New Technique to Embed Encrypted QR Code in Colored Image by Using First and Third LSB

تقنية جديدة لتضمين رمز الاستجابة السريع داخل صورة ملونة باستخدام الرتبة الاولى والثالثة LSB

Student Name: Muataz Safauldeen Abdulrahman

Student Number: 401120152

Supervisor
Dr . Hebah H. O. Nasereddin

Master Thesis

Submitted In Partial Fulfillment of the Requirements of the

Master Degree in Computer Information Systems

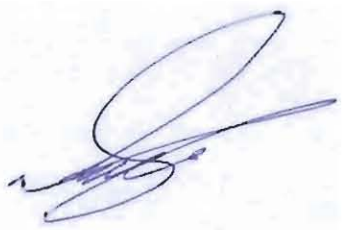
Faculty of Information Technology

Middle East University

May,2014

Authorization statement

I **Muataz Safauldeen Abdulrahman** Authorize the Middle East University to supply a copy my Thesis to libraries, establishment or individuals

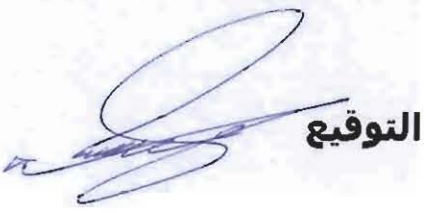
Signature 

Date: 2014-05-28

اقرار تفويض

اني معتز صفاءالدين عبدالرحمن افوض جامعه الشرق الاوسط بتزويد نسخ من رسالتي

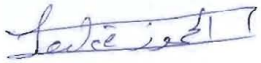
للمكتبات اوالمؤسسات او الهئات او الافراد عنده طلبها.

التوقيع 

التاريخ: 2014/05/28

Examination Committee Decision

This is to certify that the thesis entitled **"New Technique to Embed Encrypted QR Code in Colored Image by Using First and Third LSB "**was successfully defended and approved on. 28/5/2014

Examination Committee Member	**Signature**
1- **Dr . Hebah H. O. Nasereddin**	
2- **Dr . SADEQ ALHAMOUZ**	
3- **Dr . Thair Khdour**	

DEDICATION

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

To my country that have always been the science paradise

To my family who never stopped supporting me during the journey of my life and gave up a lot of things for me and taught me how to be patient, to my friends who have always been by my side, to my teachers who lightened my mind.

AKNOWLEDGEMENT

I would like to express my gratitude to my advisor for her support, patience, and encouragement throughout my graduate studies. It is not often that one finds an advisor and colleague that always finds the time for listening to the little problems and roadblocks that unavoidably crop up in the course of performing research. Her technical and editorial advice was essential to the completion of this dissertation and has taught me innumerable lessons and insights on the workings of academic research in general, I would also like to thank my family members and my dear friends.

Table of Contents

Table of Figures

List of Tables

Abbreviations

AES: Advanced Encryption Standard.

CO: Cover Object.

DCT: Discrete Cosine Transform.

DFT: Direct Fourier Transform.

DH: Data Hiding.

LSB: Least Significant Bit.

RGBA: Red, Green, Blue and Alpha.

RMSE: Root Mean Squared Error.

PNG: Portable Network Graphics.

PNSR: Peak Signal to Noise Ratio.

QR: Quick Response.

SD: Standard Deviation.

SI: Stego Image.

Abstract

Steganography is the approach by which data is hidden such that one cannot know about it and so data remains secure, this is usually done by hiding data inside an object through changing some properties of that object, this is often called the cover object.

In this thesis, a novel steganography technique based on Quick Response code (QR code) and colored images is proposed, the technique hides secret text in QR code, before hiding the QR inside the four channels PNG cover image, the QR is saved as black and white image with 1 bit per pixel and is stored in array of bytes to be encrypted using AES before embedding it inside the cover image.

The results of the technique was superior when using large data, despite the fact that the impact is greater when using the first and third bits than when using the least significant bit alone, the proposed technique was tested by 3 experiments using 600, 65536 and 131070 characters as secret text, the effect of the proposed technique on the picture was almost the same and even better in some images than LSB when hiding relatively large data.

الخلاصة

الاختزال نهج يتم إخفاء البيانات من خلاله بحيث لا يمكن للمرء معرفة ذلك، عادة يتم إخفاء بيانات حساسة داخل كائن من خلال تغيير بعض خصائص الكائن، غالبا ما يسمى الكائن بالغطاء. يقترح هذا البحث تقنية فريدة لإخفاء البيانات تعتمد على رمز الاستجابة السريع والصور الملونة، يتم تحويل النص السري المراد اخفائة الى رمز الاستجابة السريعة، ومن ثم يتم ادخالة داخل صوره PNG ، يتم حفظ رمز الاستجابة السريع كصورة بالأبيض والأسود بحيث يتم استخدام 1 بت لكل بكسل ويتم تخزينها في صفوف من البايت لتكون مشفرة باستخدام AES قبل تضمينه داخل صورة الغلاف، بعد دراسة النتائج تبين ان هذه الطريقة تعطي نتائج افضل من الـ LSB عند تخزين حجم كبير من البيانات.حيث تم فحص الطريقة من خلال ثلاثة تجارب باستخدام 3 نصوص مختلفة من طول 600, 65536 و 131070 حرف, وعلى الرغم من أن التأثير عند استخدام البتين الأول والثالث اكثر من استخدام LSB وحده، وكان تأثير التقنية المقترحة على الصورة نفسها تقريبا وحتى أفضل في بعض الصور من LSB عندما يتم تخبئة بيانات كبيرة نسبيا.

Chapter One

Chapter 1 Introduction

Introduction

Secure storage and transfer of data has always been a concern since the existence of human being, this concern has been the main focus in information security. Information security has two main approaches, the first is called data encryption which deals with performing some transformations on the data in order to secure it, these transformations can include adding of redundant data and noise and thus making the data unreadable.

The second approach is called steganography, it refers to the technique by which data is hidden such that one cannot know about it and so data remains secure, this is usually done by hiding viable data inside an object through changing some properties of that object, this is often called the cover object, one cannot detect the existence of this data by simply looking at that object (Jantan, 2008).

The change in properties of the object whether it is image, audio file or text file has to be minimal so that no noticeable noise can be observed, otherwise it can result in revealing the existence of the hidden data/message to the attacker, this puts a limitation on the payload of data to be saved (Nath, 2012).

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 and 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.

Image steganography is the focus of this research study, this study will also introduce the use of Quick Response Code (QR Code) for holding a text data before encrypting and then embedding the QR Code inside a colored image.

1.1 Images Steganography

Images steganography is the major concern of this study and it can be defined as the art of hiding information inside an image through manipulating bits that when changed will have no noticeable effect, and then the image that contains the hidden information can be exchanged without exposing the existence of the hidden information, this technique exploits the property of human eyes that are less sensitive to some color changes than others and are unable to distinguish between two colors when the difference in value is too small (Morkel T. et al 2006).

A stego image contains viable information that can be in form of text or image, this information can be as critical as plans for a company or design for new chip product. In colored images, a pixel, which is the smallest component in an image, is expressed in a form of a three color values being Red, Green and Blue (RGB).

Colored pixel is obtained by superposing these three color values, colored pixels are then drawn to form an image, however, these values (RGB) are represented using binary numbering system in computers, the change of value of a bit can be used to store the secret data, the position of the bit changed would affect the amount of change on that pixel color so selecting the appropriate bit is critical to keep the noise in the cover file minimal. One of the famous techniques is the replace of the least significant bit (LSB) in each color value to store the message, this technique will have low impact on the individual color but will spread over the image due to the fact that one needs 8 bits, or 8 colors, to store one character. However, image of the size (640x480 using 256 color) can hide up to 300KB worth of data (Rabah K., 2004).

1.2 Portable Network Graphics

Portable Network Graphics (PNG) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for

Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet.

PNG supports palette-based images (with palettes of 24-bit RGB or 32-bit RGBA colors), grayscale images (with or without alpha channel), and full-color non-palette-based RGB[A] images (with or without alpha channel). PNG was designed for transferring images on the Internet, not for professional-quality print graphics, and therefore does not support non-RGB color spaces such as CMYK.

1.3 Quick Response Code

QR Code (Quick Response Code) is a special type of two-dimensional barcode designed by Japanese automobile industry. The idea was first proposed by Denso Wave, QR Code is usually attached to an item and it contains information about that item, information can be in form of numeric data, alphanumeric and binary, this makes the QR Code capable of storing theoretically any kind of data as long as they are represented in binary.

QR Code system has fast readability and great storage capacity which made it popular outside the automotive industry, it is made of black dots arranged in a matrix-like order with white background, this design makes it easier for imaging devices to capture, correct and interpret data stored in QR Codes, the first module of QR Codes first used in 1997 by Association for Automatic Identification and Mobility (AIM) and it is being widely used ever since.

According to Suppat Rungraungsilp, QR Codes has maximum approximate capacity of 7089 characters (Suppat Rungraungsilp, 2012).

QR Generation Process:

1- Data Analysis and Encoding:

The QR encoding standards have 4 different models to encode data, these models are numeric, alphanumeric, byte, and Kanji. The output of any model is a series of bits, each model follows a different technique for generating the series of bits, since each model behaves differently and generate different length of bits the data should be analyzed to determine whether the text is preferred to be encoded in numeric, alphanumeric or byte mode, then select the most optimal model in terms of bits length to generate the stream of bits to generate the QR code.

2- Error Correction Coding:

After encoding the secret text, error correction codes are generated to be embedded inside the bits stream to detect and correct errors , QR scanners read both the codewords and data and compare them together to detect errors.

3- Module Placement in Matrix:

After embedding the codewords inside the data bites, the data must be placed in the QR matrix along with pattern commonly used with QR codes such as black boxes in corners.

4- Data Masking:

Specific patterns if found in QR matrix can make it difficult for QR scanners to correctly read the data, to avoid such case, QR code standards provide 8 different mask patterns that alters the QR code according to a particular pattern, the mask must be selected so that difficulties are minimal.

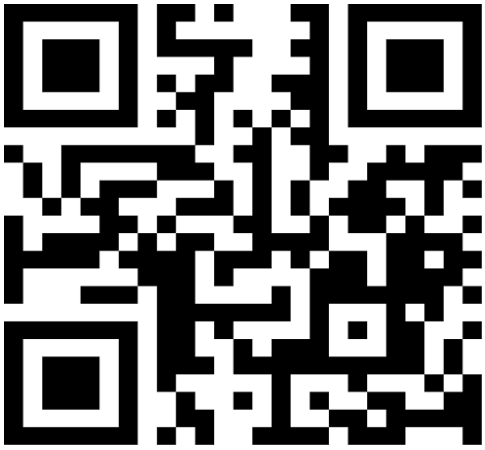


Figure 1.1. QR Code

1.4 Advanced Encryption Standard

The Advanced Encryption Standard, (AES) is a standard for the encryption of digital data; it was proposed by Joan Daemen and Vicent Rijmen under the name Rijndael cipher during the selection process by the U.S National Institute of Standards and Technology (NIST) in 2001.

Rijndael is now widespread and is used by governments, essentially Rijndael is a symmetric key algorithm that requires both parties the sender and the receiver to share the same key of 128, 192 and 256 bits in length, the encryption is performed on blocks each block size is 128 bits (Rijmen, 2002).

AES operates on a 4×4 column-major order matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field.

The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of cycles of repetition are as follows:

- 10 cycles of repetition for 128-bit keys.
- 12 cycles of repetition for 192-bit keys.
- 14 cycles of repetition for 256-bit keys.

1.5 Problem Statement

Secure storage and transfer of data has always been a concern in the field of security, this concern has led to several developments in the field of steganography and cryptography, however, proposed studies in steganography field tend to focus more on the mechanism of hiding secret data to achieve better security rather than the size of the hidden data, the goal of this thesis is to study the effect of reducing the size of the secret text by converting it to QR Code and encrypting it on the stego images and evaluate its usability to achieve better stego image quality.

1.6 Limitations

Limitations of this approach are:

1. Specific format of images have to be used.
2. QR Codes have limited capacity before reading becomes hard.
3. QR Code generation takes time for long messages.
4. Availability of QR Code readers.

1.7 Objectives

1- Reduce the effects of embedding on the stego image by reducing the size of secret data before hiding it thought converting it to QR Code.

2- Achieve better security by encrypting QR code before hiding.

3- Study the effect of using the first and third LSBs for insertion.

1.8 Problem Motivation

This study addresses a special category that is image steganography, there is still no optimal technique used to solve the problem of image steganography, colored images are the main concern of this study.

Although the topic of steganography exists since ancient times, the topic is still important in new days and is motivated by the process of watermarking to protect copyrights of multimedia on the Internet.

Chapter Two

Chapter 2 Related Work

(Zou & Shi, 2005) A novel formatted text document data hiding algorithm. Called Inter-word Space Modulation (ISM) scheme, is proposed in which the spaces between neighboring words are modulated to hide data. In contrast to prior arts, this method does not require original documents for hidden data extraction. The hidden data are robust to printing, copying and scanning. The experiments show that after printing, ten times of repeated copying, followed by scanning, the hidden data can still be extracted without a single bit error. It is expected that it can find wide applications for secure document processing, including digital notarization. Three different methods for formatted text document data hiding: line shift coding, word shift coding and feature coding. Line shift coding and word shift coding are robust to printing, copying and scanning to some extent. The major drawback is that the original intact document is needed for hidden data extraction which may not be available in many cases. Also the paper mention a baseline detection method for line shift coding which did not require the original document. However, as pointed out by the authors themselves, it is not reliable to printing, copying and scanning. Besides, the embedding capacity is about one bit per two lines.

(Asif, Shaikh, Manza, & Ramteke, 2010) The comparison of original text in the form of bitmap image and extracted image by using various fonts of text as a bitmap image. In image the basic objective of data hiding is to store as much as data in the host image without degrading the quality of the host image and which will be reconstructed again without compromising the loss of source image data and the actual hided information. Out of which the most emerging area is hiding the data into different media files such as image, audio, video, etc. In these media files the image is considered as the most suitable file format for the data processing. The study has done the preparation of the text data set of size 20 characters in single font type, variable font sizes

and the color of text as black. The bitmap image can be hided into any color image source which will acts as a medium of carrier of the text data. This color image is further decoded to get the actual data of 20 characters without any loss if possible. The experimental steps used for text data hiding in images is done with above data set and image processing functions of MATLAB. The data set of a single color source image and the data set of 2 to 3 sentence each of 24 characters with above specification. The result is found to be most satisfactory and prominent in the font VERDANA in the font size of 26 to 30 resulted into 85% to 90% of reconstruction rate of actual hided text data.

(Dutta, Bhattacharyya, & Kim, 2009) Data hiding in Audio Signal: a review. This paper introduced a robust method of imperceptible audio data hiding. This system is to provide a good and efficient method for hiding the data from hackers and sent to the destination in a safe manner. This proposed system will not change the size of the file even after encoding and also suitable for any type of audio file format. The proposed idea is to hide secret message within audio signal using with a stego key, to retrieve the embedded message should be using the extractor with the same stego key. This paper conclude 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, tamper detection. As the sky is not limit so is not for the development. Man is now pushing away its own boundaries to make every thought possible. So similarly these operations described above can be further modified as it is in the world of Information Technology. After designing any operation every developer has a thought in his mind that he could develop it by adding more features to it.

Least Significant Bit (LSB) technique is the most used technique for steganography, the idea is to replace the 8^{th} bit in a byte with a bit from data to be hidden, (Younes&Jantan2008).

This technique does not increase the image size and the possibility of changing the value of the least significant bit in a pixel is 50% (Jantan, 2008). However, new techniques showed some advantages of using the two least significant bits instead of the least significant bit alone, Figure 2.1 shows the process of LSB insertion

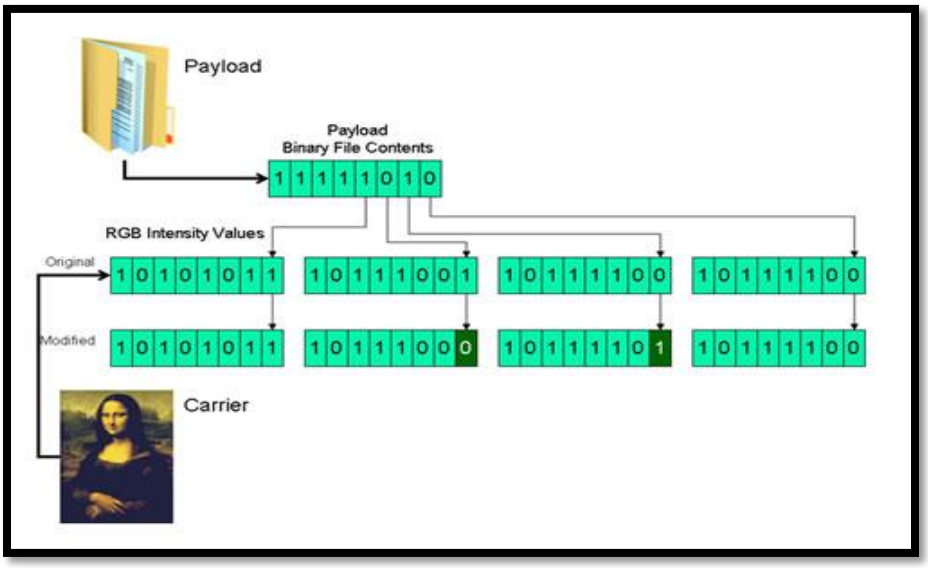


Figure 2.1. Steganography using LSB (Jantan, 2008)

(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).

Ross J. Anderson and Fabien A.P. Petitcolas (2007) claims that every steganographic approach has its own limitations, another method is proposed by H. Motameni and his colleague's that suggests using dark corners of an image for embedding (Motameni H., 2007).

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. Also they 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.

Nath, et all, 2012) developed several information security systems that combine both cryptography and steganography, in his technique ASA_QR Nath and his co-authors presented an algorithm to hide small encrypted messages inside QR Codes, the QR Code is then randomized before being embedded inside the cover image, ASA_QR is a combination of an encryption and data hiding in two stages (Nath, 2012).

(SuppatRungraungsilp, 2012) argues that 2D barcodes (QR Codes) watermarks is an interesting research in the security field, he proposed QR embedded technique for invisible watermarking by using Discrete-Cosine-Transformation (DCT), DCT and DFT (Discrete Fourier transform) allows QR Code images to be divided to different frequency bands by using blocks DCT and DFT based techniques (Suppat Rungraungsilp, 2012).

(Suraj Kumar Sahu, 2013) used QR Codes to send encrypted data to receiver, he used QR Codes as cover images to hide data inside them, he argues that QR Codes can work perfectly due to their storage capacity, he claims that they can hold any type of data and they can be used in business, advertising and social networking (Sahu, 2013).

(Abhas Tandon et.al, 2013) proposed the usage of QR Codes to design secure one time password (OTP) scheme for authentication in Net-Banking, he argues that distribution of OTPs to concerned user is a major issue, short message service is the most common way for OTP distribution, he proposes a secure OTP using QR Codes and email (Tandon, 2013), Figure 2.2 shows the workflow of the QR code based OTP.

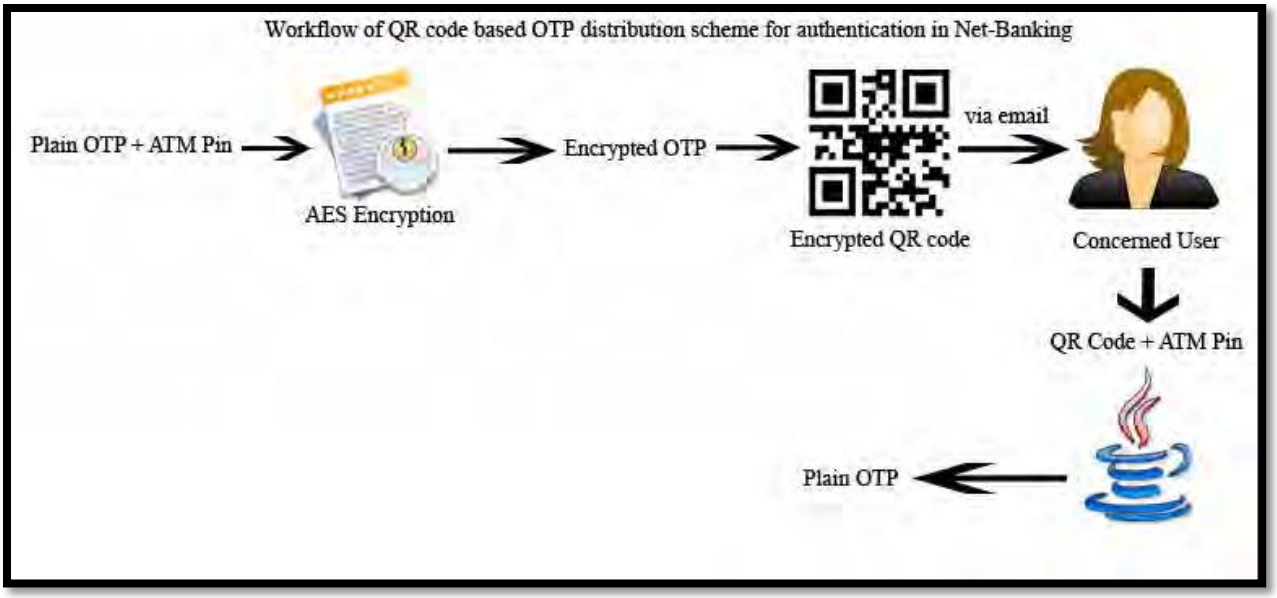


Figure 2.2. Workflow of QR Code based OTP (Tandon, 2013)

The system satisfies the high security requirements of the online users and protects them against various security attacks. Also the system does not require any technical pre-requisite and this makes it very user-friendly.

(Somdip Dey, 2012) used QR Codes to store encrypted secret messages with passwords, he suggested using this technique in the government sector or for storage of important personal data, he achieved his technique by entering the message with a password which generates a key that is added to each alphabet in the numbers or text entered in the message which is needed to be encrypted and then the first phase of encryption is generated, the newly generated encrypted text is then encrypted again using other techniques to generate the final encryption message (Dey, 2012), Figure 2.3 shows the encryption scheme in QR Code Using Steganography.

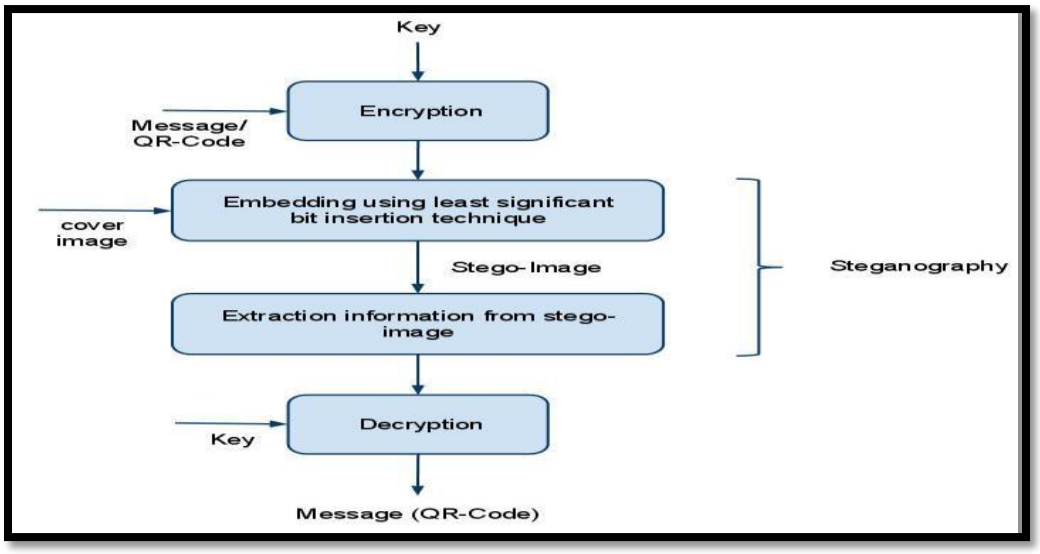


Figure 2.3. Encryption in QR Code Using Stegnography (Dey, 2012)

This method has a very large scope. Since Unicode format is used for encryption, this method can be used to encrypt any type of message or file (picture, video, audio, etc.) and send it to the receiver safely or the method can also be used to store important data or information safely.

Chapter Three

Chapter 3 Proposed Technique

3.1 Overview

Different techniques for data hiding are being discussed in the previous chapter, this chapter proposes a new technique for data hiding, the proposed technique hide secret text in QR code, before hiding the QR inside the 4 channels PNG cover image, the QR is saved as black and white image with 1 bit per pixel and is stored in array of bytes to be encrypted using AES before embedding it inside the cover image, the cover image has to be of type PNG with alpha channel, the first pixel of the stego image will contain the length of the message, the actual embedding starts after the first pixel the first and third bit of every color in the cover image is replaced with 2 bits of the encrypted QR code.

Steps:

1- Selecting cover image to hide QR code inside it
2- Inserting secret text to be hidden
3- Inserting key used for encryption using AES
4- Generating QR Code from the secret text.
5- Reading QR code as stream of bytes.
6- Store the length of the secret stream in the first pixel.
7- Encrypting QR Code, to provide extra security.
8- Embedding the encrypted QR Code in a PNG colored image; in the first and third bit of each channel in the pixel.

Most of available techniques deal with gray scale images, this study however will deal with colored images, in this context there are two aspects should be considered in order to perform the embedding inside colored images, these are, choosing the right pixel for embedding, the second level of security achieved by embedding. Below show more details for the technique.

In this technique the text to be transferred is embedded inside a QR code and is then encrypted using AES to provide extra security in case of detection of existence, this QR code is then embedded inside of the cover image for safe transfer and can be extracted and decrypted using a shared key later by the receiver, Figure 3.1 shows the flowchart of the proposed technique

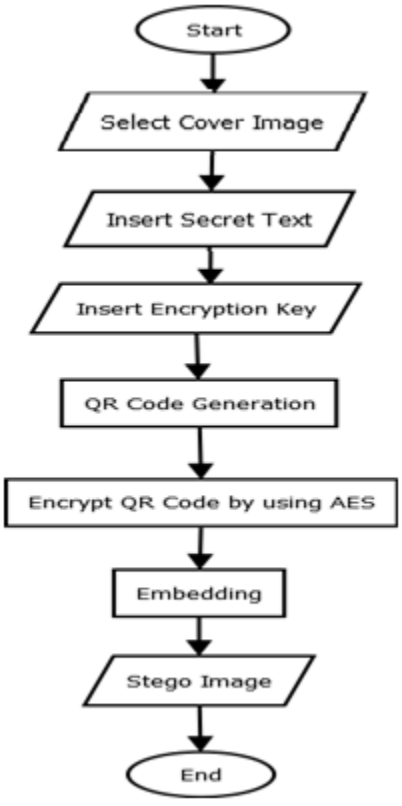


Figure 3.1. Proposed Technique Flowchart.

3.2 QR Generation and Encryption

After entering the secret text to be transferred, we would embed it into QR code to take advantage of its widespread use and of its error correction property, this could also give us the advantage of storing the text in QR format, the QR Code is then encrypted using AES and a shared key to increase the security and hardness of message retrieval in case the existence of the message is revealed, Figure 3.2 show the flowchart of QR Code generation and encryption process.

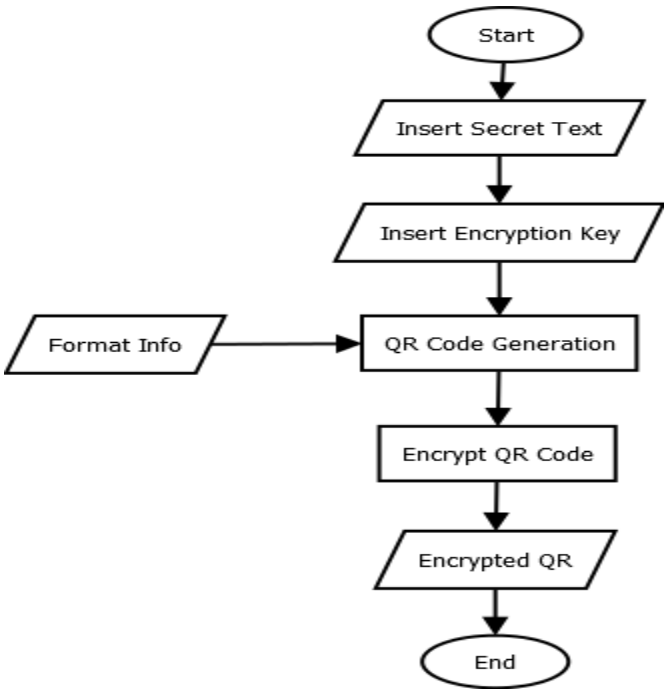


Figure 3.2. QR Code Generation and Encryption flowchart

- **QR Generation Process:**

1- Data Analysis and Encoding:

The QR encoding standards have 4 different models to encode data, these models are numeric, alphanumeric, byte, and Kanji. The output of any model is a series of bits, each model follows a different technique for generating the series of bits, since each model behaves differently and generate different length of bits the data should be analyzed to determine whether the text is preferred to be encoded in numeric, alphanumeric or byte mode, then select the most optimal model in terms of bits length to generate the stream of bits to generate the QR code.

2- Error Correction Coding:

After encoding the secret text, error correction codes are generated to be embedded inside the bits stream to detect and correct errors , QR scanners read both the codewords and data and compare them together to detect errors.

3- Module Placement in Matrix:

After embedding the codewords inside the data bites, the data must be placed in the QR matrix along with pattern commonly used with QR codes such as black boxes in corners.

4- Data Masking:

Specific patterns if found in QR matrix can make it difficult for QR scanners to correctly read the data, to avoid such case, QR code standards provide 8 different mask patterns that alters the QR code according to a particular pattern, the mask must be selected so that difficulties are minimal.

Figure 3.3 show the QR Code.

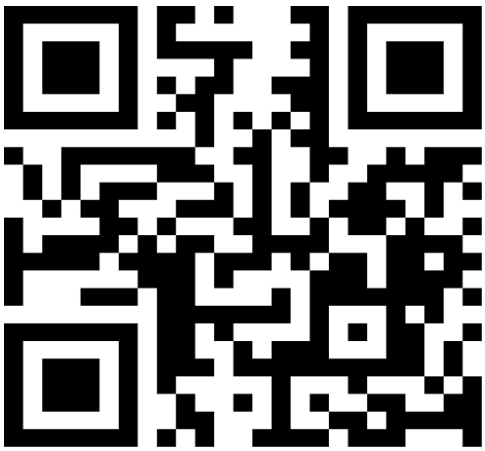


Figure 3.3. QR Code

- **QR Encryption Process:**

1- Read the QR code as stream of bytes.

2- Start the encryption process:

a. Key Expansion: the first step in the encryption technique is to generate round keys that’s derived from the main key using Rijndael's key schedule. AES requires a separate 128-bit key for each round plus one more and 176 extra bytes

b. Add Round Key: each byte of the secret data is combined with a block of the round key using bitwise xor.

c. Rounds:

- Sub Bytes: replacing each byte with another according to a lookup table.
- Shift Rows: shifting the last three rows cyclically for a number of steps.
- Mix Columns: combining the four bytes in each column.
- Add Round Key

d. Final Round (no Mix Columns)

- Sub Bytes
- Shift Rows
- Add Round Key.

3.3 Embedding

Take a colored image and store its components in four arrays, the first array would be the red components of the image, the second is green, the third is blue and the fourth is the alpha channel, the embedding is done by replacing the first and the third least significant bit of each component with bit of the QR code that contains the text, Figure 3.4 show the steps of embedding.

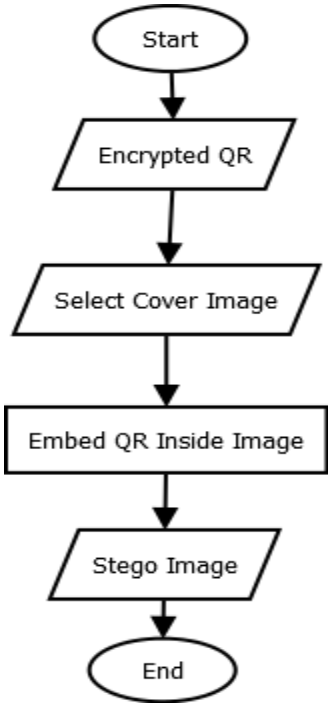


Figure 3.4. Embedding Process

- Embedding Steps:

1- Use the encrypted QR:

Use the encrypted QR Code that's generated from the secret text.

2- Choosing the cover image:

The chosen image has to be of type PNG with alpha channel, each channel will save 2 bits of the QR code, the image have to be of accepted size to be able to store the code.

3- Reading the QR data:

The QR Code is saved as black and white image, the image is then read and stored into a series of bytes.

4- Storing the length of the series:

The first pixel of the cover image will contain the length of the secret data series, the length will be presented using 24 bits. This puts a limit on the length that can be stored.

5- Embedding the secret data:

The secret data bytes are embedded inside each pixel of the cover image using the four channels (RGBA), the embedding is made through the replacement of the first and third bit of the cover image by 2 bits of the secret data which gives the capacity of 1 byte per pixel. Figure 3.5 show the embedding process.

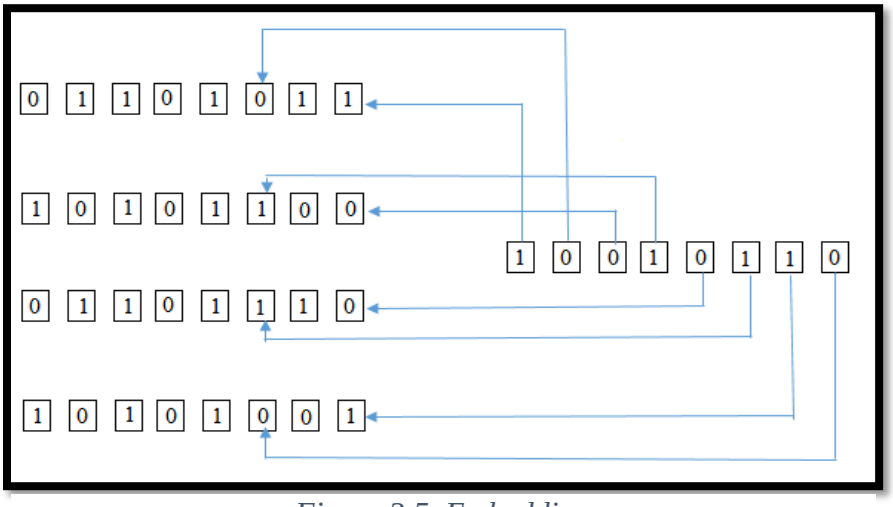


Figure 3.5. Embedding

3.4 Extracting

After receiving the PNG stego image, the following steps are required in order to extract the secret message from the image, the following Figure 3.6 show the flowchart of the extracting and decrypting the QR Code.

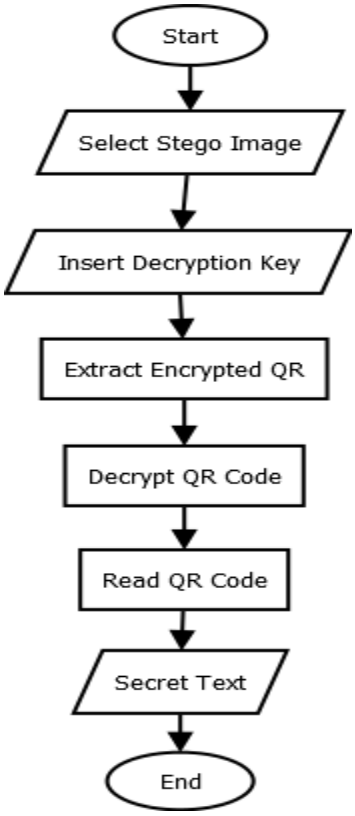


Figure 3.6. Extracting and Decrypting QR Code Flowchart.

- Extracting Steps:

1- Retrieving the length of the secret message:

The length of the data is stored in the first pixel of the stego image, it is formed of 24 bits stored in Red, Green and Blue.

2- Extracting the encrypted data series:

The extraction from the image is done by taking the first and third least significant bits from each channel to build the original byte, Figure 3.7 show the extraction process from the pixel channels.

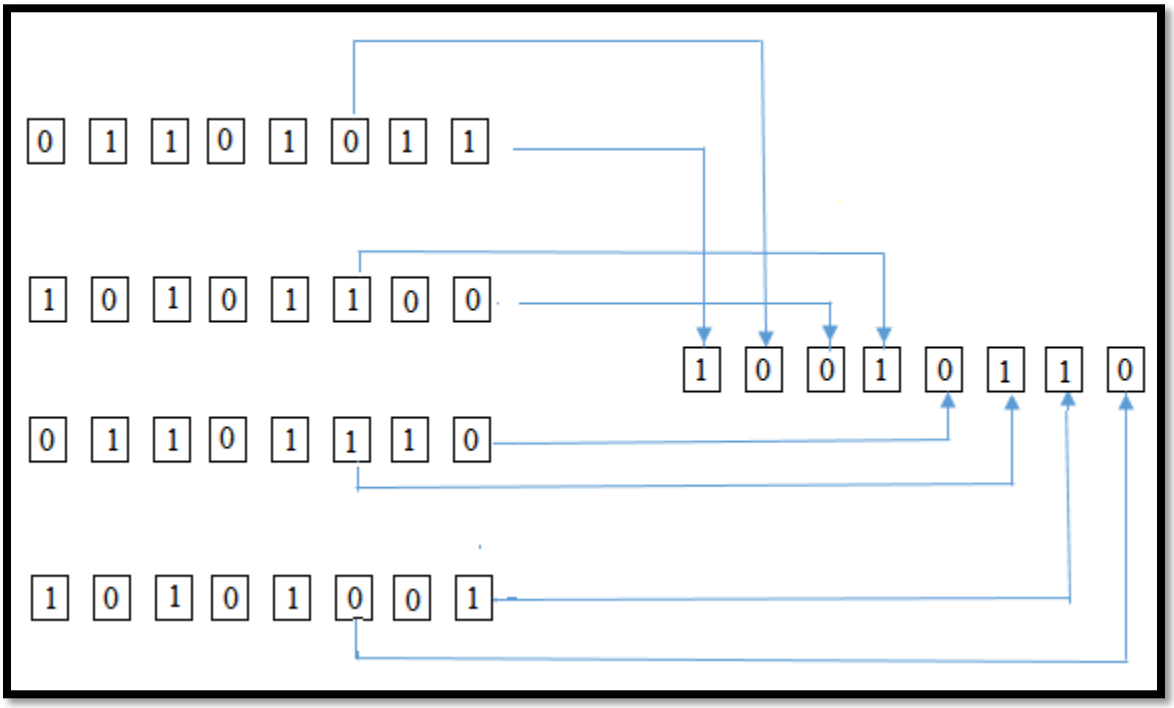


Figure 3.7. Extracting

3- Decrypting and reading the QR:

The QR code is decrypted using the shared key between the sender and the receiver, after decrypting the bytes of the QR code, a code scanner is needed to read and extract the data from the QR Code image.

Chapter Four

Chapter 4 Experimental Results

A series of experiments were conducted to show the effectiveness of the proposed technique. The efficiency of the proposed technique is measured by four parameters which are:

1. Payload of secret data hidden in the cover image after stored inside the QR Code:

   It is the size of secret text hidden inside the image.

2. Root Mean Squared Error (RMSE) of cover and stego images:

   RMSE is the measure of the differences between values predicted by a model or an estimator and the values actually observed.

3. Peak Signal-to-Noise Ratio (PSNR) of cover and stego images:

   Peak signal-to-noise ratio is a term used to indicate the ratio between the maximum possible quality of an image and the amount of corrupting noise that affects the fidelity of its representation.

4. Standard Deviation between cover and stego images.

   Standard Deviation shows how much values variation or dispersion from the average exists.

5. Amount of pixels that contain error (Error Number).

   The amount of pixels that contain difference between its values in the stego image before and after embedding of secret data.

The objective of measuring the payload of data hidden in the cover image is to show that the proposed technique can hide a relatively large payload of data, which will be compared to results obtained using DiffImgPortable.

The PSNR is used to measure the quality of stego image compared to the cover image. The quality of the image is higher if the PSNR value of the image is high. Since PSNR is inversely proportional to RMSE value of the image, the higher the PSNR value is, the lower the RMSE value will be. Therefore the better the stego image quality is the lower the RMSE value will be, RMSE can be calculated using the following equation as stated by (Mohammad Ali Bani Younes, & Aman Jantan, 2008).

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

Where:

$X_{1,t}$: the value of pixel number (t) in the first image.

$X_{2,t}$: the value of pixel number (t) in the second image.

N : The total number of pixels.

PSNR is calculated according to the following formula as stated by (Mohammad Ali Bani Younes, & Aman Jantan, 2008):

$$\text{PSNR} = 20 \cdot \log_{10}\left(\frac{255}{RMSE}\right) \quad \ldots\ldots (2).$$

Experimental Setup

The simulation for the experiment was set up and run on a Windows 7 Home on 2.00 GHz Celeron with 2.5 GB of RAM.

In the experiments 4 PNG images of size 512 × 512 and 3 different messages with different lengths been used to test the proposed technique, the first experiment was performed using text of 600 characters, the second experiment is performed using 65536 characters and the final third experiment is done using 131070 characters, the results was obtained by helper application Differential Image Portable to calculate different parameters, the application is used in scientific studies on images and calculate the discussed values.

Experiments

In this section the PSNR, RMSE, SD, Mean Error, Payload and Error Number values of each experiment are being reviewed, the experiments are performed on 4 different images of size

512 × 512 of type 4 channels PNG.

First Experiment

First Image

The first experiment was performed on the Sea image, embedding a text of length 600 characters, the embedding was performed using the proposed technique and the LSB technique, in this section, Mean Error, Standard Deviation, RMSE and PSNR parameters are calculated after the embedding process for each technique.

Figure 4.1 shows the sea image before and after embedding 600 characters using the proposed technique

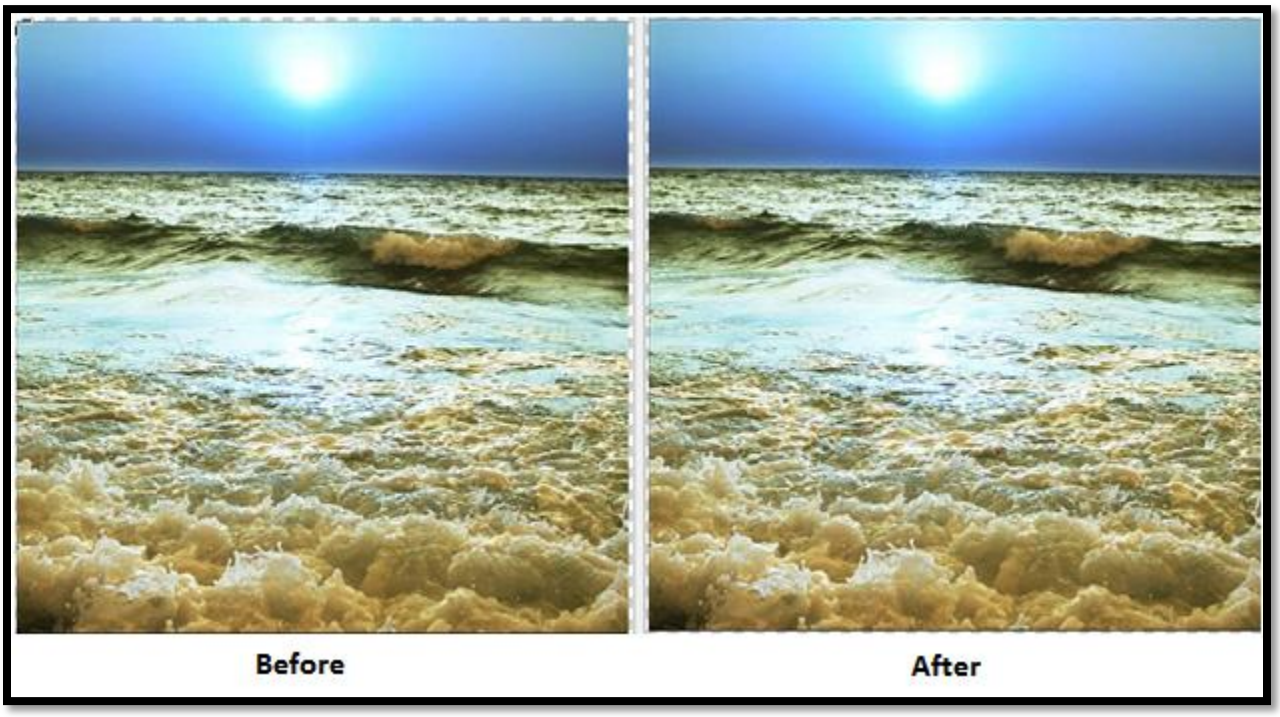


Figure 4.1. Sea before and after first experiment using the proposed technique

Table 4.1 shows the values of the studied parameters after storing 600 characters inside Sea using the proposed QR based technique and the LSB technique.

	Proposed Technique	LSB Technique
Mean Error	0.01207	0.00146
Standard Deviation	0.17765	0.04099
RMSE	0.17806	0.04102
PSNR	63.119	74.870
Error Number	1587	354

Table 4.1. Statistics for Sea first experiment after embedding.

Figure 4.2 is the histogram of Sea image after embedding the secret message in the first experiment using the proposed technique.

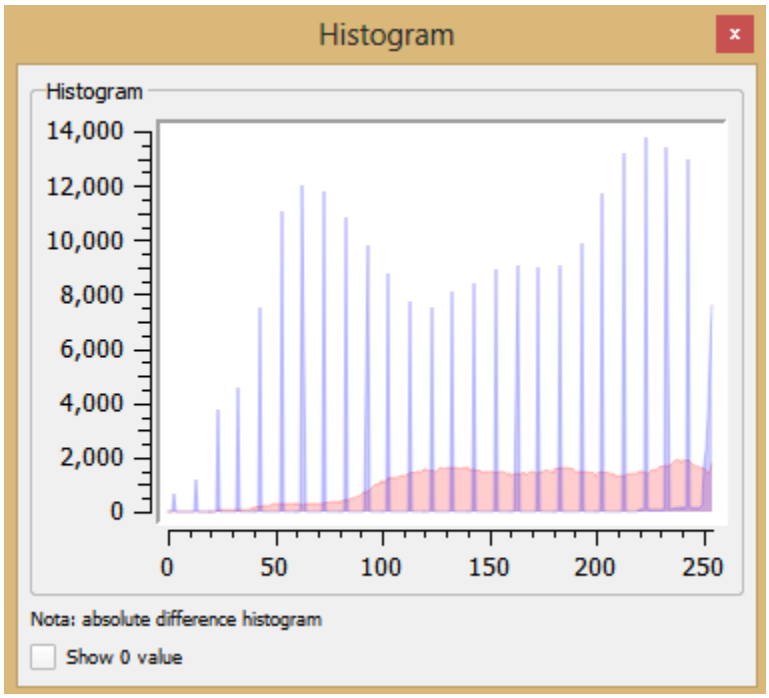


Figure 4.2. Histogram for Sea in first experiment

Second Image

Figure 4.3 shows Peppers before and after embedding 600 characters using the proposed technique

Figure 4.3. Peppers before and after first experiment using the proposed technique

Table 4.2 contains the results after storing 600 characters inside Peppers using the proposed QR based technique and LSB Method.

	Proposed Method	LSB Method
Mean Error	0.01177	0.00135
Standard Deviation	0.17151	0.03884
RMSE	0.17191	0.03884
PSNR	63.424	76.345

Error Number	1605	334

Table 4.2. Statistics for Peppers first experiment after embedding

Figure 4.4 shows the histogram of Peppers after embedding the secret message of the first experiment using the proposed technique.

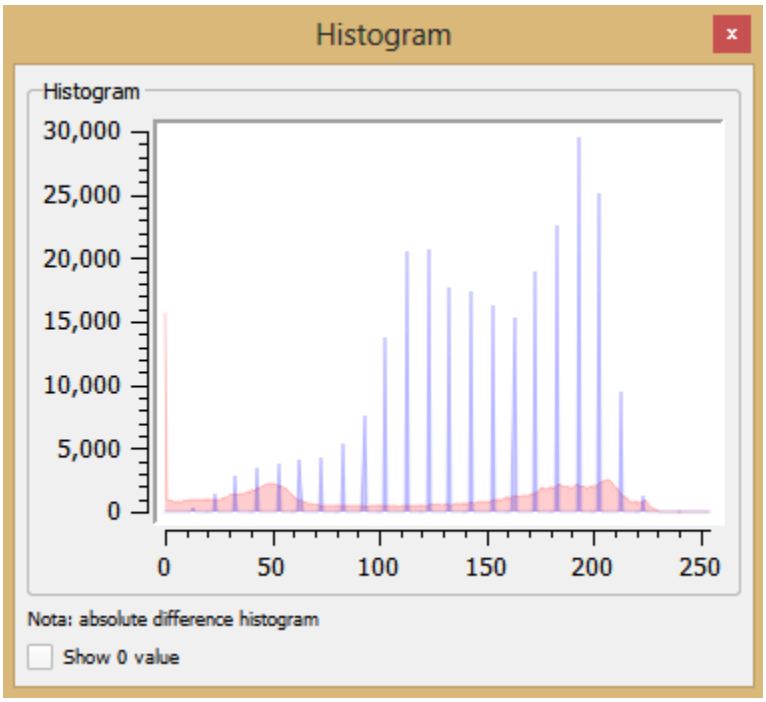


Figure 4.4. Histogram for Peppers in first experiment

Third Image

Figure 4.5 shows Lena before and after embedding 600 characters using the proposed technique

Figure 4.5. Lena before and after first experiment using the proposed technique

Table 4.3 contains the results after storing 600 characters inside Lena.

	Proposed Method	LSB Method
Mean Error	0.07067	0.00898
Standard Deviation	0.26320	0.04763
RMSE	0.27252	0.04847
PSNR	59.422	74.421
Error Number	1966	1008

Table 4.3. Statistics for Lena first experiment after embedding using proposed technique

Figure 4.6 shows the histogram Lena after embedding the secret message of the first experiment using the proposed technique.

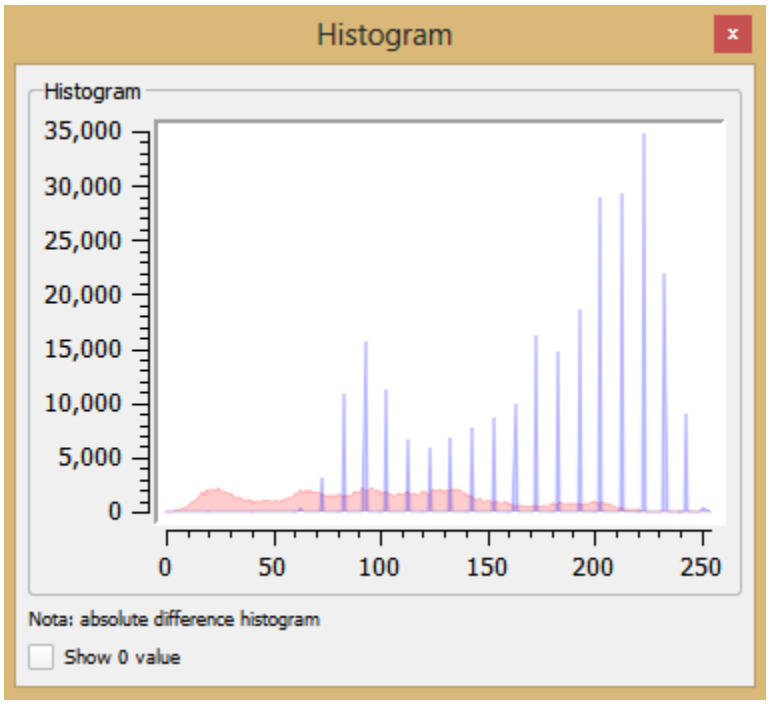


Figure 4.6. Histogram for Lena in first experiment.

Fourth Image

Figure 4.7 shows Baboon before and after embedding 600 characters of secret text using the proposed technique

Figure 4.7. Baboon before and after first experiment using the proposed technique.

Table 4.4 is the results after storing 600 characters inside Baboon using the proposed QR based technique.

	Proposed Method	LSB Method
Mean Error	0.07130	0.00914
Standard Deviation	0.26353	0.04804
RMSE	0.27301	0.04890
PSNR	59.407	74.344
Error Number	1971	1040

Table 4.4. Statistics for Baboon first experiment after embedding using proposed technique

Figure 4.8 shows the histogram Baboon after embedding the secret message of the first experiment using the proposed technique.

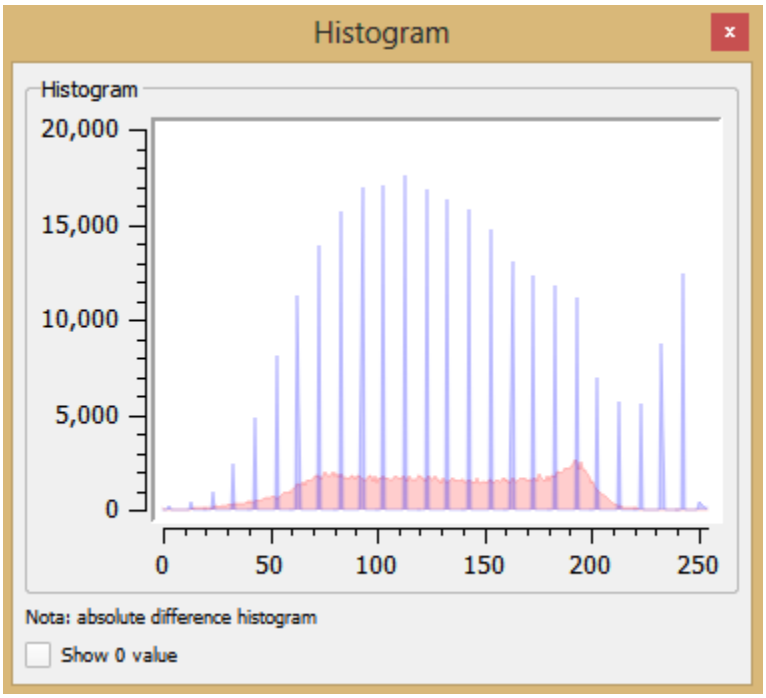


Figure 4.8. Histogram for Baboon in first experiment

After studying the results in the first experiments for hiding 600 characters inside the image it can be noted that the LSB method perform less effect on the stego image.

Second Experiment

First Image

Figure 4.9 shows Sea before and after embedding 65536 characters of secret text using the proposed technique

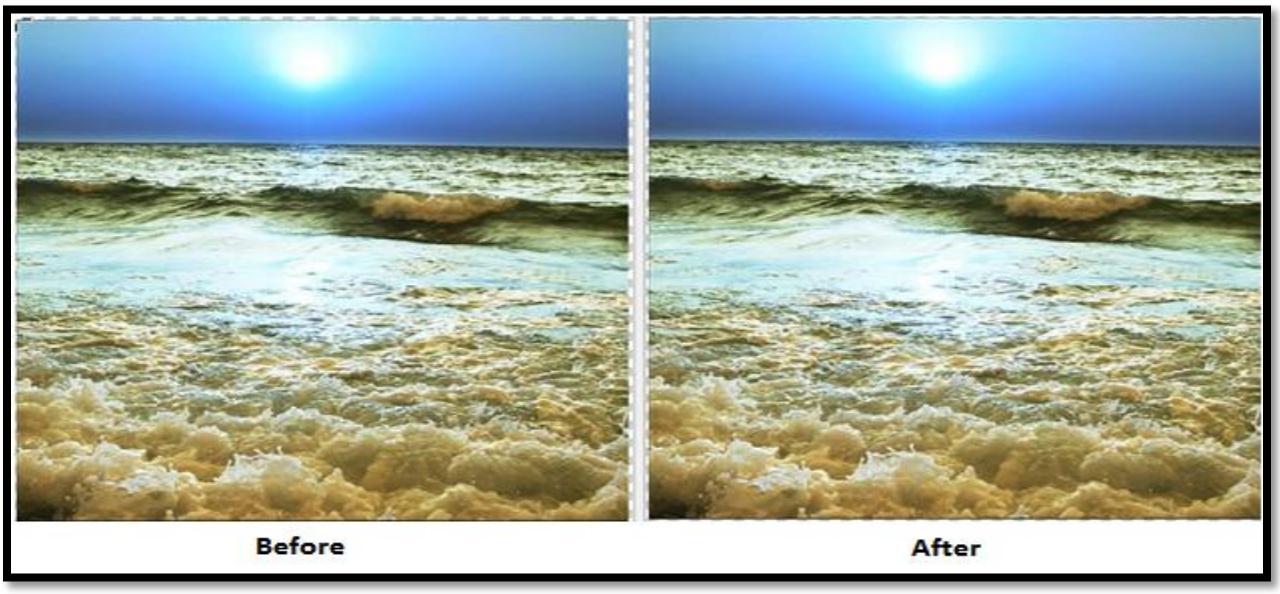


Figure 4.9. Sea before and after the second experiment using the proposed technique

Table 4.5 are the results after storing 65536 characters inside Sea using the proposed QR based technique.

	Proposed method	LSB method
Mean Error	0.69671	0.99604
Standard Deviation	0.17151	0.43704
RMSE	0.81612	1.08770
PSNR	49.895	47.40
Error Number	19388	111731

Table 4.5. Statistics for Sea second experiment after embedding

Figure 4.10 shows the histogram of Sea after embedding the secret message of the first experiment using the proposed technique.

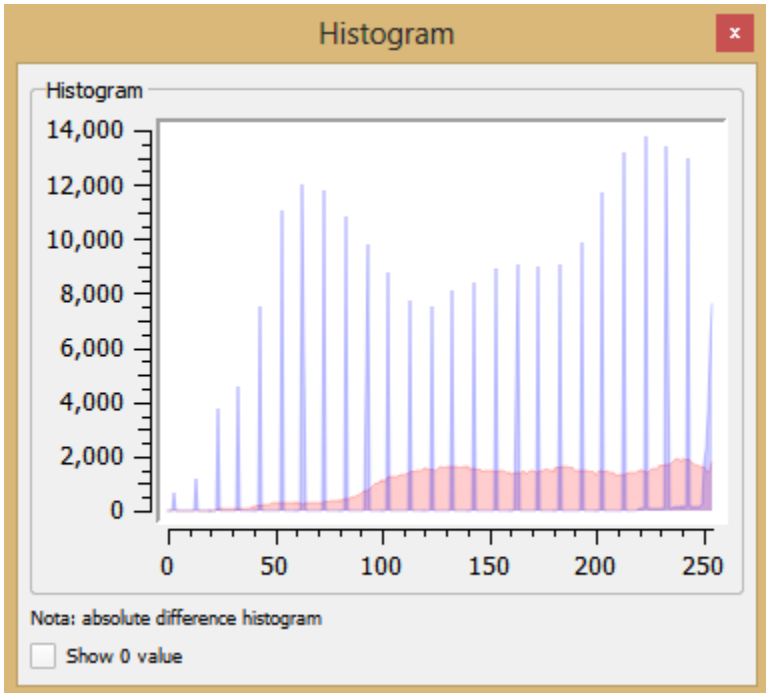


Figure 4.10. Histogram for Sea in second experiment

Second Image

Figure 4.11 shows Peppers before and after embedding 65536 characters of secret text using the proposed technique

Figure 4.11. Peppers before and after the second experiment characters using the proposed technique

Table 4.6 is the results after storing 65536 characters inside Peppers using the proposed QR based technique.

	Proposed method	LSB method
Mean Error	0.69010	0.97805
Standard Deviation	0.81041	0.43442
RMSE	1.06442	1.07019
PSNR	47.588	47.541
Error Number	19390	110618

Table 4.6. Statistics for Peppers second experiment after embedding using proposed technique

Figure 4.12 shows the histogram of Peppers after embedding the secret message of the first experiment using the proposed technique.

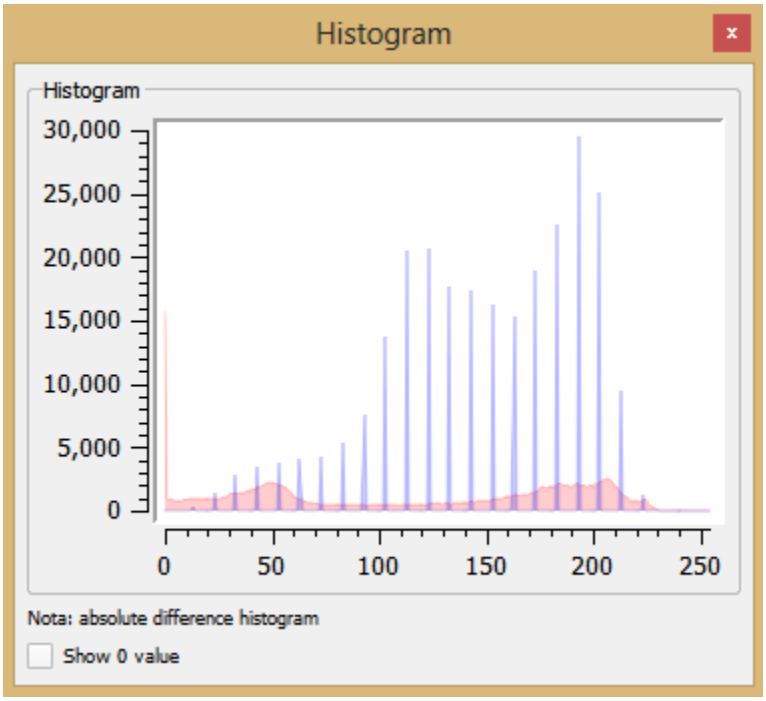


Figure 4.12. Histogram for Peppers in third experiment

Third Image

The following picture is Lena before and after embedding 65536 characters of secret text using the proposed technique

Figure 4.13. Lena before and after the second experiment characters using the proposed technique

Table 4.7 contains the results after storing 65536 characters inside Lena using the proposed QR based technique.

	Proposed method	LSB method
Mean Error	0.69652	0.98872
Standard Deviation	0.81577	0.43598
RMSE	1.07267	1.08058
PSNR	47.521	47.457

Error Number	19380	111230

Table 4.7. Statistics for Lena second experiment after embedding using proposed technique

Figure 4.14 shows the histogram of Lena after embedding the secret message of the first experiment using the proposed technique.

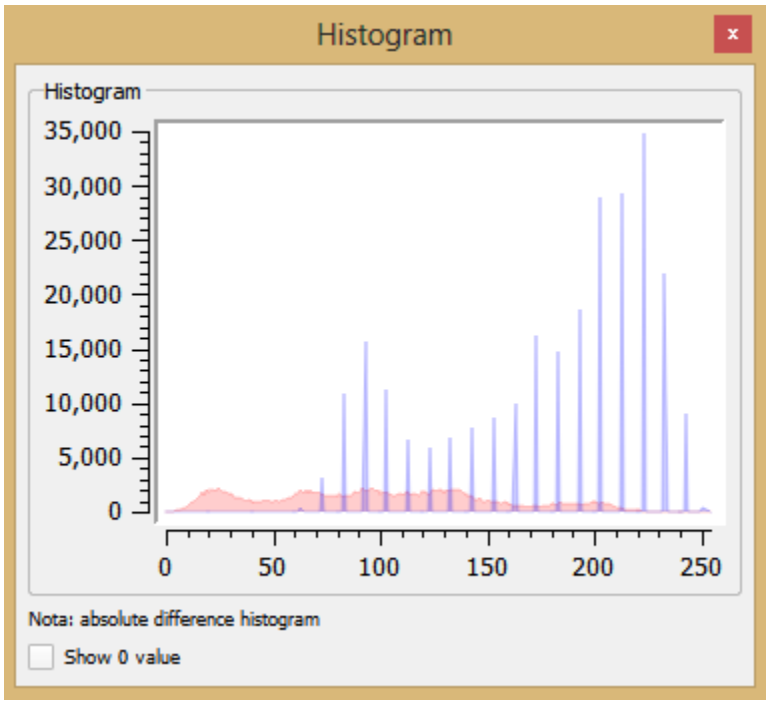


Figure 4.14. Histogram for Lena in second experiment.

Fourth Image

Figure 4.15 is Baboon before and after embedding the secret text using the proposed technique

Figure 4.15. Baboon before and after second experiment character using the proposed technique.

Table 4.8 shows the results after storing 65536 characters inside Baboon using the proposed QR based technique.

	Proposed method	LSB method
Mean Error	0.69313	0.98898
Standard Deviation	0.81344	0.43602
RMSE	1.06869	1.08083
PSNR	47.553	47.455

Error Number	19393	111426

Table 4.8. Statistics for Baboon second experiment after embedding using proposed technique

Figure 4.16 is the histogram of Baboon after embedding the secret message of the first experiment using the proposed technique.

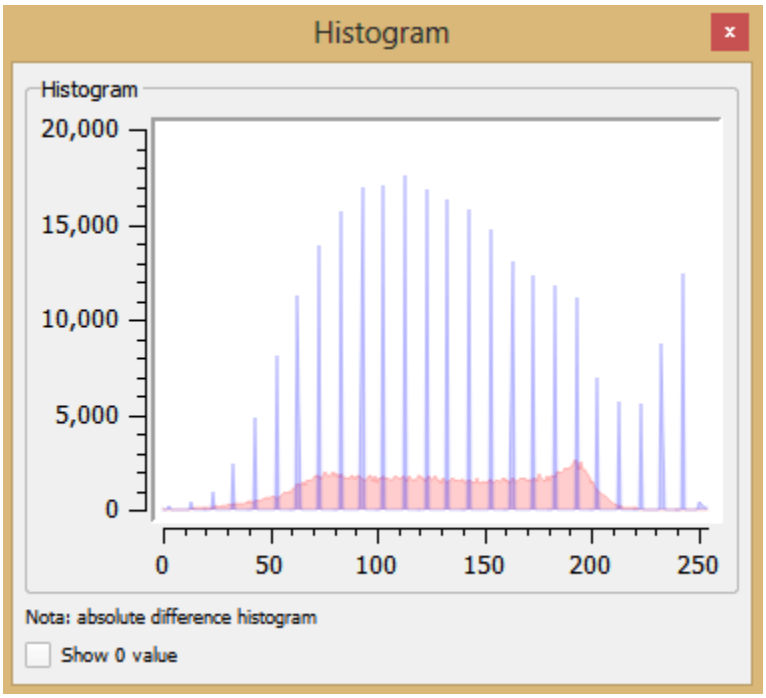


Figure 4.16. Histogram for Baboon in second experiment

After studying the results of the second experiments when hiding 65536 characters, we note that proposed method and the LSB method have almost equal effect on the stego image after embedding.

Third Experiment

First Image

Figure 4.17 is Sea before and after embedding 131072 characters of secret text using the proposed technique

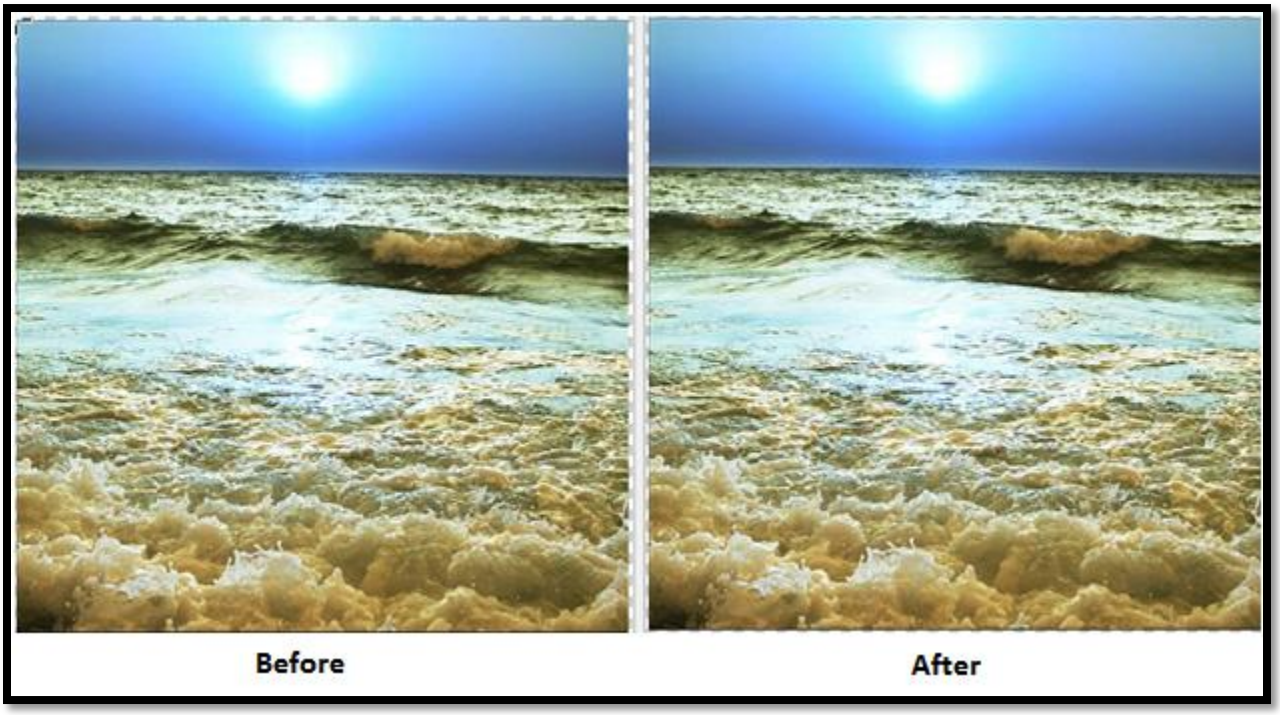


Figure 4.17. Sea before and after third experiment using the proposed technique

Table 4.9 shows the results after storing 131072 characters inside Sea using the proposed QR based technique.

	Proposed method	LSB method
Mean Error	1.64470	1.97755
Standard Deviation	1.19799	0.50492
RMSE	2.03475	2.04099
PSNR	41.960	41.933

Error Number	45915	221817

Table 4.9. Statistics for Sea third experiment after embedding using proposed technique

Figure 4.18 is the histogram Sea after embedding the secret message of the first experiment using the proposed technique.

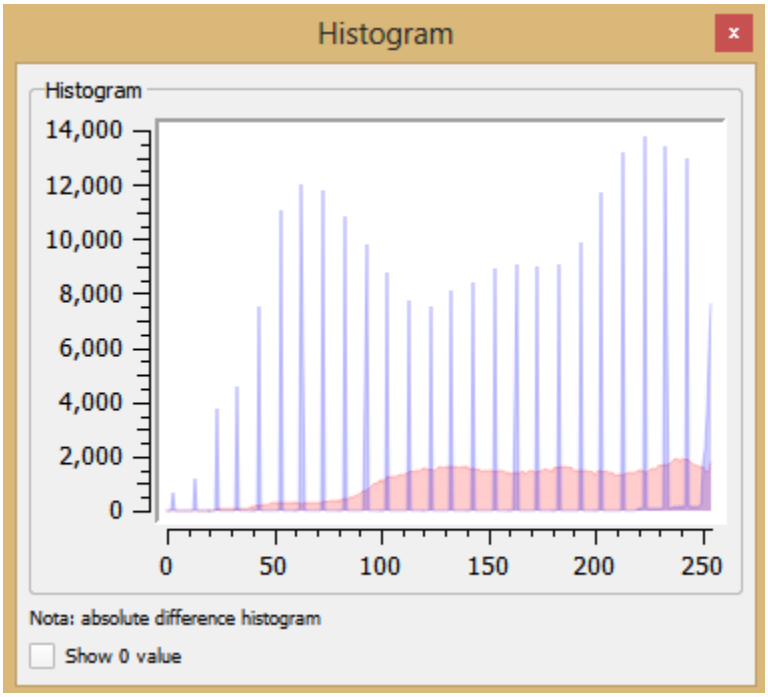


Figure 4.18. Histogram for Sea in third experiment

Second Image

Figure 4.19 shows Peppers before and after embedding 131072 characters of secret text using the proposed technique

Figure 4.19. Peppers before and after third experiment characters using the proposed technique

Table 4.10 contains the results after storing 131072 characters inside Peppers using the proposed QR based technique.

	Proposed method	LSB method
Mean Error	1.64699	1.94503
Standard Deviation	1.19996	0.50476
RMSE	2.03776	2.00946
PSNR	41.947	42.069
Error Number	45871	219974

Table 4.10. Statistics for Peppers third experiment after embedding using proposed technique

Figure 4.20 is the histogram of Peppers after embedding the secret message of the first experiment using the proposed technique.

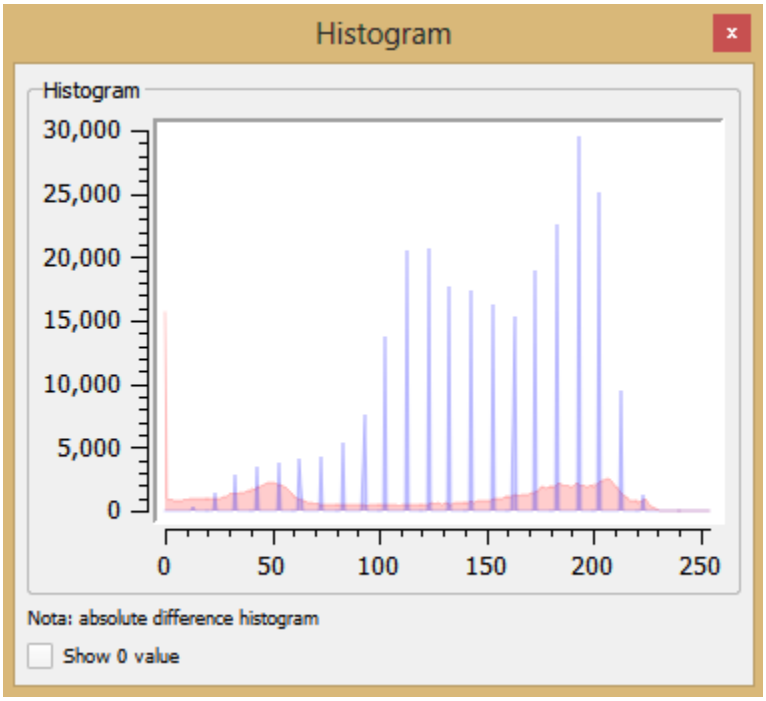


Figure 4.20. Histogram for Peppers in third experiment

Third Image

Figure 4.21 shows Lena before and after embedding 131072 characters of secret text using the proposed technique.

Figure 4.21. Lena before and after third experiment characters using the proposed technique

Table 4.11 shows the results after storing 131072 characters inside Lena using the proposed QR based technique.

	Proposed method	LSB method
Mean Error	1.64180	1.96491
Standard Deviation	1.19541	0.50487
RMSE	2.03089	2.02874
PSNR	41.977	41.986
Error Number	45878	221088

Table 4.11. Statistics for Lena third experiment after embedding using LSB technique

Figure 4.22 is the histogram of Lena after embedding the secret message of the first experiment using the proposed technique.

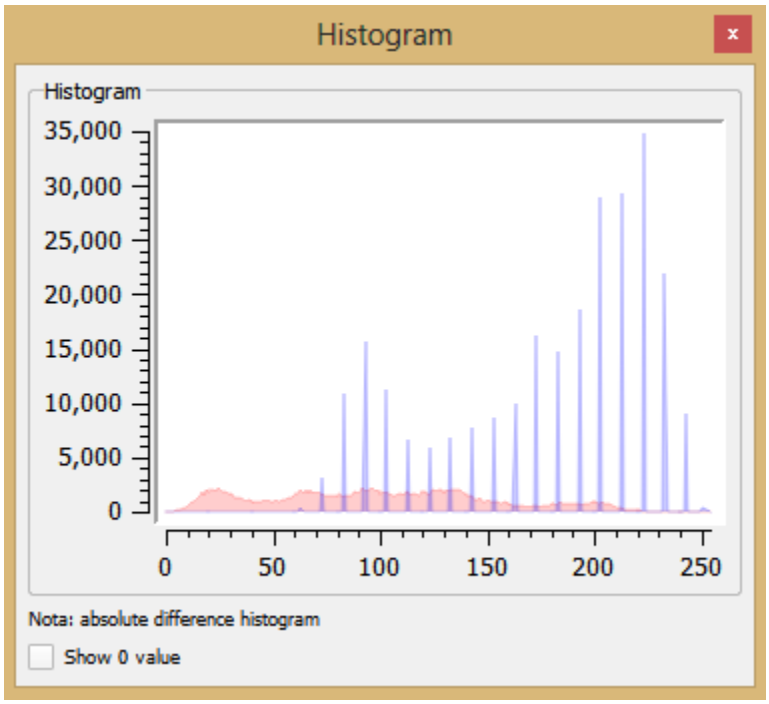


Figure 4.22. Histogram for Lena in third experiment.

Fourth Image

Figure 4.23 is Baboon before and after embedding 131072 characters of secret text using the proposed technique

Figure 4.23. Baboon before and after third experiment character using the proposed technique.

Table 4.12 is the results after storing 131072 characters inside Baboon using the proposed QR based technique.

	Proposed method	LSB method
Mean Error	1.64152	1.96383
Standard Deviation	1.19686	0.50487
RMSE	2.03152	2.02769
PSNR	41.974	41.990
Error Number	45916	221448

Table 4.12. Statistics for Baboon third experiment after embedding using proposed technique

Figure 4.24 shows histogram for Baboon after embedding the secret message of the first experiment using the proposed technique.

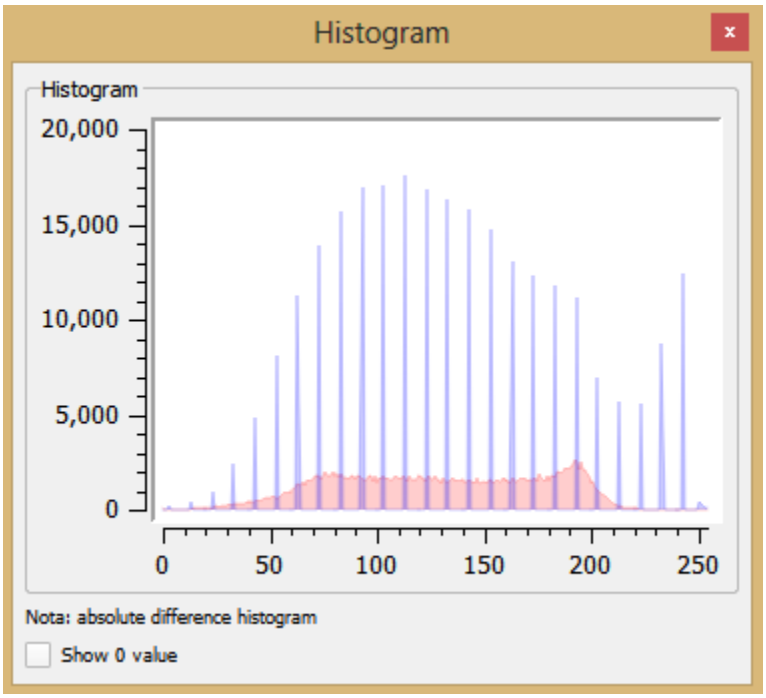


Figure 4.24. Histogram for Baboon in third experiment

Based on the above results, the following table show the best results achieved for each technique in the three experiments.

First Experiment:

In the first experiment, the LSB method was better than the proposed method, however, the performance of the proposed method was best in peppers

	Proposed Method	LSB Method
Mean Error	0.01177	0.00135
Standard Deviation	0.17151	0.03884
RMSE	0.17191	0.03884
PSNR	63.424	76.345
Error Number	1605	334

Table 4.13. Peppers in first experiment

Second Experiment:

In the second experiment, the proposed method was better than the LSB, the performance of the proposed method was best in peppers.

	Proposed method	LSB method
Mean Error	0.69010	0.97805
Standard Deviation	0.81041	0.43442
RMSE	1.06442	1.07019
PSNR	47.588	47.541
Error Number	19390	110618

Table 4.14 . Peppers in second experiment

Third Experiment:

In the first experiment, the proposed method was better than the LSB mehod, the performance of the proposed method was best in baboon.

	Proposed method	LSB method
Mean Error	1.64152	1.96383
Standard Deviation	1.19686	0.50487
RMSE	2.03152	2.02769
PSNR	41.974	41.990
Error Number	45916	221448

Table 4.15. Baboon in third experiment

Chapter Five

Chapter 5 Conclusion

Conclusion

The proposed technique to hide secret text in QR code, before hiding the QR inside the 4 channels PNG cover image, the QR is saved as black and white image with 1 bit per pixel and is stored in array of bytes to be encrypted using AES before embedding it inside the cover image.

After analyzing the results of the experiments of the proposed technique and comparing the results with the LSB insertion it became obvious that the proposed technique provide superior results than LSB when using large data, despite the fact that the impact is greater when using the first and third bits than when using the least significant bit alone, the proposed technique effect on the picture was almost the same and even better in some images than LSB when hiding relatively large data.

Future Work

This study can provide the base for several future researches, the following points are suggested to further study the performance of the proposed technique:

1. Combining QR codes with compression to enhance the payload.
2. Study the effect when using different types of images.
3. Use QR Codes to store audio data and compare the capacity with known techniques.

References

- Anderson, R. &. (1998). On the limits of steganography. *IEEE Journal of Selected Areas in Communications, 16*(4), 474-481.
- Arts, D. (2001). Digital Steganography: Hiding Data within Data. *Internet Computing, IEEE, 5*(3), 75-80.
- Currie, D. &. (1996). Surmounting the effects of lossy compression on steganography. *19th National Information Systems Security Conference.*, (pp. 194-201).
- Dey, S. (2012). SD-EQR: A New Technique To Use QR CodesTM. *International Journal of Information Technology & Computer Science (IJITCS)*, 29-35.
- Jantan, M. A. (2008). A New Steganography Approach for Image Encryption Exchange by using LSB insertion. *International Journal of Computer Science and Network Security, 8*(6), 247-254.
- Motameni H., N. M. (2007). Labeling technique in steganography. *Proceedings of world academy of science, engineering and technology*, (pp. 349-354).
- Nath, J. (2012). Advanced Steganography Algorithm Using Randomized Intermediate QR Host Embedded With Any Encrypted Secret Message: ASA_QR Algorithm. *6*, 59-67.
- Po Yunch Chen, &. H. (2006). A DWT Based Approach for Image Steganography. *International journal of Applied Science and Engineering, 4*(3), 275-290.
- Rijmen, J. D. (2002). The Design of Rijndael, AES - The Advanced Encryption Standard. 238.
- Sahu, S. K. (2013). Encryption in QR Code Using Stegnography. *International Journal of Engineering Research and Applications*, (pp. 1783-1741).

- Suppat Rungraungsilp, M. K. (2012). Data Hiding Technique for QR Code Based on Watermark by compare DCT with DFT Domain. *International Conference on Computer and Communication Technologies (ICCCT'2012)*, 144-148.
- Tandon, A. (2013). QR Code based secure OTP distribution. *International Journal of Engineering and Technology (IJET), 5*(3), 2502-2505.
- Morkel T., Eloff J.H.P., & Olivier M.S., (2006)"An Overview of Image Steganography", University of Pretoria.
- Kefa Rabah. (2004), Steganography-The Art of Hiding Data, Department of Physics, Eastern Mediterranean University, Gazimagusa, North Cyprus, via Mersin 10, Turkey.
- 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.
- 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.
- Luis von Ahn and Nicholas J. Hopper. Public-key steganography. In Advances in Cryptology Proceedings of Eurocrypt 04, pages 323-341. Springer-Verlag, 2004.

- Asif, A. A., Shaikh, A., Manza, R. R., & Ramteke, R. J. (2010). Conversion of Bitmap Text Images for Data Hiding. Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference, 1 – 4
- 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.
- Zou, D., & Shi, Y. Q. (2005). Formatted Text Document Data Hiding Robust to Printing, Copying and Scanning. Institute of Electrical and Electronics Engineers (IEEE), 4971 – 4974.

Appendix: Source 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;
using System.Runtime.InteropServices;

namespace Image_Steganography
{
    public partial class Form1 : Form
    {

        Image CoverImage;
        BitmapIP CoverImageIP;
        Bitmap CoverImageBitmap;

        Image SecretImage;


        byte[] ImageBytes;

        public Form1()
        {
            InitializeComponent();
            SecretText.Visible = true;
            SecretImageBox.Visible = false;
            GenerateQR.Visible = true;
            EncryptQR.Visible = false;
            EmbedQR.Visible = false;
            SecretKey.Enabled = true;
            ExtractQR.Visible = false;
            DecryptQR.Visible = false;
        }

        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;
            }



            return true;
        }

        private void Hide_Click(object sender, EventArgs e) //this generates a QR
        {
            if (!ValidateInputs())
                return;
            string SecretText_ = new string(SecretText.Text.ToCharArray());

            int textLength = SecretText_.Length;
            int remainder = textLength % 600;
            int _noOfSegments = (int)(textLength / 600);
            string[] chuncks = new string[_noOfSegments];

            //Obaidah

            //Obaidah
            int j;
            Bitmap QR_Image = null;


            for (j = 0; j < _noOfSegments; j++)
            {
                chuncks[j] = SecretText_.Substring(j * 600, 600);

                QRCodeGenerator qrGenerator = new QRCodeGenerator();
                QRCodeGenerator.QRCode qrCode = qrGenerator.CreateQrCode(chuncks[j],
QRCodeGenerator.ECCLevel.L);
                QR_Image = qrCode.GetGraphic(3); //tested with 20
                QR_Image = BitmapTo1Bpp(QR_Image);

                QR_Image.Save(@"D:\Stego\" + j.ToString() + ".gif", ImageFormat.Gif);

            }


            DirectoryInfo directory = new DirectoryInfo("D:\\Stego\\");
            if (directory != null)
```

```
    {
        FileInfo[] files = directory.GetFiles();
        CombineImages(files);
    }


    //SecretImageBox.Image = QR_Image;
    SecretText.Visible = false;
    SecretImageBox.Visible = true;
    GenerateQR.Visible = false;
    EncryptQR.Visible = true;
    //ImageConverter converter = new ImageConverter();
    //return (byte[])converter.ConvertTo(QR_Image, typeof(byte[]));
}

private void CombineImages(FileInfo[] files)
{
    //change the location to store the final image.
    string finalImage = @"D:\\QR.bmp";
    List<int> imageHeights = new List<int>();

    int nIndex = 0;
    int width = 0;
    foreach (FileInfo file in files)
    {
        Image img = Image.FromFile(file.FullName);
        imageHeights.Add(img.Height);
        width += img.Width;
        img.Dispose();
    }
    imageHeights.Sort();
    int height = imageHeights[imageHeights.Count - 1];
    Bitmap img3 = new Bitmap(width, height);
    Graphics g = Graphics.FromImage(img3);
    g.Clear(SystemColors.AppWorkspace);
    foreach (FileInfo file in files)
    {
        Image img = Image.FromFile(file.FullName);
        if (nIndex == 0)
        {
            g.DrawImage(img, new Point(0, 0));
            nIndex++;
            width = img.Width;
        }
        else
        {
            g.DrawImage(img, new Point(width, 0));
            width += img.Width;
        }
        img.Dispose();
    }
    g.Dispose();
    img3 = BitmapTo1Bpp(img3);
    //img3.Save(finalImage, System.Drawing.Imaging.ImageFormat.Bmp);
    img3.Save(finalImage);
    img3.Dispose();
    SecretImageBox.Image = Image.FromFile(finalImage);
}

public static Bitmap BitmapTo1Bpp(Bitmap img)
{
    int w = img.Width;
    int h = img.Height;
```

```
            Bitmap bmp = new Bitmap(w, h, PixelFormat.Format1bppIndexed);
            BitmapData data = bmp.LockBits(new Rectangle(0, 0, w, h),
ImageLockMode.ReadWrite, PixelFormat.Format1bppIndexed);
            byte[] scan = new byte[(w + 7) / 8];
            for (int y = 0; y < h; y++)
            {
                for (int x = 0; x < w; x++)
                {
                    if (x % 8 == 0) scan[x / 8] = 0;
                    Color c = img.GetPixel(x, y);
                    if (c.GetBrightness() >= 0.5) scan[x / 8] |= (byte)(0x80 >> (x % 8));
                }
                Marshal.Copy(scan, 0, (IntPtr)((long)data.Scan0 + data.Stride * y),
scan.Length);
            }
            bmp.UnlockBits(data);
            return bmp;
        }

        public int GetLength()
        {
            /*int iWidth = CoverImageIP.GetBitmap().Width;
            int iHeight = CoverImageIP.GetBitmap().Height;
            Color[,] ImageArray = CoverImageIP.GetImage2DArray();*/
            int Length;
            Color tmpColor = new Color();
            tmpColor = CoverImageBitmap.GetPixel(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()
        {
            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();
            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)));
                    w++;
                }
```

```
            if (completed)
                break;
        }
        ImageBytes = GetBytesFromString(sHiddenData.ToString());
        MessageBox.Show("Encrypted QR Extracted Successfuly!");
    }

    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)
    {

        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;




        iFirstBit = iRedBits[7];
        iSecondBit = iRedBits[5];


        iThirdBit = iGreenBits[7];
        iFourthBit = iGreenBits[5];

        iFifthBit = iBlueBits[7];
        iSixthBit = iBlueBits[5];

        iSeventhBit = iAlphaBits[7];
        iEightethBit = iAlphaBits[5];

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

```
        public void SetLength(int TxtLength)
        {

            Color tmpColor = new Color();
            tmpColor = CoverImageBitmap.GetPixel(0,0);
            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);
        }


        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);
            //MessageBox.Show(bRed.ToString());
            //test here obaidah
            iRedBits[7] = iFirstBit;
            iRedBits[5] = iSecondBit;

            iGreenBits[7] = iThirdBit;
            iGreenBits[5] = iFourthBit;
```

```
            iBlueBits[7] = iFifthBit;
            iBlueBits[5] = iSixthBit;

            iAlphaBits[7] = iSeventhBit;
            iAlphaBits[5] = 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 void Hide_Message()
        {
            WSImages myImage = new WSImages();
            SetLength(ImageBytes.Length);
            Color tmpColor;
            int iHeight = CoverImageBitmap.Height;
            int iWidth = CoverImageBitmap.Width;
            bool completed = false;
            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);
                    tmpColor = ChangeColorValue(tmpColor, (ImageBits[0]), (ImageBits[1]),
                        (ImageBits[2]), (ImageBits[3]), (ImageBits[4]),
                        (ImageBits[5]), (ImageBits[6]), (ImageBits[7]));
                    CoverImageBitmap.SetPixel(i, j, tmpColor);
                }
                if (completed)
                    break;
            }
            CoverImageBitmap.Save("d:\\Stegno.png");
            SecretImageBox.Image = CoverImageBitmap;

        }
```

```
        private void EncryptQR_Click(object sender, EventArgs e)
        {

            WSImages myImage = new WSImages();
            ImageBytes = myImage.GetImage("d:\\QR.bmp");
            if (SecretKey.Text.Length != 0)

                ImageBytes = RijndaelHelper.EncryptBytes(ImageBytes, SecretKey.Text,
"key");


            MemoryStream memStream = new MemoryStream(ImageBytes);
            // Convert memory stream to a Bitmap
            try
            {
                File.WriteAllBytes(@"D:\Encrypted QR.dmp", ImageBytes);
                MessageBox.Show(@"Encrypted Successfuly!, file stored on D:\");
                EncryptQR.Visible = false;
                EmbedQR.Visible = true;
                SecretKey.Enabled = false;
            }
            catch
            {
                MessageBox.Show("Unable to Encrypt QR!");
            }
            // save image returned to local disk(requested server/client machine)
        }

        private void EmbedQR_Click(object sender, EventArgs e)
        {
            Hide_Message();
            EmbedQR.Visible = false;
        }

        private void Embedding_CheckedChanged(object sender, EventArgs e)
        {
            SecretText.Visible = true;
            SecretImageBox.Visible = false;
            GenerateQR.Visible = true;
            EncryptQR.Visible = false;
            EmbedQR.Visible = false;
            SecretKey.Enabled = true;
            ExtractQR.Visible = false;
            DecryptQR.Visible = false;
        }

        private void Extracting_CheckedChanged(object sender, EventArgs e)
        {
            SecretText.Visible = false;
            SecretImageBox.Visible = false;
            GenerateQR.Visible = false;
            EncryptQR.Visible = false;
            EmbedQR.Visible = false;
            SecretKey.Enabled = true;
            ExtractQR.Visible = true;
        }

        private void ExtractQR_Click(object sender, EventArgs e)
        {
            Extract_Message();
            ExtractQR.Visible = false;
            DecryptQR.Visible = true;
```

```
        }

        private void DecryptQR_Click(object sender, EventArgs e)
        {

            SecretImageBox.Visible = true;
            byte [] ImageBytes1 = RijndaelHelper.DecryptBytes(ImageBytes, SecretKey.Text,
"key");
            if (ImageBytes1 == null)
                return;
            ImageBytes = ImageBytes1;
            WSImages myImage = new WSImages();


            MemoryStream memStream = new MemoryStream(ImageBytes);
            // Convert memory stream to a Bitmap
            try
            {
                Bitmap bm = new Bitmap(memStream);
                bm = BitmapTo1Bpp(bm);
                bm.Save("d:\\Extracted QR.bmp");
                MessageBox.Show("Extracted Successfuly!");
                SecretImageBox.Image = bm;
                DecryptQR.Visible = false;
            }
            catch
            {
                MessageBox.Show("Unable to extract image, please check key");
            }
        }

        private void SecretText_TextChanged(object sender, EventArgs e)
        {
            textlength.Text = SecretText.TextLength.ToString();
        }
    }
}
```