VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Hiding confidential data of a industry with image steganography using LSB embedding

Network Security and Cryptography Fundamentals (CSE1029)

Slot: B2

Team Members

20MIA1031 - Sanjay.M
20MIA1039 - Sriganth.R
20MIA1117 - Pilaram Manoj

TABLE OF CONTENTS

Abstract

Steganography is one of the methods used for the hidden exchange of information, and it is described as the study of invisible communication, which mainly deals with methods of hiding the presence of the conveyed message. As a result, if the communication is successfully sent, attackers are not drawn to it. Information may be buried in various embedding materials, known as carriers, using steganography. Images, audio files, video files, and text files can all be used as carriers. Because the focus of this study is on using an image file as a carrier, a description of existing steganographic approaches for image files has been offered. These strategies are examined and explored not just in terms of their capacity to hide information in picture files, but also in terms of how much information can be hidden and their ability to resist various image processing attacks.Steganography is one of the methods used for the hidden exchange of information and it can be defined as the study of invisible communication that usually deals with the ways of hiding the existence of the communicated message. In this way, if successfully achieved, the message does not attract attention from eavesdroppers and attackers. Using steganography, information can be hidden in different embedding mediums, known as carriers. These carriers can be images, audio files, video files, and text files.

Introduction about the project

Consider an industry for a moment. It contains a range of manufacturing locations. Their monthly production levels vary depending on previous sales prediction. Therefore, the branches received the private product manufacturing information from the main branch. If data is hacked by an attacker, the company's reputation could damage. Thus, safety must be maintained during this operation. Here, utilising the steganography approach, the text can be hidden behind the image file. In general, according to the purposes for which information hiding is used, information hiding techniques can be divided into two, and they are steganography and watermarking. There has been a surge of interest in incorporating these techniques in recent years. Based on the domain type, steganography methods can be divided into two spatial and transform. In the cover image, spatial domain procedures entrench the hidden data directly. On the other hand, transforming domain procedures embed the data after transforming the image of the cover into another domain. Spatial domain algorithms such as LSB take less time to perform and have a higher embedding performanceTherefore, as a result of this research, a steady information system based on a modified least significant bit (LSB) algorithm was suggested. The proposed system was introduced in the programming environment of Java. This functions by moving the LSB of the red (), green (), and blue () of concealed object pixel components to the given number of times the sender determines. The bits of the undisclosed message would then be substituted for the moving bits. The aim of our research is to send the message secretly to the destination, encrypt the secret image using a steganography technique and hide the encrypted image in cover image using LSB technique, and at the receiving end the encrypted image is first retrieved from the image and then decrypted to obtain secret image.

Module Description

- Using java.awt.Image, Image IO.
- The package contains all the necessary classes and methods along with interfaces that are necessary for the manipulation of the images
- User space is created for preserving the original file, so that all the modifications are done in the user space.
- In the object of buffered image, using the IO.read method we take the original image.
- Using the create graphics and draw rendered image method of graphics class, we create our user space in a buffered image object.
- The text file is taken as input and separated in a stream of bytes.
- Now each bit of these bytes are encoded in the lsb of each next pixel, finally we get the final image that contains the encoded message and is saved ,at the specified path given by the user, in PNG format using Image IO.write method.

METHODOLOGY

We will be conducting the following steps

- Import Required Packages
- Upload the image
- Convert to binary
- Encryption of the text message into the image
- Decoding of the message, decryption & source message retrieval.

Hardware /software used

Edition	Windows 11 Home Single Language
Experience	Windows Feature Experience Pack 1000.22000.1219.0
Device name	LAPTOP-61IV32JE
Processor	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
Installed RAM	8.00 GB
System type	64-bit operating system, x64-based processor
Free space	200 MB free space

Language used Java
IntelliJ IDEA Community Edition 2022.2.1
jDK 16

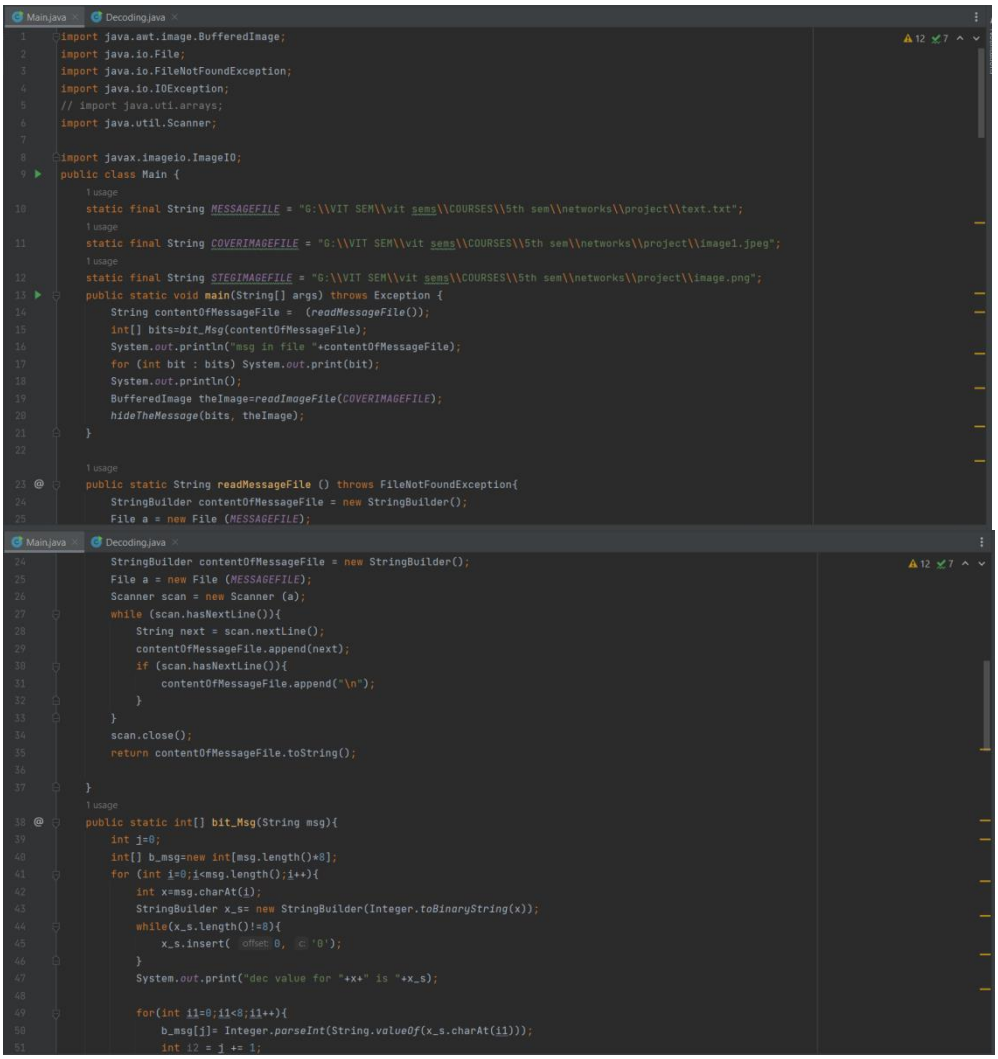
Files

- main.java is encoding code file
- decoding.java is decoding code file

Sample coding

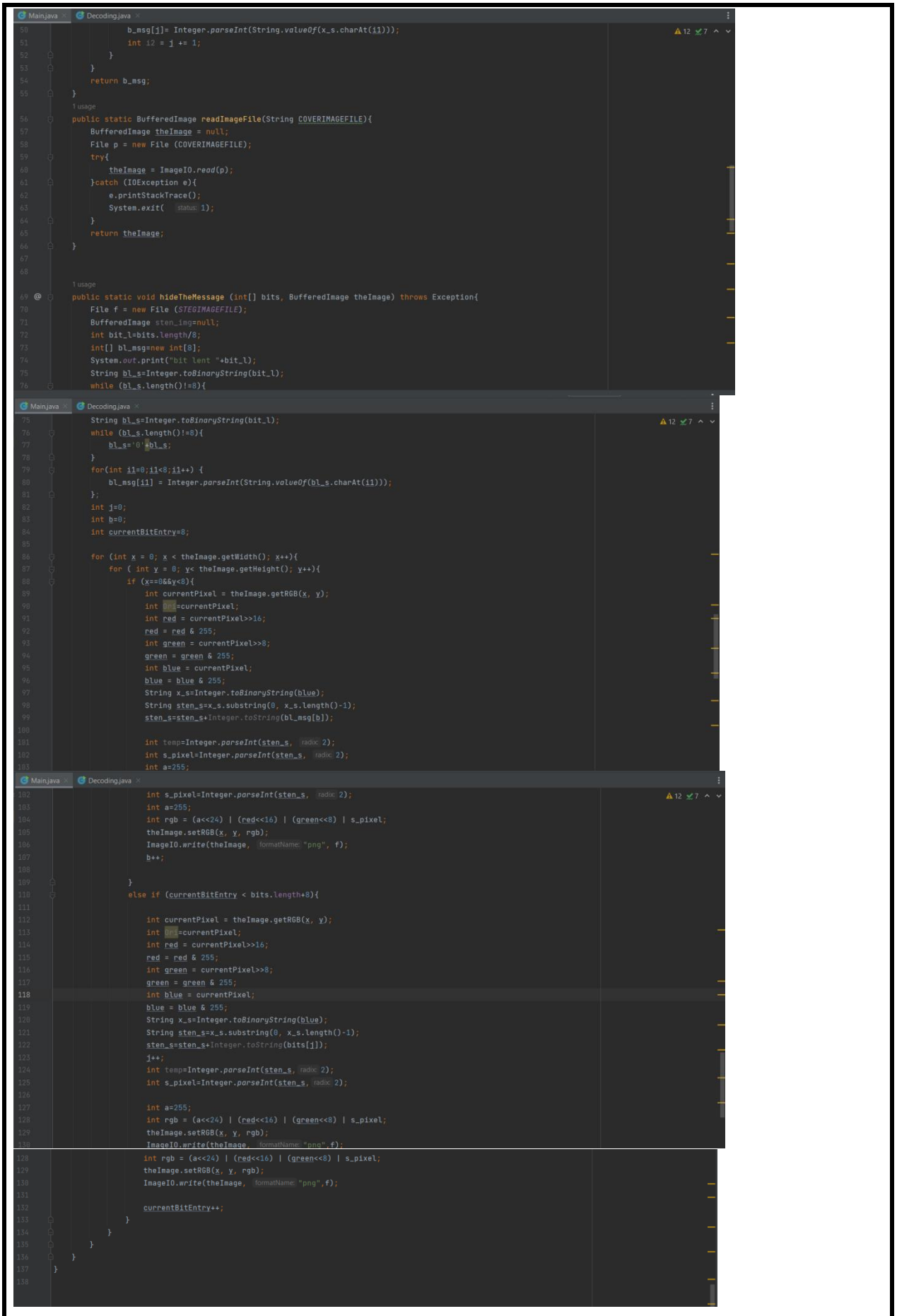
Encoding

- User space is created for preserving the original file, so that all the modifications are done in the user space.
- In the object of buffered image, using the IO.read method we take the original image.
- Using the create graphics and draw rendered image method of graphics class, we create our user space in a buffered image object.
- The text file is taken as input and separated in a stream of bytes.
- Now each bit of these bytes are encoded in the lsb of each next pixel, finally we get the final image that contains the encoded message and is saved ,at the specified path given by the user, in PNG format using Image IO.write method.



```
Main.java  Decoding.java
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
// import java.uti.arrays;
import java.util.Scanner;

import javax.imageio.ImageIO;
public class Main {
    1 usage
    static final String MESSAGEFILE = "G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\networks\\project\\text.txt";
    1 usage
    static final String COVERIMAGEFILE = "G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\networks\\project\\image1.jpeg";
    1 usage
    static final String STEGIMAGEFILE = "G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\networks\\project\\image.png";
    public static void main(String[] args) throws Exception {
        String contentOfMessageFile =  (readMessageFile());
        int[] bits=bit_Msg(contentOfMessageFile);
        System.out.println("msg in file "+contentOfMessageFile);
        for (int bit : bits) System.out.print(bit);
        System.out.println();
        BufferedImage theImage=readImageFile(COVERIMAGEFILE);
        hideTheMessage(bits, theImage);
    }

    1 usage
    public static String readMessageFile () throws FileNotFoundException{
        StringBuilder contentOfMessageFile = new StringBuilder();
        File a = new File (MESSAGEFILE);
```

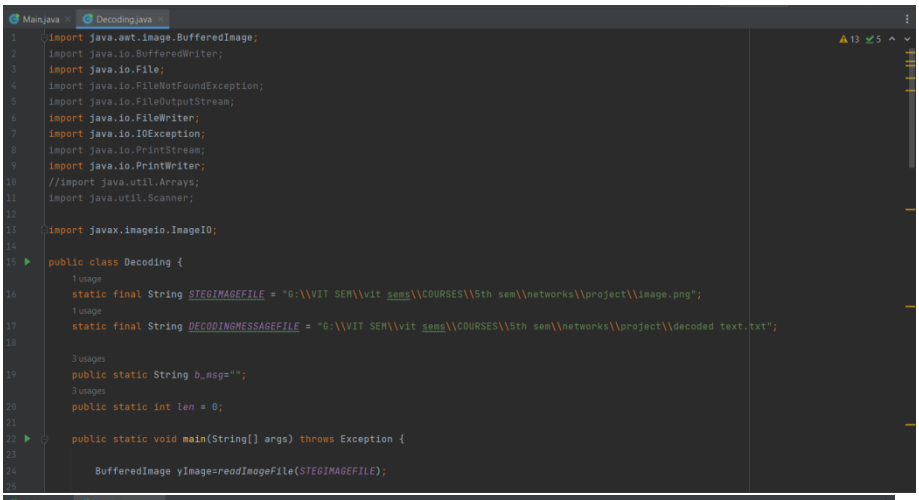
```
Main.java  Decoding.java
        StringBuilder contentOfMessageFile = new StringBuilder();
        File a = new File (MESSAGEFILE);
        Scanner scan = new Scanner (a);
        while (scan.hasNextLine()){
            String next = scan.nextLine();
            contentOfMessageFile.append(next);
            if (scan.hasNextLine()){
                contentOfMessageFile.append("\n");
            }
        }
        scan.close();
        return contentOfMessageFile.toString();

    }
    1 usage
    public static int[] bit_Msg(String msg){
        int i=0;
        int[] b_msg=new int[msg.length()*8];
        for (int i=0;i<msg.length();i++){
            int x=msg.charAt(i);
            StringBuilder x_s= new StringBuilder(Integer.toBinaryString(x));
            while(x_s.length()!=8){
                x_s.insert( offset: 0,  c: '0');
            }
            System.out.print("dec value for "+x+" is "+x_s);

            for(int i1=0;i1<8;i1++){
                b_msg[j]= Integer.parseInt(String.valueOf(x_s.charAt(i1)));
                int i2 = j += 1;
```


Main.java
Decoding.java
b_msg[j]= Integer.parseInt(String.valueOf(x_s.charAt(i1)));
int i2 = j += 1;
}
}
return b_msg;
}
1 usage
public static BufferedImage readImageFile(String COVERIMAGEFILE){
BufferedImage theImage = null;
File p = new File (COVERIMAGEFILE);
try{
theImage = ImageIO.read(p);
}catch (IOException e){
e.printStackTrace();
System.exit(status: 1);
}
return theImage;
}
1 usage
public static void hideTheMessage (int[] bits, BufferedImage theImage) throws Exception{
File f = new File (STEGIMAGEFILE);
BufferedImage sten_img=null;
int bit_l=bits.length/8;
int[] bl_msg=new int[8];
System.out.print("bit lent "+bit_l);
String bl_s=Integer.toBinaryString(bit_l);
while (bl_s.length()!=8){
String bl_s=Integer.toBinaryString(bit_l);
while (bl_s.length()!=8){
bl_s='0'+bl_s;
}
for(int i1=0;i1<8;i1++) {
bl_msg[i1] = Integer.parseInt(String.valueOf(bl_s.charAt(i1)));
};
int j=0;
int b=0;
int currentBitEntry=8;
for (int x = 0; x < theImage.getWidth(); x++){
for (int y = 0; y< theImage.getHeight(); y++){
if (x==0&&y<8){
int currentPixel = theImage.getRGB(x, y);
int brg=currentPixel;
int red = currentPixel>>16;
red = red & 255;
int green = currentPixel>>8;
green = green & 255;
int blue = currentPixel;
blue = blue & 255;
String x_s=Integer.toBinaryString(blue);
String sten_s=x_s.substring(0, x_s.length()-1);
sten_s=sten_s+Integer.toString(bl_msg[b]);
int temp=Integer.parseInt(sten_s, radix: 2);
int s_pixel=Integer.parseInt(sten_s, radix: 2);
int a=255;
int s_pixel=Integer.parseInt(sten_s, radix: 2);
int a=255;
int rgb = (a<<24) | (red<<16) | (green<<8) | s_pixel;
theImage.setRGB(x, y, rgb);
ImageIO.write(theImage, formatName: "png", f);
b++;
}
else if (currentBitEntry < bits.length+8){
int currentPixel = theImage.getRGB(x, y);
int brg=currentPixel;
int red = currentPixel>>16;
red = red & 255;
int green = currentPixel>>8;
green = green & 255;
int blue = currentPixel;
blue = blue & 255;
String x_s=Integer.toBinaryString(blue);
String sten_s=x_s.substring(0, x_s.length()-1);
sten_s=sten_s+Integer.toString(bits[j]);
j++;
int temp=Integer.parseInt(sten_s, radix: 2);
int s_pixel=Integer.parseInt(sten_s, radix: 2);
int a=255;
int rgb = (a<<24) | (red<<16) | (green<<8) | s_pixel;
theImage.setRGB(x, y, rgb);
ImageIO.write(theImage, formatName: "png",f);
currentBitEntry++;
}
}
}
}
}

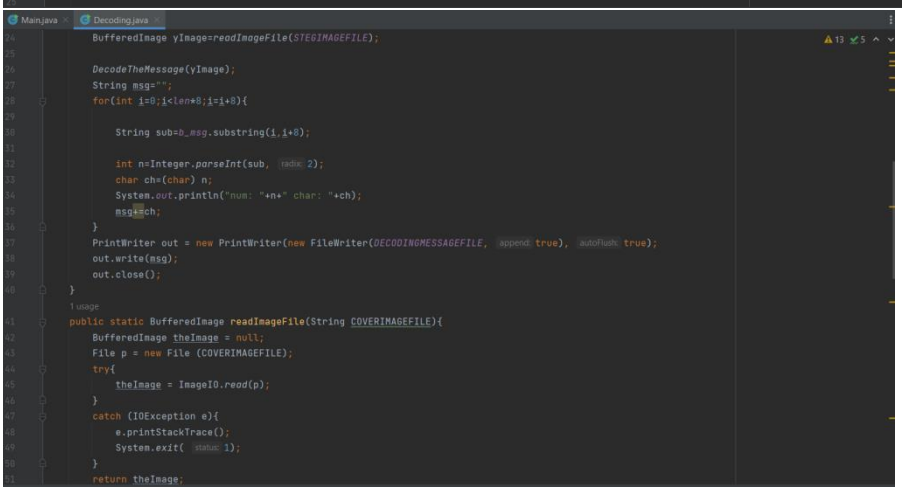
```
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.1\lib\idea_rt.jar=51482:C:\Program Files\JetBrains\
dec value for 68 is 01000100dec value for 101 is 01100101dec value for 99 is 01100011dec value for 101 is 01100101dec value for 109 is 01101101dec value for 98 is 01100010dec va
Red ten thousand
Blue twelve thousand
Yellow seven thousand
Green six thousand
Black three thousand
0100010001100101011000110110010101101101011000100110010101110010001000001110000011011101110100011000010111010001101110000001100011011010000110100101110000011100110010000000
bit lent 126
```

Decoding

- The offset of the image is retrieved from its header
- Create the user space using the same process as in Encoding.
- Using getRaster() and getDataBuffer() methods of writable Raster and Data Buffer Byte classes. The data of the image is taken into a byte array.
- Using the above byte array, the bit stream of the original text file, is retrieved into another byte array.
- Above byte array is written into the decoded text file, which leads to the original message.



```
import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintStream;
import java.io.PrintWriter;
//import java.util.Arrays;
import java.util.Scanner;

import javax.imageio.ImageIO;

public class Decoding {
    1 usage
    static final String STEGIMAGEFILE = "G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\networks\\project\\image.png";
    1 usage
    static final String DECODINGMESSAGEFILE = "G:\\VIT SEM\\vit sems\\COURSES\\5th sem\\networks\\project\\decoded text.txt";

    3 usages
    public static String b_msg="";
    3 usages
    public static int len = 0;

    public static void main(String[] args) throws Exception {

        BufferedImage yImage=readImageFile(STEGIMAGEFILE);
```



```
        BufferedImage yImage=readImageFile(STEGIMAGEFILE);

        DecodeTheMessage(yImage);
        String msg="";
        for(int i=0;i<len*8;i=i+8){

            String sub=b_msg.substring(i,i+8);

            int n=Integer.parseInt(sub, radix: 2);
            char ch=(char) n;
            System.out.println("num: "+n+" char: "+ch);
            msg+=ch;
        }
        PrintWriter out = new PrintWriter(new FileWriter(DECODINGMESSAGEFILE, append: true), autoFlush: true);
        out.write(msg);
        out.close();
    }
    1 usage
    public static BufferedImage readImageFile(String COVERIMAGEFILE){
        BufferedImage theImage = null;
        File p = new File (COVERIMAGEFILE);
        try{
            theImage = ImageIO.read(p);
        }
        catch (IOException e){
            e.printStackTrace();
            System.exit( status: 1);
        }
        return theImage;
```

```java
        }
        return theImage;
    }

    1 usage
    public static void DecodeTheMessage (BufferedImage yImage) throws Exception{

        int j=0;
        int currentBitEntry=0;
        String bx_msg="";
        for (int x = 0; x < yImage.getWidth(); x++){
            for (int y = 0; y < yImage.getHeight(); y++){
                if(x==0&&y<8){
                    int currentPixel = yImage.getRGB(x, y);
                    int red = currentPixel>>16;
                    red = red & 255;
                    int green = currentPixel>>8;
                    green = green & 255;
                    int blue = currentPixel;
                    blue = blue & 255;
                    String x_s=Integer.toBinaryString(blue);
                    bx_msg+=x_s.charAt(x_s.length()-1);
                    len=Integer.parseInt(bx_msg, radix: 2);

                }

                else if(currentBitEntry<len*8){
                    int currentPixel = yImage.getRGB(x, y);
                    int red = currentPixel>>16;
                    red = red & 255;
                    int green = currentPixel>>8;
                    green = green & 255;
                    int blue = currentPixel;
                    blue = blue &255;
                    String x_s=Integer.toBinaryString(blue);
                    b_msg+=x_s.charAt(x_s.length()-1);

                    currentBitEntry++;
                }
            }
        }
        System.out.println("bin value of msg hided in img is "+b_msg);
    }
}
```

```
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.2.1\lib\idea_rt.jar=51466:C:\Program Files\JetBrains\I
bin value of msg hided in img is 0100010001100101011000110100101011010110001001100101011100100010000001110000011011110110001100001011010001101110010000011000110110100
num: 68 char: D
num: 101 char: e
num: 99 char: c
num: 101 char: e
num: 109 char: m
num: 98 char: b
num: 101 char: e
num: 114 char: r
```

Result

Input

Input image :-

Input text file:-

text - Notepad

File Edit View

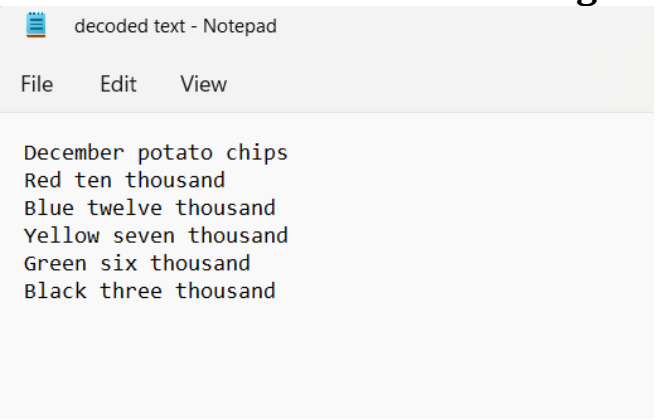
```
December potato chips
Red ten thousand
Blue twelve thousand
Yellow seven thousand
Green six thousand
Black three thousand
```

Encoded image

image

we can see that the encoded image looks exactly the same as the input image. The only difference between the encoded image is that there is a hidden text message inside it, which is not visible and hidden well.

Decoded text from encoded image



CONCLUSION

It is observed that through the LSB Substitution Steganographic method, the results obtained in data hiding are pretty impressive as it utilizes the simple fact that any image could be broken up to individual bit-planes each consisting of different levels of information.
In this paper, a technique is presented that integrates steganography to encrypt the Hiding confidential data of a industry data and embed it into a product image. steganography enhance industry data security, and ensure data confidentiality, and integrity. Steganography is done by embedding the encrypted data into the LSBs of the industry's images. The experimental results shows that the quality of stego image is relatively less distortive, preserve the sensitive data and increase the data payload. The PNG large file size has a good advantage for hiding data with less distortion and with high payload capacity.

REFERENCES

1. S.A. Halim and M.F.A Sani. “Embedding using spread spectrum image steganography with GF ” in Proc. IMT-GT-ICMSA, 2010, pp. 659-666.
2. N.Hamid, A.Yahya, R. B. Ahmad & O. M. Al-Qershi, Image Steganography Techniques: An Overview,IJCSS-670,2012.
3. T. Morkel, J.H.P. Eloff, and M.S. Oliver. “An overview of image steganography.” in Proc. ISSA, 2005, pp. 1-11.