A Project Report

on

DATA PROTECTION USING CRYPTOGRAPHY AND STEGNOGRAPHY

Submitted in partial fulfillment of the
requirement for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Mr. Peerzada Hamid Ahmad
Assistant Professor

Submitted By

Rahul Kumar 20SCSE1290084
Yugal Joshi 20SCSE1290087

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
MAY, 2022

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the project, entitled **"DATA PROTECTION USING CRYPTOGRAPHY AND STEGNOGRAPHY"** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of month, Year to Month and Year, under the supervision of **Mr. Peerzada Hamid Ahmad , Assistant Professor**, Department of Computer Science and Engineering, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the project/ has not been submitted by us for the award of any other degree of this or any other places.

Rahul Kumar – 20SCSE1290084

Yugal Joshi – 20SCSE1290087

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

(Mr. Peerzada Hamid Ahmad, Assistant Professor**)**

CERTIFICATE

The Final Project Viva-Voce examination of **RAHUL KUMAR: 20SCSE1290084, YUGAL JOSHI: 20SCSE1290087** has been held on _____ and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING.**

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: 9, May, 2022

Place: Greater Noida

Abstract

Using cryptography and steganography techniques to secure data through encryption and decryption. Providing security to personal contents, messages, or digital images using steganography (the practice of concealing a message within another message or a physical object) has become difficult due to recent developments in Stego analysis (Stego analysis is the study of detecting messages hidden using steganography). The existence of concealed information in carrier files can be easily revealed via Stego analysis. This project introduces a revolutionary steganographic information exchange for two private parties. Both steganographic and cryptographic techniques are used for the approach presented in this project. We apply image steganography in steganography to hide the data. We also employ the Mutual Authentication procedure to provide all cryptographic services, such as access control, confidentiality, integrity, and authentication. We'll be able to keep the data more securely this way. We use a cryptographic technique to secure the data, and we use Steganography to hide the data in a picture. As a result, any other person on the network is unable to access the network's data. The message can only be retrieved from the data by the sender and receiver.

Table of Contents

List of Figures

CHAPTER-1

Introduction

INTRODUCTION

Digital communication witnesses a noticeable and continuous development in many applications in the Internet. Hence, secure communication sessions must be provided. The security of data transmitted across a global network has turned into a key factor on the network performance measures. So, the confidentiality and the integrity of data are needed to prevent eavesdroppers from accessing and using transmitted data. Steganography and Cryptography are two important techniques that are used to provide network security. The aim of this project is to develop a new approach to hiding a secret information in an image, by taking advantage of benefits of combining cryptography and steganography.

1.1 Cryptography

Cryptography is one of the traditional methods used to guarantee the privacy of communication between parties. This method is the art of secret writing, which is used to encrypt the plaintext with a key into ciphertext to be transferred between parties on an insecure channel. Using a valid key, the ciphertext can be decrypted to the original plaintext. Without the knowledge of the key, nobody can retrieve the plaintext. Cryptography plays an essential role in many factors required for secure communication across an insecure channel, like confidentiality, privacy, nonrepudiation, key exchange, and authentication.

1.2 Symmetric / Secret Key Cryptography

The technique of Secret key encryption can also be known as the symmetric-key, shared key, single-key, and eventually private-key encryption. The technique of private key uses for all sides encryption and decryption of secret data. The original information or plaintext is encrypted with a key by the sender side also the similarly key is used by the receiver to decrypt a message to obtain the plaintext. the key will be known only by a people who are authorized to the encryption/decryption. However, the technique affords the good security for transmission but there is a difficulty with the distribution of the key. If one stole

or explore the key he can get whole data without any difficulty. An example of Symmetric-Key is DES Algorithm

1.3 Asymmetric / Public Key Cryptography

We can call this technique as asymmetric cryptosystem or public key cryptosystem, this technique uses two keys which are mathematically associated, use separately for encrypting and decrypting the information. In this technique, when we use the private key, there are no possibilities to obtain the data or simply discover the other key. The key used for encryption is stored public therefore it's called public key, and the decryption key is stored secret and called private key. An example of Asymmetric-Key Algorithm is RSA.

1.4 Steganography

It can be defined as the science of hiding and communicating data through apparently reliable carriers in attempt to hide the existence of the data. So, there is no knowledge of the existence of the message in the first place. If a person views the cover which the information is hidden inside, he or she will have no clue that there is any covering data, in this way the individual won't endeavour to decode the data. The secret information can be inserted into the cover media by the stego system encoder with using certain algorithm. A secret message can be plaintext, an image, ciphertext, or anything which can be represented in form of a bitstream. after the secret data is embedded in the cover object, the cover object will be called as a stego object also the stego object sends to the receiver by selecting the suitable channel, where decoder system is used with the same stego method for obtaining original information as the sender would like to transfer.

1.5 Types of Steganography

There are various types of steganography.

A. Text Files The technique of embedding secret data inside a text is identified as text stego. Text steganography needs a low memory because this type of file can only store text files. It affords fast transfer or communication of files from a sender to receiver.

B. Image Files It is the procedure in which we embed the information inside the pixels of image. So, that the attackers cannot observe any change in the cover image. LSB approach is a common image steganography algorithm.

C. Audio Files It is the process in which we hide the information inside an audio. There are many approaches to hide secret information in an audio file for examples Phase Coding, LSB.

D. Video Files It is the process of hiding some secret data inside the frames of a video.

1.6 Steganography versus Cryptography

Steganography and cryptography are used for the purpose of data transmission over an insecure network without the data being exposed to any unauthorized persons. Steganography embeds the data in a cover image while cryptography encrypts the data. The advantage of Steganography is that, the look of the file isn't changed and this it will not raise any doubt for the attacker to suspect that there may be some data hidden unlike cryptography that encrypts the data and sends it to network.

1.7 Benefits of Steganography and Cryptography

It is noted that steganography and cryptography alone is insufficient for the security of information, therefore if we combine these systems, we can generate more reliable and strong approach. The combination of these two strategies will improve the security of the information. This combined will fulfill the prerequisites, for example, memory space, security, and strength for important information transmission across an open channel. Also, it will be a powerful mechanism which enables people to communicate without interferes of 6 eavesdroppers even knowing there is a style of communication in the first place.

1.8 Applications of Steganography

(i) Secret Communication: Steganography does not advertise secret communication and therefore avoids scrutiny of the sender message. A trade secret, blueprint, or other sensitive information can be transmitted without alerting potential attackers.

(ii) Feature Tagging: Elements can be embedded inside an image, such as the names of the individuals in a photo or location in a map. Copying the stego image also copies all of the embedded features and only parties who possess the decode stego key will be able to extract and view the features.

(iii) Copyright Protection: Copy protection mechanisms that prevent the data, usually digital data from being copied. The insertion and analysis of water marks to protect copyrighted material is responsible for the percent rise of interest digital steganography and data embedding.

2. Digital Image Steganography

Digital Image Steganography system is a stand-alone application that combine steganography and encryption to enhance the confidentiality of intended message. The user's intended message is first encrypted to create unintelligible cipher text will be hidden within an image file in such a way as to minimize the perceived loss in quality. The recipient of the image is able to retrieve the hidden message back from the image with this system.

PROBLEM STATEMENT

The purpose of this project is to provide the correct data with security to the users. For some of the users the data might be lost during the transmission process in the network and for some, the data might be changed by the unauthorized person in the network and there are some other security problems in the network. Our application will give you more Security to the data present in the network and there will be able to reduce the loss of data in the network which will be transmitted from the sender to the receiver using the latest technologies. Only the Authorized persons i.e., who are using our application will be 8 there in the Network. The proposed algorithm is to hide the audio data effectively in an image

without any suspicion of the data being hidden in the image. It is to work against the attacks by using a distinct new image that isn’t possible to compare.

The aim of the project is to hide the data in an image using steganography and ensure that the quality of concealing data must not be lost.

We used a method for hiding the data in a distinct image file in order to securely send over the network without any suspicion the data being hidden. This algorithm, though requires a distinct image which we can use as a carrier and hide the data which is well within the limits of the threshold that the image can hide, that will secure the data

CHAPTER-2

Literature Survey

As we said the significance of network security is increased day by day as the size of data being transferred across the Internet. This issue pushes the researchers to do many studies to increase the ability to solve security issues. A solution for this issue is using the advantage of cryptography and steganography combined in one system. Many studies propose methods to combine cryptography with steganography systems in one system. This Project has been implemented on the basis of the requirements of security i.e., authentication, confidentiality, and robustness.

There has been a continuous rise in the number of data security threats in the recent past and it has become a matter of concern for the security experts. Cryptography and steganography are the best techniques to nullify this threat. The researchers today are proposing a blended approach of both techniques because a higher level of security is achieved when both techniques are used together.

In proposed an encrypting technique by combining cryptography and steganography techniques to hide the data. In cryptography process, we proposed an effective technique for data encryption using one's complement method. It used an Asymmetric key method where both sender and receiver share the Secret key for encryption and decryption. In steganography part, we used the LSB method that is used and mostly preferred.

We present a method based on combining both the strong encrypting algorithm and steganographic technique to make the communication of confidential information safe, secure and extremely hard to decode. An encryption technique is employed for encrypting a secret message into a Cipher text using the Senders Private Key and receiver public key. The Cipher Text is finally embedded in a suitable cover 11 image and transferred securely to deliver the secret information. They utilized a least significant bit method to accomplish the digital image steganography.

At the receiver's side, the secret data is retrieved through the decoding process. Thus, a three-level security has been rendered for them a secret message to be transferred.

Image Steganography

Image Steganography deals with the hiding of data within the image, data can be any file, such as an image, audio, text or another file. We have to wrap up the data using an image. We have chosen to bind data in an image. This kind of embedding an audio in an image helps to authenticate the sender, verify whether valid user is receiving the data or not and to find whether a third-party attacker is present in the channel of communication or not. Since, image is used as a cover file, we have to make sure that the image must be accountable for the data that is being embedded. Hence a 24-bit image format proved to be the best solution for hiding the data, since it holds a large memory space and convenient to hide a considerable amount of data. Furthermore, the threshold sure of the image must be calculated for the given image size which will be explained in the later parts.

LSB Positioning Method This method is the simplest method of hiding data within in the given image. We utilize the LSB bits of the pixels within the given image. When converting the image to digital format, we usually choose between three different ways of representing colors: 23

- 24-bit color: every pixel can have one in 2^24 colors, and these are represented as different quantities of three basic colors: red(R), green(G), blue(b) given by 8 bits (256) each.

- 8-bit color: every pixel can have one in 256 (2^28) colors, chosen from a palette, or a table of colors.

- 8-bit gray-scale: every pixel can have one in 256 (2^28) shades of gray.

LSB insertion modifies the LSBs of each color in 24-bit images, or the LSBs of the 8-bit value for 8-bit images. The most basic of LSBs insertion for 24-bit pictures inserts 3 bits/pixel

For image steganography we are using Spatial methods. In spatial method, the most common method used is LSB substitution method. Least significant bit (LSB) method is a common, simple approach to embedding information in a cover file. In steganography, LSB substitution method is used. I.e., since every image has three components (RGB). This pixel information is stored in encoded format in one byte.

The first bits containing this information for every pixel can be modified to store the hidden text. For this, the preliminary condition is that the text to be stored has to be smaller or of equal size to the image used to hide the text. LSB based method is a spatial domain method. But this is vulnerable to cropping and noise. In this method, the MSB (most significant bits) of the message image to be hidden are stored in the LSB (least significant bits) of the image used as the cover image.

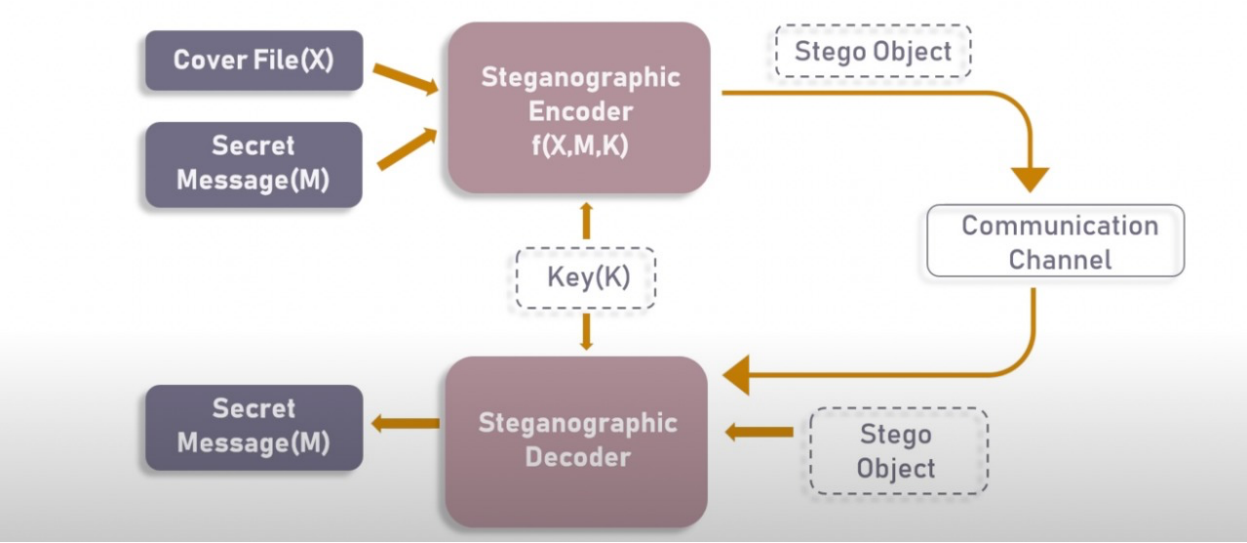
The Human visual system (HVS) cannot detect changes in the colour or intensity of a pixel when the LSB bit is modified. This is psycho-visual redundancy since this can be 24 used as an advantage to store information in these bits and yet notice no major difference in the image.

Chapter – 3

Project Design

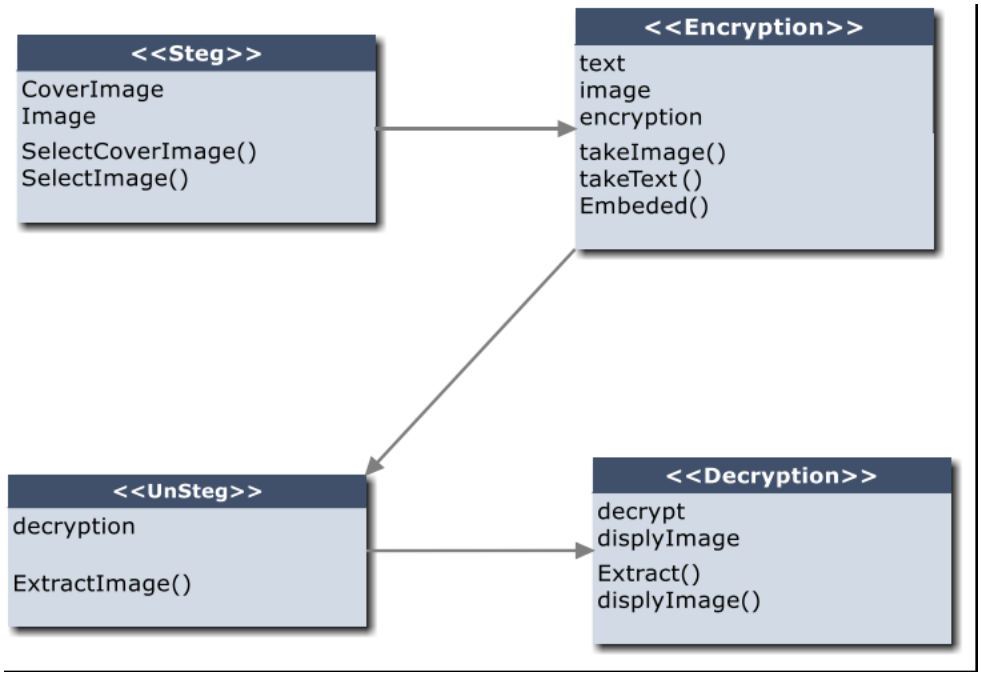
Use Case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.



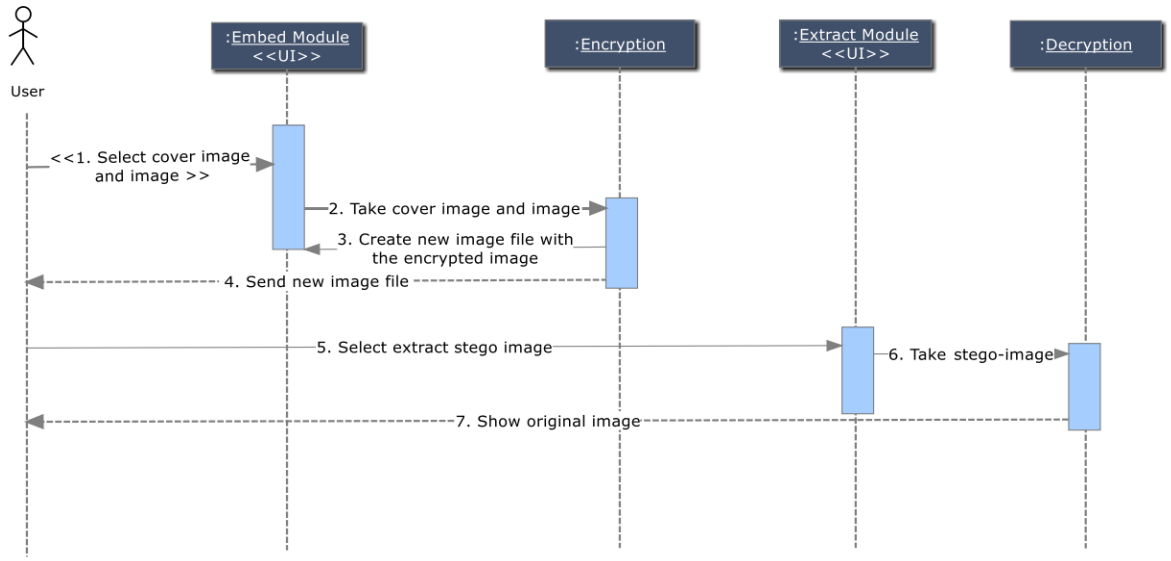
Class Diagram

A class diagram in the Unified Modelling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application. Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.



Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios. A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



Chapter – 4

Module Description

Module 1 DCT(Discrete Cosine Transform)

Discrete Cosine Transform is used in lossy image compression because it has very strong energy compaction, i.e., its large amount of information is stored in very low

frequency component of a signal and rest other frequency having very small data which can be stored by using very less number of bits (usually, at most 2 or 3 bit).
To perform DCT Transformation on an image, first we have to fetch image file information (pixel value in term of integer having range 0 – 255) which we divide in block of 8 X 8 matrix and then we apply discrete cosine transform on that block of data.
After applying discrete cosine transform, we will see that its more than 90% data will be in lower frequency component. For simplicity, we took a matrix of size 8 X 8 having all value as 255 (considering image to be completely white) and we are going to perform 2-D discrete cosine transform on that to observe the output.

DCT plays a very important role in digital signal processing. By using the DCT, the signals can be compressed. DCT can be used in electrocardiography for the compression of ECG signals. DCT2 provides a better compression ratio than DCT. The DCT is widely implemented in digital signal processors (DSP), as well as digital signal processing software. Many companies have developed DSPs based on DCT technology. DCTs are widely used for applications such as encoding, decoding, video, audio, multiplexing, control signals, signalling, and analog-to-digital conversion. DCTs are also commonly used for high-definition television (HDTV) encoder/decoder chips.

DCT Encoding

The general equation for a 1D (*N* data items) DCT is defined by the following equation:

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

and the corresponding ***inverse*** 1D DCT transform is simple $F^{-1}(u)$, i.e.:

where

$$\Lambda(i) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

The general equation for a 2D (*N* by *M* image) DCT is defined by the following equation:

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

and the corresponding ***inverse*** 2D DCT transform is simple $F^{-1}(u,v)$, i.e.:

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

Module 2 Quantization

DCT-based image compression relies on two techniques to reduce the data required to represent the image. The first is quantization of the image's DCT coefficients. Quantization is the process of reducing the number of possible values of a quantity, thereby reducing the number of bits needed to represent it. Entropy coding is a technique for representing the quantized data as compactly as possible. We will develop functions to quantize images and to calculate the level of compression provided by different degrees of quantization. We will not implement the entropy coding required to create a compressed image file.

Module 3 Dequantization

De-quantization is the task of reversing the quantization effect and recovering the original multi-chromatic level image. Existing techniques achieve de-quantization by imposing suitable constraints on the ideal image in order to make the recovery problem feasible since it is otherwise ill-posed. Our goal in this work is to develop a de-quantization mechanism through a rigorous mathematical analysis which is based on the classical statistical estimation theory. In this effort we incorporate generative modeling of the ideal image as a suitable prior information. The resulting technique is simple and capable of de-quantizing successfully images that have experienced severe quantization effects. Interestingly, our method can recover images even if the quantization process is not exactly known and contains unknown parameters.

Module 4 Inverse Discrete Cosine Transform

The inverse discrete cosine transform reconstructs a sequence from its discrete cosine transform (DCT) coefficients. The idct function is the inverse of the dct function.

The DCT has four standard variants. For a transformed signal y of length N, and with $\delta_{k\ell}$ the Kronecker delta, the inverses are defined by:

- Inverse of DCT-1:

$$x(n)=\sqrt{2N-1_N}\sum_{k=1}{}^{y(k)}{}_1\sqrt{1+\delta_{k1}+\delta_{kN}}{}_1\sqrt{1+\delta_{n1}+\delta_{nN}}\cos\left({}_{\pi N-1}{}^{(k-1)(n-1)}\right)$$

- Inverse of DCT-2:

$$x(n)=\sqrt{2N_N}\sum_{k=1}{}^{y(k)}{}_1\sqrt{1+\delta_{k1}}\cos\left({}_{\pi 2N}{}^{(k-1)(2n-1)}\right)$$

- Inverse of DCT-3:

$$x(n)=\sqrt{2N_N}\sum_{k=1}{}^{y(k)}{}_1\sqrt{1+\delta_{n1}}\cos\left({}_{\pi 2N}{}^{(2k-1)(n-1)}\right)$$

- Inverse of DCT-4:

$$x(n)=\sqrt{2N_N}\sum_{k=1}{}^{y(k)}\cos\left({}_{\pi 4N}{}^{(2k-1)(2n-1)}\right)$$

All variants of the DCT are unitary (or, equivalently, orthogonal): To find the forward transforms, switch k and n in each definition. DCT-1 and DCT-4 are their own inverses. DCT-2 and DCT-3 are inverses of each other.

Chapter – 5

Code of Project

Main.java

```
package com.company;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class Main {

    public static void main(String[] args) throws IOException {
   // write your code here

        BufferedImage outputImage = new
BufferedImage(8,8,BufferedImage.TYPE_BYTE_GRAY);
       // BufferedImage hostImage = ImageIO.read(new
File("userResources/dog_384x256_1.jpg"));
        System.out.println("11");
        BufferedImage hostImage = ImageIO.read(new
File("userResources/host.png"));
        System.out.println("22");
        BufferedImage imgToEncode = ImageIO.read(new
File("userResources/sample.png"));
        System.out.println("33");
        BufferedImage encodedImg = ImageIO.read(new
File("userResources/temp.png"));
        System.out.println("44");

        //DCT dcg = new DCT(hostImage);
        encoderDCT encoderDCT = new encoderDCT(hostImage,imgToEncode);
        //decoderDCT decoderDCT = new decoderDCT(encodedImg,hostImage);

        /*
```

```
        for(int k=0;k<hostImage.getWidth();k++)
            for(int l=0;l<hostImage.getHeight();l++){
                int px = encodedImg.getRGB(k,l) - hostImage.getRGB(k,l);
                System.out.print(px+" ");
            }

         */


    }
}
```

DCT.java

```
package com.company;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import static java.lang.Math.PI;

public class DCT {

    public int N = 8;
    //double[][] output = new double[N][N];
    public final double Ci[] = new double[N];
    public final double Cj[] = new double[N];

    public double[][] quantizationTable = {
            {16,   11,   10,   16,   24,   40,   51,   61},
            {12,   12,   14,   19,   26,   58,   60,   55},
            {14,   13,   16,   24,   40,   57,   69,   56},
            {14,   17,   22,   29,   51,   87,   80,   62},
            {18,   22,   37,   56,   68,  109,  103,   77},
            {24,   35,   55,   64,   81,  104,  113,   92},
            {49,   64,   78,   87,  103,  121,  120,  101},
            {72,   92,   95,   98,  112,  100,  103,   99}
    };
```

```
    BufferedImage imageDCT = null;



    DCT(){ initializeCoefficient(); }


    private void initializeCoefficient(){
        for(int i=0;i<N;i++){
            if(i==0)
                Ci[i]=1/Math.sqrt(N);
            else
                Ci[i]=Math.sqrt(2)/Math.sqrt(N);
        }
        for(int j=0;j<N;j++){
            if(j==0)
                Cj[j]=1/Math.sqrt(N);
            else
                Cj[j]=Math.sqrt(2)/Math.sqrt(N);

        }
    }

    protected double[][] getPixelMatrix(int indexX,int indexY, BufferedImage
image){
        double token[][] = new double[N][N];

        //int indexX =208, indexY =104;
        //token
        {
            token[0][0] = image.getRGB(indexX + 0, indexY);
            token[0][1] = image.getRGB(indexX + 0, indexY + 1);
            token[0][2] = image.getRGB(indexX + 0, indexY + 2);
            token[0][3] = image.getRGB(indexX + 0, indexY + 3);
            token[0][4] = image.getRGB(indexX + 0, indexY + 4);
            token[0][5] = image.getRGB(indexX + 0, indexY + 5);
            token[0][6] = image.getRGB(indexX + 0, indexY + 6);
            token[0][7] = image.getRGB(indexX + 0, indexY + 7);
```

```
token[1][0] = image.getRGB(indexX + 1, indexY);
token[1][1] = image.getRGB(indexX + 1, indexY + 1);
token[1][2] = image.getRGB(indexX + 1, indexY + 2);
token[1][3] = image.getRGB(indexX + 1, indexY + 3);
token[1][4] = image.getRGB(indexX + 1, indexY + 4);
token[1][5] = image.getRGB(indexX + 1, indexY + 5);
token[1][6] = image.getRGB(indexX + 1, indexY + 6);
token[1][7] = image.getRGB(indexX + 1, indexY + 7);

token[2][0] = image.getRGB(indexX + 2, indexY);
token[2][1] = image.getRGB(indexX + 2, indexY + 1);
token[2][2] = image.getRGB(indexX + 2, indexY + 2);
token[2][3] = image.getRGB(indexX + 2, indexY + 3);
token[2][4] = image.getRGB(indexX + 2, indexY + 4);
token[2][5] = image.getRGB(indexX + 2, indexY + 5);
token[2][6] = image.getRGB(indexX + 2, indexY + 6);
token[2][7] = image.getRGB(indexX + 2, indexY + 7);

token[3][0] = image.getRGB(indexX + 3, indexY);
token[3][1] = image.getRGB(indexX + 3, indexY + 1);
token[3][2] = image.getRGB(indexX + 3, indexY + 2);
token[3][3] = image.getRGB(indexX + 3, indexY + 3);
token[3][4] = image.getRGB(indexX + 3, indexY + 4);
token[3][5] = image.getRGB(indexX + 3, indexY + 5);
token[3][6] = image.getRGB(indexX + 3, indexY + 6);
token[3][7] = image.getRGB(indexX + 3, indexY + 7);

token[4][0] = image.getRGB(indexX + 4, indexY);
token[4][1] = image.getRGB(indexX + 4, indexY + 1);
token[4][2] = image.getRGB(indexX + 4, indexY + 2);
token[4][3] = image.getRGB(indexX + 4, indexY + 3);
token[4][4] = image.getRGB(indexX + 4, indexY + 4);
token[4][5] = image.getRGB(indexX + 4, indexY + 5);
token[4][6] = image.getRGB(indexX + 4, indexY + 6);
token[4][7] = image.getRGB(indexX + 4, indexY + 7);

token[5][0] = image.getRGB(indexX + 5, indexY);
token[5][1] = image.getRGB(indexX + 5, indexY + 1);
token[5][2] = image.getRGB(indexX + 5, indexY + 2);
token[5][3] = image.getRGB(indexX + 5, indexY + 3);
```

```
            token[5][4] = image.getRGB(indexX + 5, indexY + 4);
            token[5][5] = image.getRGB(indexX + 5, indexY + 5);
            token[5][6] = image.getRGB(indexX + 5, indexY + 6);
            token[5][7] = image.getRGB(indexX + 5, indexY + 7);

            token[6][0] = image.getRGB(indexX + 6, indexY);
            token[6][1] = image.getRGB(indexX + 6, indexY + 1);
            token[6][2] = image.getRGB(indexX + 6, indexY + 2);
            token[6][3] = image.getRGB(indexX + 6, indexY + 3);
            token[6][4] = image.getRGB(indexX + 6, indexY + 4);
            token[6][5] = image.getRGB(indexX + 6, indexY + 5);
            token[6][6] = image.getRGB(indexX + 6, indexY + 6);
            token[6][7] = image.getRGB(indexX + 6, indexY + 7);

            token[7][0] = image.getRGB(indexX + 7, indexY);
            token[7][1] = image.getRGB(indexX + 7, indexY + 1);
            token[7][2] = image.getRGB(indexX + 7, indexY + 2);
            token[7][3] = image.getRGB(indexX + 7, indexY + 3);
            token[7][4] = image.getRGB(indexX + 7, indexY + 4);
            token[7][5] = image.getRGB(indexX + 7, indexY + 5);
            token[7][6] = image.getRGB(indexX + 7, indexY + 6);
            token[7][7] = image.getRGB(indexX + 7, indexY + 7);
        }

        //important
/*
        {
            token[0][0] = image.getRGB(indexX + 0, indexY) & 0xFF;
            token[0][1] = image.getRGB(indexX + 0, indexY + 1) & 0xFF;
            token[0][2] = image.getRGB(indexX + 0, indexY + 2) & 0xFF;
            token[0][3] = image.getRGB(indexX + 0, indexY + 3) & 0xFF;
            token[0][4] = image.getRGB(indexX + 0, indexY + 4) & 0xFF;
            token[0][5] = image.getRGB(indexX + 0, indexY + 5) & 0xFF;
            token[0][6] = image.getRGB(indexX + 0, indexY + 6) & 0xFF;
            token[0][7] = image.getRGB(indexX + 0, indexY + 7) & 0xFF;

            token[1][0] = image.getRGB(indexX + 1, indexY) & 0xFF;
            token[1][1] = image.getRGB(indexX + 1, indexY + 1) & 0xFF;
            token[1][2] = image.getRGB(indexX + 1, indexY + 2) & 0xFF;
            token[1][3] = image.getRGB(indexX + 1, indexY + 3) & 0xFF;
```

```
token[1][4] = image.getRGB(indexX + 1, indexY + 4) & 0xFF;
token[1][5] = image.getRGB(indexX + 1, indexY + 5) & 0xFF;
token[1][6] = image.getRGB(indexX + 1, indexY + 6) & 0xFF;
token[1][7] = image.getRGB(indexX + 1, indexY + 7) & 0xFF;

token[2][0] = image.getRGB(indexX + 2, indexY) & 0xFF;
token[2][1] = image.getRGB(indexX + 2, indexY + 1) & 0xFF;
token[2][2] = image.getRGB(indexX + 2, indexY + 2) & 0xFF;
token[2][3] = image.getRGB(indexX + 2, indexY + 3) & 0xFF;
token[2][4] = image.getRGB(indexX + 2, indexY + 4) & 0xFF;
token[2][5] = image.getRGB(indexX + 2, indexY + 5) & 0xFF;
token[2][6] = image.getRGB(indexX + 2, indexY + 6) & 0xFF;
token[2][7] = image.getRGB(indexX + 2, indexY + 7) & 0xFF;

token[3][0] = image.getRGB(indexX + 3, indexY) & 0xFF;
token[3][1] = image.getRGB(indexX + 3, indexY + 1) & 0xFF;
token[3][2] = image.getRGB(indexX + 3, indexY + 2) & 0xFF;
token[3][3] = image.getRGB(indexX + 3, indexY + 3) & 0xFF;
token[3][4] = image.getRGB(indexX + 3, indexY + 4) & 0xFF;
token[3][5] = image.getRGB(indexX + 3, indexY + 5) & 0xFF;
token[3][6] = image.getRGB(indexX + 3, indexY + 6) & 0xFF;
token[3][7] = image.getRGB(indexX + 3, indexY + 7) & 0xFF;

token[4][0] = image.getRGB(indexX + 4, indexY) & 0xFF;
token[4][1] = image.getRGB(indexX + 4, indexY + 1) & 0xFF;
token[4][2] = image.getRGB(indexX + 4, indexY + 2) & 0xFF;
token[4][3] = image.getRGB(indexX + 4, indexY + 3) & 0xFF;
token[4][4] = image.getRGB(indexX + 4, indexY + 4) & 0xFF;
token[4][5] = image.getRGB(indexX + 4, indexY + 5) & 0xFF;
token[4][6] = image.getRGB(indexX + 4, indexY + 6) & 0xFF;
token[4][7] = image.getRGB(indexX + 4, indexY + 7) & 0xFF;

token[5][0] = image.getRGB(indexX + 5, indexY) & 0xFF;
token[5][1] = image.getRGB(indexX + 5, indexY + 1) & 0xFF;
token[5][2] = image.getRGB(indexX + 5, indexY + 2) & 0xFF;
token[5][3] = image.getRGB(indexX + 5, indexY + 3) & 0xFF;
token[5][4] = image.getRGB(indexX + 5, indexY + 4) & 0xFF;
token[5][5] = image.getRGB(indexX + 5, indexY + 5) & 0xFF;
token[5][6] = image.getRGB(indexX + 5, indexY + 6) & 0xFF;
token[5][7] = image.getRGB(indexX + 5, indexY + 7) & 0xFF;
```

```
            token[6][0] = image.getRGB(indexX + 6, indexY) & 0xFF;
            token[6][1] = image.getRGB(indexX + 6, indexY + 1) & 0xFF;
            token[6][2] = image.getRGB(indexX + 6, indexY + 2) & 0xFF;
            token[6][3] = image.getRGB(indexX + 6, indexY + 3) & 0xFF;
            token[6][4] = image.getRGB(indexX + 6, indexY + 4) & 0xFF;
            token[6][5] = image.getRGB(indexX + 6, indexY + 5) & 0xFF;
            token[6][6] = image.getRGB(indexX + 6, indexY + 6) & 0xFF;
            token[6][7] = image.getRGB(indexX + 6, indexY + 7) & 0xFF;

            token[7][0] = image.getRGB(indexX + 7, indexY) & 0xFF;
            token[7][1] = image.getRGB(indexX + 7, indexY + 1) & 0xFF;
            token[7][2] = image.getRGB(indexX + 7, indexY + 2) & 0xFF;
            token[7][3] = image.getRGB(indexX + 7, indexY + 3) & 0xFF;
            token[7][4] = image.getRGB(indexX + 7, indexY + 4) & 0xFF;
            token[7][5] = image.getRGB(indexX + 7, indexY + 5) & 0xFF;
            token[7][6] = image.getRGB(indexX + 7, indexY + 6) & 0xFF;
            token[7][7] = image.getRGB(indexX + 7, indexY + 7) & 0xFF;
        }
    */



        //unnecessary
        printResult(token,"get pixel matrix");

        BufferedImage hostImage = new
BufferedImage(8,8,BufferedImage.TYPE_BYTE_GRAY);

        //set host image
        for(int i =0;i<N;i++){
            for (int j=0;j<N;j++){
                hostImage.setRGB(i,j, (int) token[i][j]);
            }

        }

        try {
            ImageIO.write(hostImage,"jpg",new
File("userResources/hostImage_8x8.jpg"));
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }



        return token;
    }

    protected double[][] applyDCT(double matrix[][]) {
        int i, j, k, l, N = 8;

        // dct will store the discrete cosine transform
        double[][] dct = new double[N][N];

        double ci, cj, dct1, sum;

        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                // ci and cj depends on frequency as well as
                // number of row and columns of specified matrix
/*
                if (i == 0)
                    ci = 1 / Math.sqrt(N);
                else
                    ci = Math.sqrt(2) / Math.sqrt(N);

                if (j == 0)
                    cj = 1 / Math.sqrt(N);
                else
                    cj = Math.sqrt(2) / Math.sqrt(N);


 */
                // sum will temporarily store the sum of
                // cosine signals
                sum = 0;
                for (k = 0; k < N; k++) {
                    for (l = 0; l < N; l++) {
                        dct1 = matrix[k][l] * Math.cos((2 * k + 1) * i * PI / (2 * N))
```

```
                        * Math.cos((2 * l + 1) * j * PI / (2 * N));
                    sum = sum + dct1;
                }
            }
            dct[i][j] = Ci[i] * Cj[j] * sum;
        }
    }

    double[][] op = new double[8][8];
    for (int m = 0; m < dct.length; m++) {
        for (int m2 = 0; m2 < dct.length; m2++) {
            op[m][m2] = (int) dct[m][m2];
        }
    }

    //unnecessary
    printResult(op,"DCT");


    return op;
  }

  protected int[][] initializeQuantization(double[][] unQuantizedMatrix){
    int[][] result = new int[N][N];
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++){
            result[i][j]= (int) (unQuantizedMatrix[i][j]/quantizationTable[i][j]);
        }
//unnecessary_____________________

    double[][] temp = new double[N][N];
    for(int i =0;i<N;i++){
        for (int j=0;j<N;j++){
            temp[i][j]= result[i][j];
        }
    }

    printResult(temp,"initialize Quantization");
//_____________________________________
```

```
    return result;
}

protected double [][] initializeInverseQuantization(int [][] quantized){

    double[][] result = new double[N][N];
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++){
            result[i][j]= (int) (quantized[i][j]*quantizationTable[i][j]);
        }

    //unnecessary

    printResult(result,"inverse Quantization");


    return result;

}

protected final double[][] applyIDCT(double[][] input)
{
    final int N = input.length;
    final double mathPI = Math.PI;
    final int halfN = N/2;
    final double doubN = 2.0*N;

    double[][] c = new double[N][N];
    //c = initCoefficients(c);

    double[][] output = new double[N][N];


    for (int x=0; x<N; x++)
    {
        int temp_x = 2*x+1;
        for (int y=0; y<N; y++)
        {
            int temp_y = 2*y+1;
            double sum = 0.0;
```

```
            for (int u=0; u<N; u++)
            {
               double temp_u = u*Math.PI;
               for (int v=0; v<N; v++)
               {
                  sum += Ci[u]*Cj[v] * input[u][v] *
Math.cos((temp_x/doubN)*temp_u) * Math.cos((temp_y/doubN)*v*Math.PI);
               }
            }
            //sum /= halfN;
            output[x][y] = sum;
         }
      }

      //unnecessary
      printResult(output,"IDCT");

      BufferedImage idctImage = new
BufferedImage(8,8,BufferedImage.TYPE_BYTE_GRAY);

      //set host image
      for(int i =0;i<N;i++){
         for (int j=0;j<N;j++){
            idctImage.setRGB(i,j, (int) output[i][j]);
         }

      }

      try {
         ImageIO.write(idctImage,"jpg",new
File("userResources/idctImage_8x8.jpg"));
      } catch (IOException e) {
         e.printStackTrace();
      }

      return output;
   }
```

```
   private void printResult(double[][] output, String methodName){
/*
       System.out.println("-------------------------------------");
       System.out.println("------------"+methodName+"------------");
       for(int i=0;i<N;i++) {
          for (int j = 0; j < N; j++) {

             System.out.print((int) output[i][j]+"  ");
          }
          System.out.println();
       }
       System.out.println("-------------------------------------");


 */
   }



}
```

decoderDCT.java

```
package com.company;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class decoderDCT extends DCT {

   BufferedImage hostImage = null;
   BufferedImage originalImage = null;
   BufferedImage encodedImage = null;

   int width;
   int height;
```

```
   int[][] dctMatrix = new int[N][N];
   double[][] output = new double[N][N];


   decoderDCT(BufferedImage hostImage,BufferedImage originalImage){
      super();
      this.hostImage =hostImage;
      this.originalImage = originalImage;
      this.width = hostImage.getWidth();
      this.height = hostImage.getHeight();

      encodedImage = new
BufferedImage(this.width,this.height,BufferedImage.TYPE_INT_RGB);
      decode();

      try {
         ImageIO.write(encodedImage,"png",new File("userResources/xxx.png"));
      } catch (IOException e) {
         e.printStackTrace();
      }


   }

   private void decode(){
      for(int i=0;i<this.width;i+=8)
         for(int j=0;j<this.height;j+=8){

            //dctImageMatrix =
initializeQuantization(applyDCT(getPixelMatrix(i,j,this.encodeImg)));
            //output = applyIDCT(initializeInverseQuantization(dctImageMatrix));


            for(int k=0;k<N;k++) {
               for (int l = 0; l < N; l++) {

                  //int px = this.hostImage.getRGB(i+k,j+l)+dctImageMatrix[k][l];
                  int px = this.hostImage.getRGB(i + k, j + l) -
this.originalImage.getRGB(i + k, j + l);
                  //this.imageOutput.setRGB(i+k,j+l,px);
```

```
                dctMatrix[k][l] = px;
                System.out.println((px&0xFF));

            }
            //System.out.println();
        }
        System.out.println("--------------------------");

        output = applyIDCT(initializeInverseQuantization(dctMatrix));

        for(int k=0;k<N;k++)
            for(int l=0;l<N;l++){

                encodedImage.setRGB(i+k,j+l, (int) output[k][l]);

            }



    }

  }


}
```

encoderDCT.java

```
package com.company;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class encoderDCT extends DCT {

   BufferedImage hostImage = null;
   BufferedImage encodeImg = null;
   BufferedImage imageOutput = null;
```

```
    int width;
    int height;
    int[][] dctImageMatrix = new int[N][N];

    encoderDCT(BufferedImage hostImage,BufferedImage imageToEncode){
        super();
        this.hostImage =hostImage;
        this.width = hostImage.getWidth();
        this.height = hostImage.getHeight();

        imageOutput = new BufferedImage(this.width,this.height,
BufferedImage.TYPE_INT_RGB);

        //setEncodeImgToHostImgDimension
        int widthSkip = (hostImage.getWidth()-imageToEncode.getWidth())/2;
        int heightSkip = (hostImage.getHeight()-imageToEncode.getHeight())/2;

        this.encodeImg = new
BufferedImage(hostImage.getWidth(),hostImage.getHeight(),BufferedImage.TYP
E_INT_RGB);

        //new img with host image width and height
        for(int i=0;i<imageToEncode.getWidth();i++)
            for(int j=0;j<imageToEncode.getHeight();j++)
                this.encodeImg.setRGB(i+ widthSkip,j+
heightSkip,imageToEncode.getRGB(i,j));
        //END-setEncodeImgToHostImgDimension


        encode();

        try {
            ImageIO.write(this.imageOutput,"png", new
File("userResources/temp.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
```

```
   private void encode(){
      for(int i=0;i<this.width;i+=8)
         for(int j=0;j<this.height;j+=8){

            dctImageMatrix =
initializeQuantization(applyDCT(getPixelMatrix(i,j,this.encodeImg)));
            //output = applyIDCT(initializeInverseQuantization(dctImageMatrix));


            for(int k=0;k<N;k++)
               for(int l=0;l<N;l++){

                  int px = this.hostImage.getRGB(i+k,j+l)+dctImageMatrix[k][l];

                  System.out.println(px);
                  this.imageOutput.setRGB(i+k,j+l,px);
               }
         }

   }

}
```

Chapter – 6

OUTPUT

Cover Image

Stego Image

Output Stego Image

Chapter - 7

Conclusion and Future Enhancement

This project is intended to improve performance by using steganography and cryptography technologies. Because the data is hidden in the image, the attacker will have no way of knowing that it is being hidden. By building a Java application, we were able to perform our procedure on an image. This project describes a system for securely communicating between two private parties while transmitting significant amounts of secret information. The key benefit of this system is that it use DCT transformation steganography techniques, which are extremely difficult to detect. Also, the proposed approach is easy and simple to execute.

We intend to try out the proposed method on audio and video in the future.

Chapter - 8

Reference

[1] D. Seth, L. Ramanathan, and A. Pandey, “Security enhancement: Combining cryptography and steganography,” International Journal of Computer Applications (0975–8887) Volume, 2010.

[2] H. Abdulzahra, R. AHMAD, and N. M. NOOR, “Combining cryptography and steganography for data hiding in images,” ACACOS, Applied Computational Science, pp. 978–960, 2014.

[3] J. V. Karthik and B. V. Reddy, “Authentication of secret information in image stenography,” International Journal of Computer Science and Network Security (IJCSNS), vol. 14, no. 6, p. 58, 2014.

[4] M. H. Rajyaguru, “Crystography-combination of cryptography and steganography with rapidly changing keys,” International Journal of Emerging.

[5] M. K. I. Rahmani and N. P. Kamiya Arora, “A crypto-steganography: A survey,” International Journal of Advanced Computer Science and Application, vol. 5, pp. 149–154, 2014.

[6] Mr. Vikas Tyagi(2012), “Data Hiding in Image Using least significant bit with cryptography”, International Journal of Advanced Research in computer science and Software Engineering, Volume 2, Issue 4.

[7] P. R. Ekatpure and R. N. Benkar, “A comparative study of steganography & cryptography,” 2013.