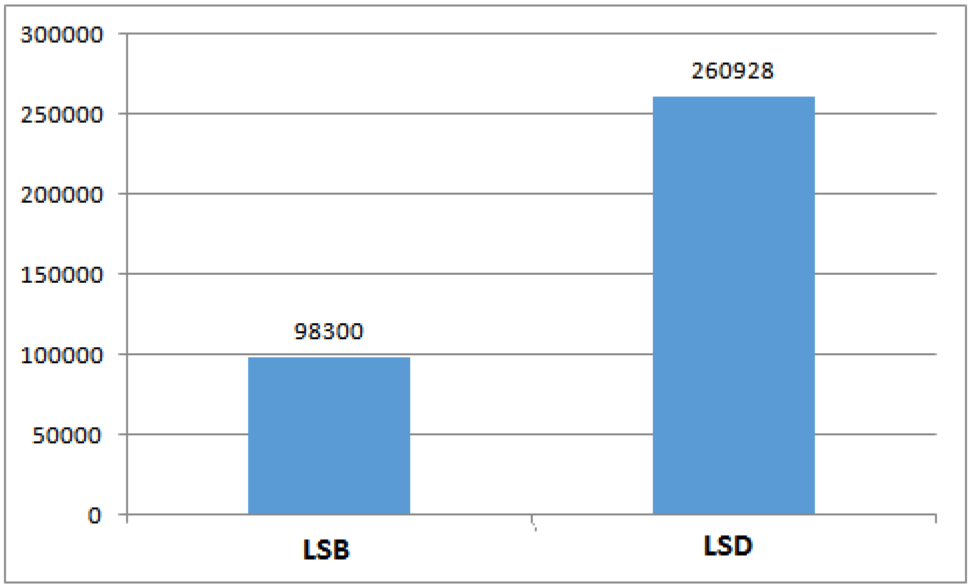


Figure 4.5.Capacity of LSB/LSD in bytes for Lena.

The proposed method (LSD) allowed a payload of 260928, LSB allowed 98300 characters, the proposed method provided a higher payload.

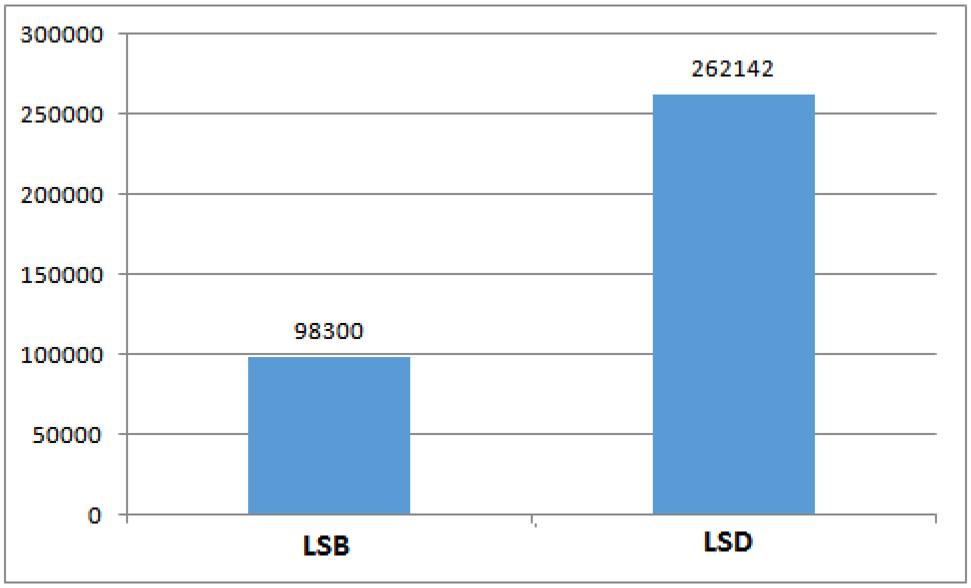


Figure 4.6. Capacity of LSB/LSD in bytes for Peppers

The proposed method (LSD) allowed a payload of 262142, LSB allowed 98300 characters, the proposed method provided a higher payload.

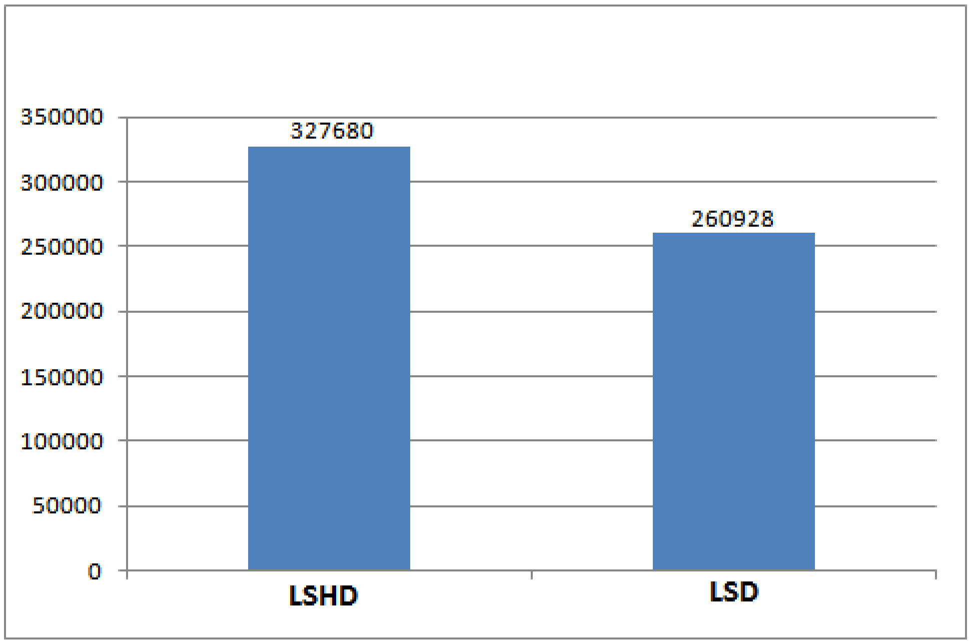


Figure 4.7. Capacity for LSHD/LSD for Lena

The proposed method (LSD) allowed a payload of 260928, LSHD allowed 327680 characters, the proposed method provided a higher payload.

The LSB and DCT-based methods provide static capacity in relation to the image dimensions, the proposed method however provides different capacities according to the color nature of the picture since components that hold value greater than 249 are not used for storage and thus pictures that contain bright colors are not preferable.

The following figures 4.8-4.16 show the cover images used in the proposed method before/after the embedding of data using full capacity of ~260883 characters.

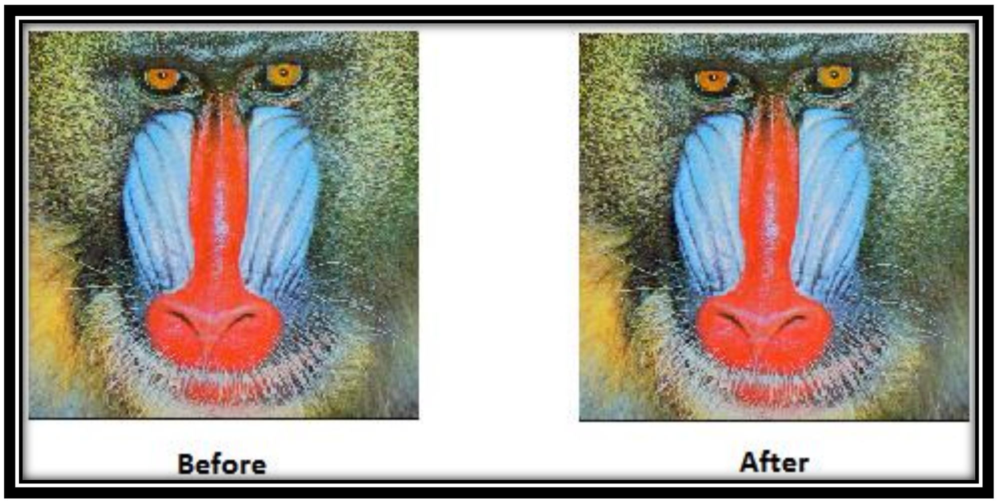


Figure 4.8. Baboon Before and After Embedding Using the Proposed Method

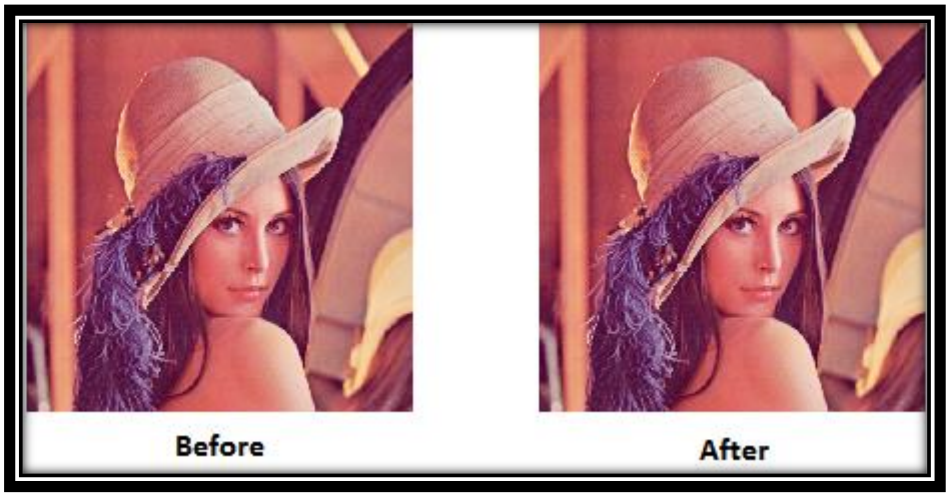


Figure 4.9. Lena Before and After Embedding Using the Proposed Method.

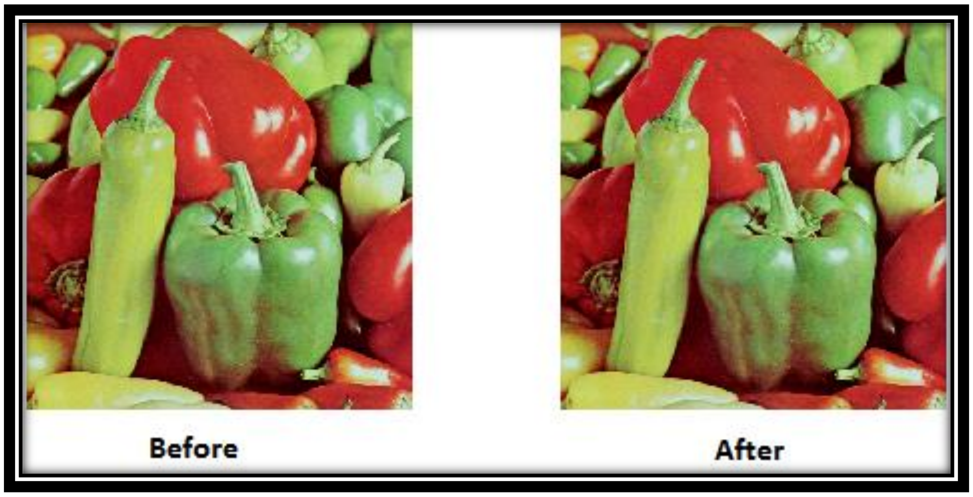


Figure 4.10. Peppers Before and After Embedding Using the Proposed Method.

The following figures show the cover images used in LSB method before/after the embedding of data using full capacity of 98300 characters.

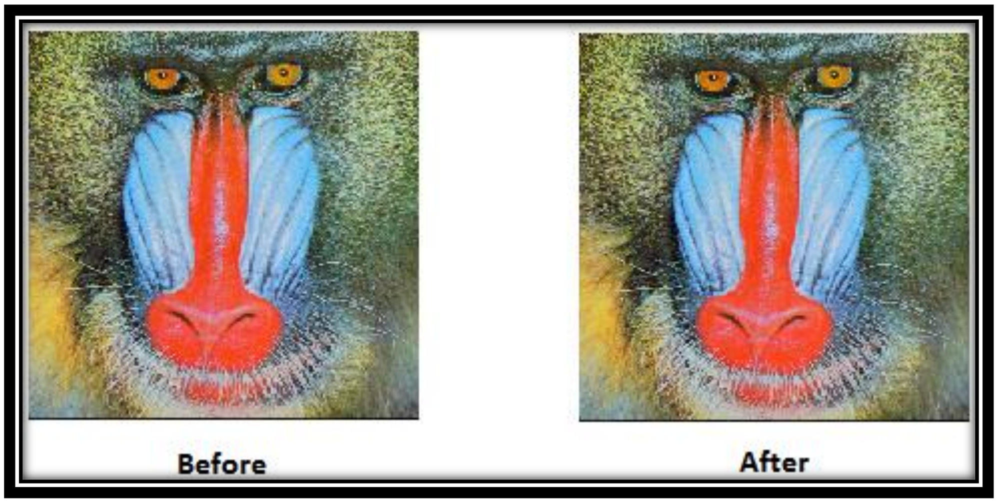


Figure 4.11. Baboon Before and After Embedding Using the LSD Method

Figure 4.12. Lena Before and After Embedding Using the LSB Method.

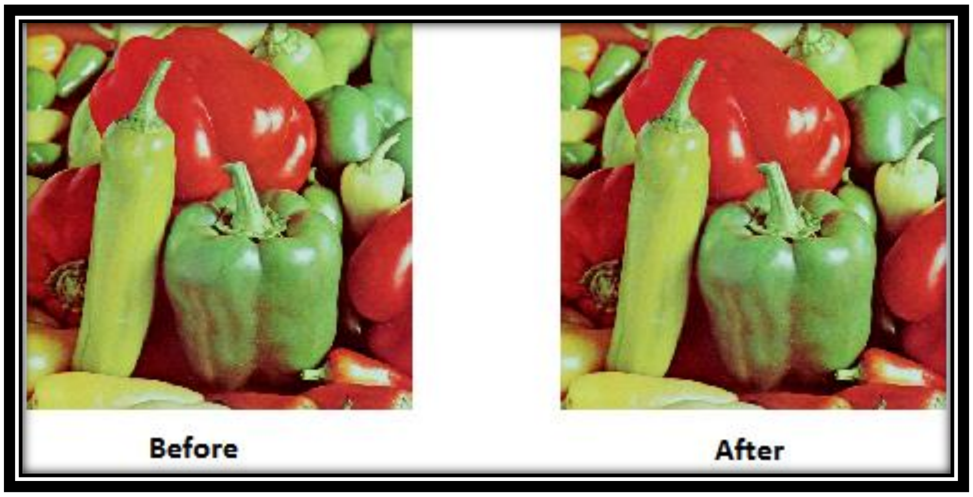


Figure 4.13. Peppers Before and After Embedding Using the LSB Method.

The following figures show the cover images used in DCT-Based method before/after the embedding of data using full capacity of 34816 characters.

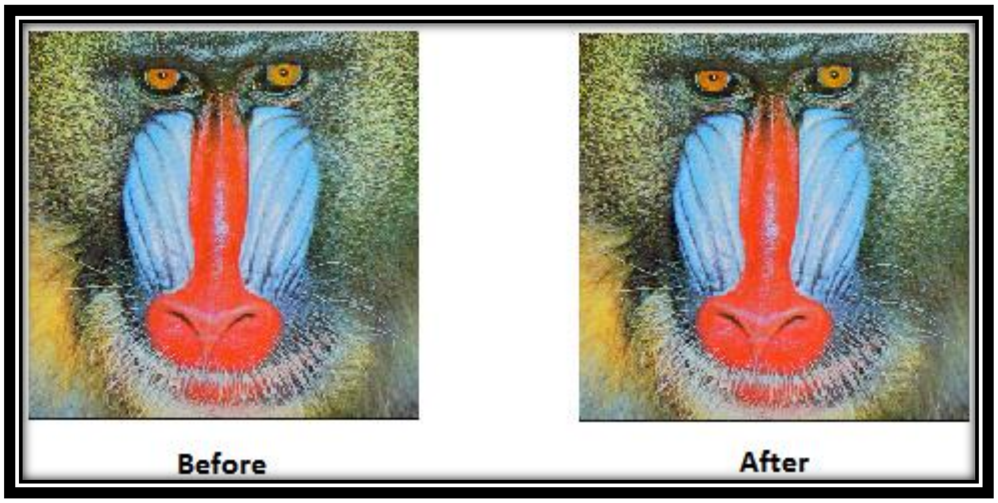


Figure 4.14. Baboon Before and After Embedding Using the DCT-Based Method.

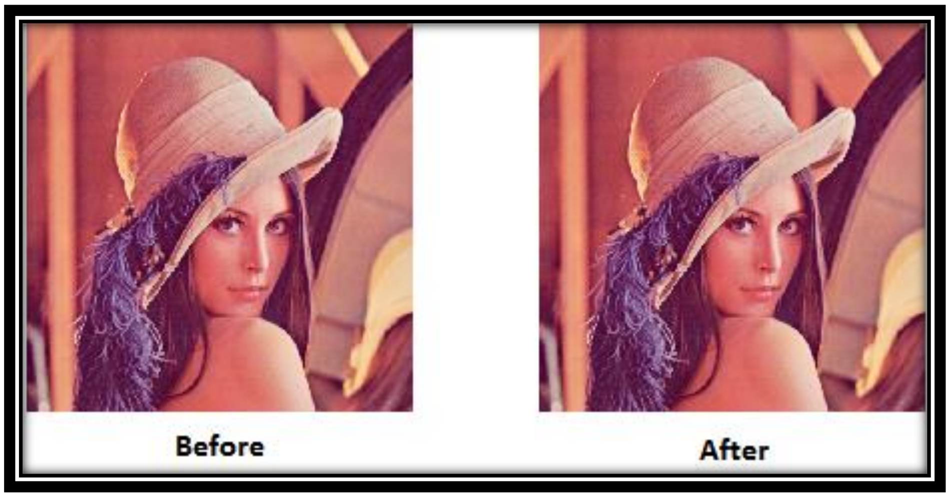


Figure 4.15. Lena Before and After Embedding Using the DCT-Based Method.

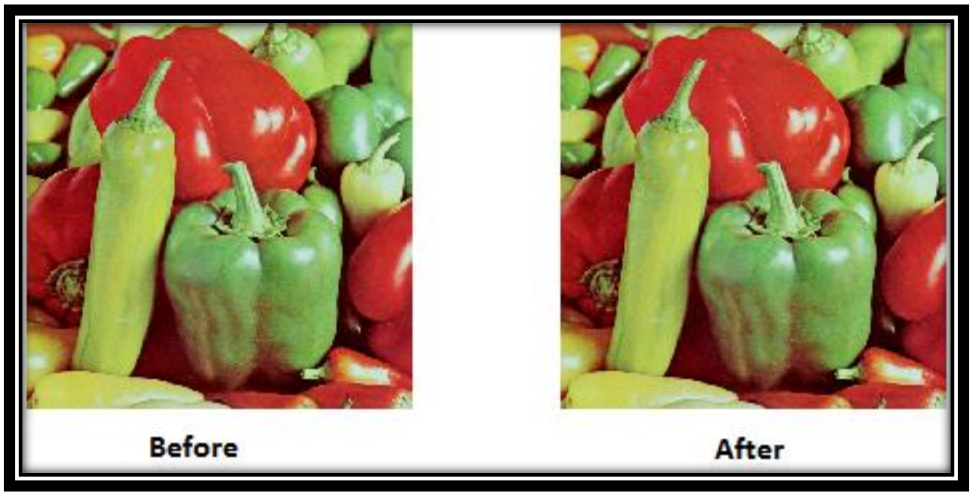


Figure 4.16. Peppers Before and After Embedding Using the DCT-Based Method.

It is worth to mention that is there were no apparent differences between the original cover image and the stego image in all methods used.

4.3 Security Test

This test compares the original image with the image after embedding through a statistical tool called Histogram, the degradation of the quality can be noticed by applying the histogram analysis.

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.

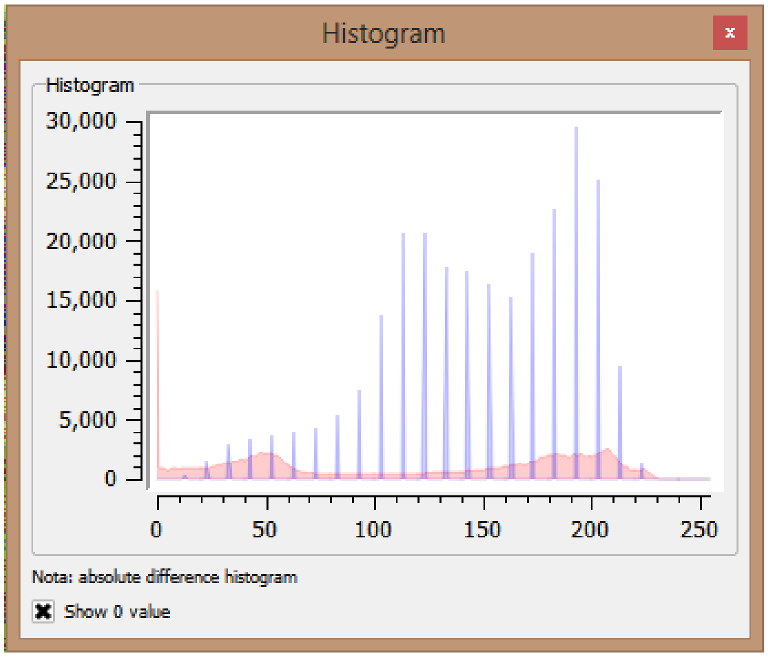


Figure 4.17. Histogram for Peppers After Embedding Using LSD

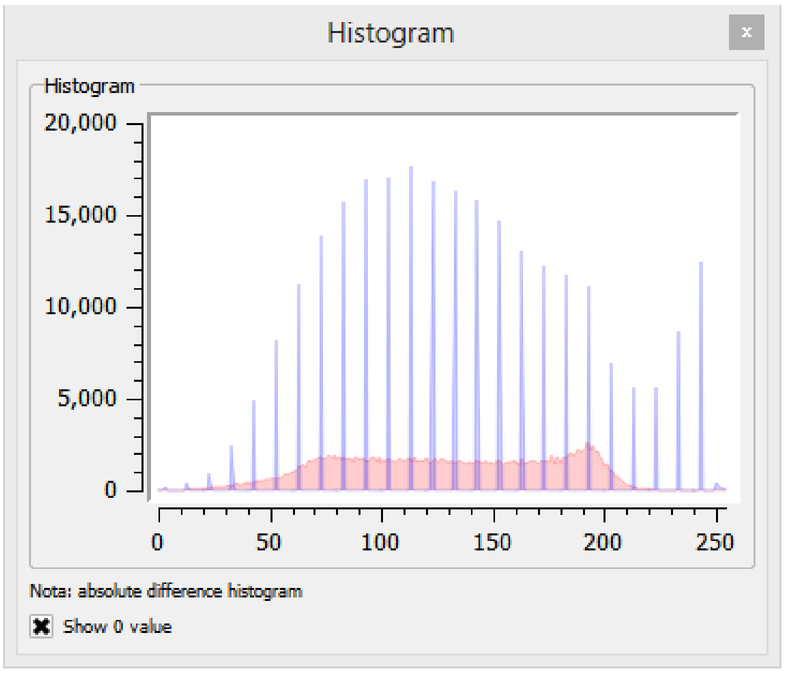


Figure 4.18. Histogram for Baboon After Embedding Using LSD.

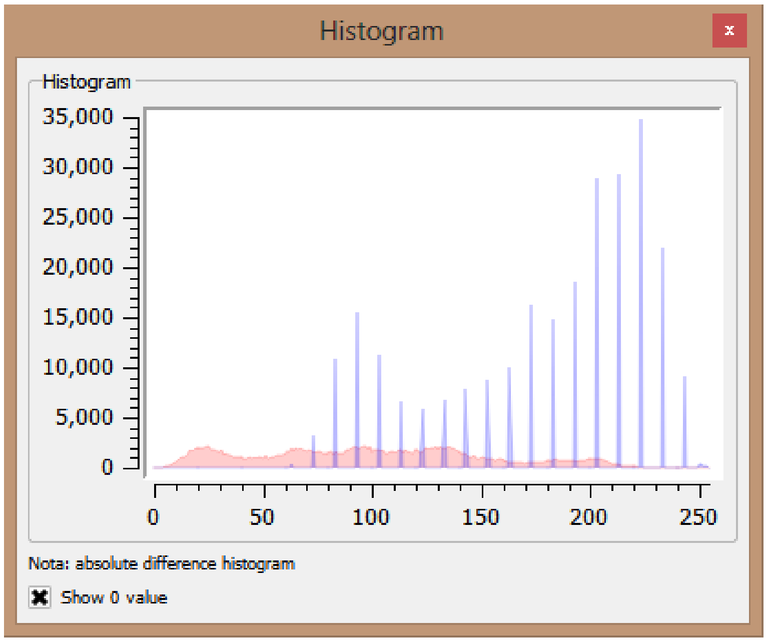


Figure 4.19.Histogram for Lena After Embedding Using LSD.

After considering the results, one can see that the proposed method give more hiding capacity than LSB and DCT based methods, it also provides more security when using small messages to hide, the analysis of difference would look pretty much like an image went through image processing for the purpose of enhancing the look of the image rather than a data been embedded inside, however, bright images are less useful and provide less storage capacity.

4.4 Robustness

Algorithms that uses image as cover objects to hide data inside them are irresistible to image processing operations like rotation, cut…etc. As such operations will damage the stored data. An algorithm usually uses certain point to store data, if these points undergo through image processing operations that changes their value all or some of the data stored in that point is lost which in turn results in partial loss of the message hidden inside the image, however, the robustness of the proposed method lies in the relatively big capacity that can be up to 786431 characters per 512×512 images if color values do not exceed 250. Also, minor modifications produced by image processing on the stegoimage result in partial loss of the secret text.

CHAPTER FIVE

Chapter 5 : Conclusions

5.1 Summary

The analysis of comparison the suggested method with LSB and DCT techniques show that this method provides more storage capacity.

Increase in security can be provided by using AES encryption to encrypt the text before embedding it into cover image.

5.2 Conclusion

After studying the results it is now obvious that the proposed method provides better storage capacity than LSB and DCT, however, LSHD provided better storage capacity than the proposed method, the quality measures for of the images after embedding using the proposed method compared to LSB, DCT and LSHD show that the effect the proposed method left on the image after storing data is within the accepted range, however, LSB and DCT effects on the image was lower, the effect of LSHD was the highest among the tested methods which exposes the secret data if the image is analyzed in intent to detect secret hidden data.

5.3 Future Work

This study forms the base for several future researches, the following points are suggested to improve the performance of the algorithm:

1. The proposed algorithm was done on PNG images without taking advantage of the alpha channel, a good idea would be to exploit the alpha channel for further increasing the storage capacity.

2. Enhancing the algorithm to make the message remain intact even after some image processing operations like rotations.
3. Applying the same method to hide different types of data other than text.
4. Applying encryption on data before embedding to provide extra security.
5. Using key to embed data inside image in a random manner.

References

Anderson, R.J. & Petitcolas, F. A. P. (1998), On the limits of steganography, IEEE Journal of selected Areas in Communications, 16(4), 474-481.

Kefa Rabah. (2004), Steganography-The Art of Hiding Data, Department of Physics, Eastern Mediterranean University, Gazimagusa, North Cyprus, via Mersin 10, Turkey.

Motameni H., Norouzi M., M.Jahandar, & Hatami A. (2007). Labeling method in steganography. *Proceedings of world academy of science, engineering and technology*, 24, 349-354.

Po Yuch Chen, & Hung Ju Lin. (2006). A DWT Based Approach for Image Steganography. *International journal of Applied Science and Engineering*, 4(3), 275-290.

Kumar P. Mohan, & Roopa D. (2007). An Image Steganography Framework with Improved Tamper Proofing. Asian Journal of Information Technology, 6(10), 1023-1029.

Al Husainy A.F. (2009). Image Steganography by mapping Pixels to letters. *Journal of Computer Science*, 5(1), 33-38.

Marvel Lisa M., & Boncelet Charles G. (1999). Spread Spectrum Image Steganography. *IEEE Transactions on Image Processing*, 8(8), 1075-1083.http://dx.doi.org/10.1109/83.777088

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.

Ran-Zan Wang, & Yeh-Shun Chen. (2006). High Payload Image Steganography Using Two-Way Block Matching. *IEEE Signal Processing Letters*, 13(3), 161 - 164. http://dx.doi.org/10.1109/LSP.2005.862603

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

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.

Ashfaaq M., Bosco J. & Rayappan B. (2010), Color Guided color Image Steganography, R. Amirtharajan, Sandeep Kumar Behera, Motamarri Abhilash Swarup.

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

Hideki Noda, MichiharuNimi, & Eiji Kawaguchi. (2006). High-performance JPEG steganography using Quantization index modulation in DCT domain. *Pattern Recognition Letters*, 27, 455-461. http://dx.doi.org/10.1016/j.patrec.2005.09.008

Artz, D. (2001), Digital Steganography: Hiding Data within Data, *IEEE Internet Computing Journal*, 5(3), 75-80.

Bret Dunbar (2002), A detailed look at Steganographic Techniques and their use in an Open-Systems Environment, SANS Institute InfoSec Reading Room.

Natarajan Meghanathan and Lopamudra Nayak (2010), Steganalysis Algorithms for Detecting the Hidden Information in Image, Audio and Video Cover Media, Jackson State University, 1400 Lynch St, Jackson, MS, USA.

C. Kraetzer and J. Dittmann, (2008), Pros and Cons of Mel-cepstrum based Audio Steganalysis using SVM Classification, Lecture Notes in Computer Science, vol. 4567, pp. 359 – 377.

Mohammad Fahmi Alalem, Abdullah Muhanah Manasra, A Steganographic Data Security Algorithm with Reduced Steganalysis Threat, Birzeit University, Birzeit – Palestine.

Mohammad Ali Bani Younes, & Aman Jantan. (2008). A New Steganography Approach for Image Encryption Exchange by using the LSB insertion. *IJCSNS International Journal of Computer Science and Network Security*, 8(6), 247-254.

Anderson Ross J., & Petitcolas Fabian A.P.. (1998). On the Limits of steganography. *IEEE Journal of selected Areas in communication. Special Issue on Copyright and Privacy protection.* 16(4), 474-481.

Arvind Kumar, (2010). Steganography- A Data Hiding Technique. *International Journal of Computer Applications. 9(7), 975-8887.*

Bailey, K. and Curran, K. “An evaluation of image-based steganography methods”. International Journal of Digital Evidence, Fall 2003.

Abdullah, Y. F., & Nasereddin, H. H. (2013). Proposed Data Hiding Technique – Text under Text. American Academic & Scholarly Research Journal (AASRJ), 243-248.

imbedding Text in an image. (n.d.). Retrieved 3 29, 2014, from wundes: http://wundes.com/imbed/

Appendix A: Hiding and Extracting Algorithm

```
public static string LSD_Colored_Extract(BitmapIP bImage)

        {

            int txtLength = GetTxtLength(bImage);

            int test = 0;

            int width = bImage.GetBitmap().Width;

            int height = bImage.GetBitmap().Height;

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

            Color tmpColor = new Color();

            byte[] alpha = new byte[txtLength];

            int[] result = new int[txtLength];

            char[] chres = new char[txtLength];

            string[] txt = new string[txtLength];

            int x = new int();

            int y = new int();

            int z = new int();

            //int capacityz = 262144;

            for (int i = 0; i < width; i++)

            {

                for (int j = 0; j < height; j++)

                {


                    tmpColor = ImageArray[i, j];

                    tmpColor = Color.FromArgb(tmpColor.R, tmpColor.G, tmpColor.B);

                    int RColor = tmpColor.R;

                    int GColor = tmpColor.G;

                    int BColor = tmpColor.B;
                    x = 0;
                    y = 0;
                    z = 0;

                    if (test < txtLength && RColor < 250 && GColor < 250 && BColor < 250
&& ImageArray[i, j] != ImageArray[0, 0])
                    {
                        if (RColor > 99 && (RColor / 100) < 3)
                        {
                            z = RColor % 10;
                            x = GColor % 10;
                            y = BColor % 10;
                            result[test] = (z * 100) + (x * 10) + (y);
                            chres[test] = Convert.ToChar(result[test]);
                            test++;
                        }
                        else
                        {
                            if (RColor < 100)
                            {
                                z = RColor % 10;
                                x = GColor % 10;
                                y = BColor % 10;
                                result[test] = (z * 100) + (x * 10) + (y);
                                chres[test] = Convert.ToChar(result[test]);
                                test++;
                            }
                        }
                    }
```

```
            }
        }
        string last = new string(chres);
        return last;
    }
    public static Bitmap LSD_Colored_Hide(BitmapIP bImage, char[] OriginalText)
    {
        Color[,] ImageArray = bImage.GetImage2DArray();
        Color tmpColor = new Color();
        int txtLength = OriginalText.Length;
        char[] EncMsg = new char[txtLength];
        EncMsg = OriginalText;
        int iHeight = bImage.GetBitmap().Height;
        int iWidth = bImage.GetBitmap().Width;
        int[] Msg = getTextVal(EncMsg);
        int test = 0;
        SetTxtLength(bImage, txtLength);
        int errorz = 0;
        int x = new int();
        int y = new int();
        int z = new int();
        for (int i = 0; i < iWidth; i++)
        {
            for (int j = 0; j < iHeight; j++)
            {
                if (ImageArray[i, j] != ImageArray[0, 0])
                {
                    tmpColor = ImageArray[i, j];
                    tmpColor = Color.FromArgb(tmpColor.R, tmpColor.G, tmpColor.B);
```

```
int RColor = tmpColor.R;

int GColor = tmpColor.G;

int BColor = tmpColor.B;

x = 0;

y = 0;

z = 0;


if (RColor < 250 && GColor < 250 && BColor < 250)

{

    if (test < txtLength)

    {

        if (RColor > 99)

        {


            y = Msg[test] % 10; //zxy


            x = (Msg[test] % 100) / 10; //zxy


            z = Msg[test] / 100; //zxy

            RColor = (RColor - RColor % 10) + z;

            GColor = (GColor - GColor % 10) + x;

            BColor = (BColor - BColor % 10) + y; //

            tmpColor = Color.FromArgb(RColor, GColor, BColor);

            test++;

        }

        else

        {

            z = Msg[test] / 100;

            x = (Msg[test] % 100) / 10;
```

```
                            y = Msg[test] % 10;
                            RColor = (RColor - RColor % 10) + z;
                            GColor = (GColor - GColor % 10) + x;
                            BColor = (BColor - BColor % 10) + y;
                            tmpColor = Color.FromArgb(RColor, GColor, BColor);
                            test++;
                        }
                    }
                    else
                    {
                        RColor = (RColor - RColor % 10) + 3;
                        tmpColor = Color.FromArgb(RColor, GColor, BColor);
                    }
                }
                else
                {
                    errorz++;
                    tmpColor = Color.FromArgb(tmpColor.R, tmpColor.G, tmpColor.B);
                }
                ImageArray[i, j] = tmpColor;
                bImage.GetBitmap().SetPixel(i, j, ImageArray[i, j]);
            }
        }
    }
    bImage.Save("d:\\Stego.png", ImageFormat.Png);
    return bImage.GetBitmap();
}
```

Appendix B: Secret Data

this is a test using 98300 characters repeated x times.