Project Report

On Steg

Submitted in Partial fulfillment for the award of degree of

Bachelor of Technology in Information Technology

Submitted to

Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)

Submitted by:

Prince kumar (0126IT191088)

Palak Singh (0126IT191078)

Sakshi sen (0126IT191098)

Rajhanshi singh(0126IT191092)

Under the Guidance of

Prof. Ruchi Jain

Department of Information Technology

Oriental college of technology

Approved by AICTE New Delhi & Govt. of M.P.

Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)

ORIENTAL COLLEGE OF TECHNOLOGY, BHOPAL

Approved by AICTE New Delhi & Govt. of M.P. &
Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal(M.P).

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the work embodied in this Project, Dissertation Report as **"Steg" being Submitted by Prince Kumar (0126IT191088), Palak Singh (0126IT191078), Sakshi Sen (0126IT191098),Rajhanshi Singh (0126IT191092)**

in partial fulfillment of the requirement for the award of "Bachelor of Technology" in Information Technology discipline to Rajiv Gandhi Proudyogiki Vishwavidyalaya,Bhopal (M.P.) during the academic year 2022-23 is a record of bonafide piece of work, carried out under my supervision and guidance in the Department of Information Technology, Oriental College of Technology, Bhopal.

Approved By:-

Prof. Amit Kanskar -Head of Department

Dr. Amita Mahor - Director

ORIENTAL COLLEGE OF TECHNOLOGY, BHOPAL

Approved by AICTE New Delhi & Govt. of M.P. &
Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal(M.P).

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE OF APPROVAL

This Project **"Steg" being submitted by Prince Kumar (0126IT191088), Palak singh (0126IT191078), Sakshi Sen (0126IT191098),Rajhanshi singh (0126IT191092).**

Has been examined by me & hereby approved for the partial fulfillment of the requirement for the award of **" Bachelor of Technology in Information Technology "** for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but the Project only for the purpose for which it has been submitted.

CANDIDATE DECLARATION

We hereby declare that the Project dissertation work presented in the report entitled as "Steg" submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology from Oriental College of Technology is an authentic record of our own work.

We have not submitted the part and partial of this report for the award of any other degree or diploma.

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

Prince Kumar(0126IT191088)

Palak singh(0126IT191078)

Sakshi Sen (0126IT191098)

Rajhanshi singh(0126IT191092)

ACKNOWLEDGMENT

We are heartily thankful to the **Management of Oriental College of Technology** for providing us all the facilities and infrastructure to take our work to the final stage.

It is the constant supervision, moral support and proper guidance of our respected Director **Dr. Amita Mahor**, who motivated throughout the work.

We express a deep sense of gratitude and respect to our learned guide **Mr. Sapna Kushwaha,**Professor in the Department of Information technology, during all phases of our work.With out her enthusiasm and encouragement this dissertation would not have been completed. His valuable knowledge and innovative ideas helped us to take the work to the final stage. He has timely suggested actions and procedures for which we are really grateful and thankful to him. We express our gratefulness to **Prof. Amit Kanskar** Head of Information Technology Department for providing all the facilities available in the department for his continuous support, advice, and encouragement during this work and also help to extend our knowledge and proper guidelines.Constant help, moral and financial support of our loving parents motivated us to complete the work. We express our heartfelt thanks to all our family members for their cooperation.

We really admire the fond support of our class-mates for their cooperation and constant help.It gives immense pleasure to acknowledge the encouragement and support extended by them.Last but not the least we are extremely thankful to all who have directly or indirectly helped us for the completion of the work.

Prince Kumar(0126IT191088)

Palak singh(0126IT191078)

Sakshi Sen (0126IT191098)

Rajhanshi singh(0126IT191092)

ABSTRACT

1. Introduction

Steganography is defined as "the art of hiding messages inside media files", in other words, it is the way in which we can hide any message (text, image, audio, …) inside any file (.mp3, .wav, .png, …). This helps people to make sure that only those who know about the presence of the message can obtain it. There are many different methods of performing steganography, but the most famous are the LSB (Least Significant Bit) ones. This kind of method modifies the LSB of different bytes with a bit from the message that will be hidden.

2. Objective

The common modern technique of steganography exploits the property of the media itself to convey a message. The following media are the candidate for digitally embedding message : -

- Plaintext
- Still imagery
- Audio and Video
- IP datagram.

3. Methodology

In this method we are storing both parts of the sink image at the end of the data part of the container image with a stego key, finally we get the stego image, which contains container image and sink image together. In view of the fact that the structural part of container image is unchanged, the visual appearance of both container image and new stego-image will be remain same. The advantage of this method is solely due to its simplicity, which can easily be seen from Fig. 2. The pseudo code for the above methodology is given below. /* Pseudo Code for Methodology-I */

Addition Method { Array container_image []=LoadImage(“Container Image”, Length);

Array sink_image [] = LoadImage (“Sink Image”, Length); Array stego_image; String stego_key; Integer cont_data_part,cont_str_part,snk_data_part,snk_str_part; Integer i, j ,k; Input cont_data_part, cont_str_part,snk_data_part,snk_str_part; For i=0 to cont_str_part Stego_image [i] = container_image [i]; End for; For i= cont_str_part to cont_data_part Stego_image [i] = container_image [i]; End for; For j=0 to Length (stego_key) Stego_image [j] = stego_key [i]; End for; For k=0 to snk_str_part Stego_image [k] = sink_image [k]; End for; For k= snk_str_part to snk_data_part Stego_image [k] = sink_image [k]; End for; SaveImage (stego_image []) RemoveImage(container_image []); RenameImage (stego_image, container_image)

4. ALGORITHM DESIGN AND IMPLEMENTATION

4.1 Image Steganography

ENCODE:-

Using Modified LSB Algorithm where we overwrite the LSB bit of actual image with the bit of text message character. At the end of the text message we push a delimiter to the message string as a checkpoint useful in the decoding function. We encode data in order of Red, then Green and then Blue pixel for the entire message.

```
def encode_img_data(img):
    data=input()        # input text message data

    data +='*^*^*'      # adding delimiter to the end of text message to mark end of string

    binary_data=msgtobinary(data)         #converting this whole string to binary data format

    length_data=length(binary_data)       #length of binary data file
    index_data = 0                        #variable which act as a counter

    for i in img:
        for pixel in i:
            r, g, b = msgtobinary(pixel)  #converting each pixel to binary data format
            if index_data < length_data:
                pixel[0] = int(r[:-1] + binary_data[index_data], 2)  #overwrite the LSB bit of red pixel with the bit of
                index_data += 1                                      #text message character
            if index_data < length_data:
                pixel[1] = int(g[:-1] + binary_data[index_data], 2)  #overwrite the LSB bit of green pixel with the bit of
                index_data += 1                                      #text message character
            if index_data < length_data:
                pixel[2] = int(b[:-1] + binary_data[index_data], 2) #overwrite the LSB bit of blue pixel with the bit of
                index_data += 1                                     #text message character
            if index_data >= length_data:                           #loop run to overwrite LSB bit till the length of text
                break                                               #message binary length

    cv2.imwrite(nameoffile,img)
```

DECODE:-

In the decode part, we take **all the LSB bits of each pixel** until we get a checkpoint/delimiter and then **we split them by 8 bits** and convert them to characters data type and **print the string (i.e., the secret text message) without delimiter.**

```python
def decode_img_data(img):
    data_binary = ""                #string to store each LSB bit of every pixel
    for i in img:
        for pixel in i:
            r, g, b = msgtobinary(pixel)   #converting each pixel to binary data format
            data_binary += r[-1]           #adding red pixel LSB bit
            data_binary += g[-1]           #adding green pixel LSB bit
            data_binary += b[-1]           #adding blue pixel LSB bit
            total_bytes = [ data_binary[i: i+8] for i in range(0, len(data_binary), 8) ]  #split the string bits into 8 bit
            decoded_data = ""           #string to store and check if we extracted the complete text message
            for byte in total_bytes:
                decoded_data += chr(int(byte, 2))   #convert the binary data format file back to character string format
                if decoded_data[-5:] == "*^*^*":   # delimiter which we use as check point to show we have reached the end of
                    print(decoded_data[:-5])       # secret text message which was encoded in Stego Image
                    return
```

4.2 **Text Steganography**

<u>Encode:</u>

ZWCs- In Unicode, there are specific zero-width characters (ZWC). We used four ZWCs for hiding the SM through the CT.

The embedding algorithm contains following stages:-

Secret Message (SM): This can be secret or confidential information.

Cover Text (CT): This is an innocent text that can be any type of meaningful text.

For every character of the secret message :-

We get its ascii value and it is incremented or decremented based on if ascii value between 32 and 64 , it is incremented by 48(ascii value for 0) else it is decremented by 48

Then xor the the obtained value with 170(binary equivalent-10101010)

Convert the obtained number from first two step to its binary equivalent then add "0011" if it earlier belonged to ascii value between 32 and 64 else add "0110" making it 12 bit for each character.

With the final binary equivalent we also 111111111111 as delimiter to find the end of message

Now from 12 bit representing each character every 2 bit is replaced with equivalent ZWCs according to the table. Each character is hidden after a word in the cover text

2 bit classification	Hexcode
00	0x200C
01	0x202C
11	0x202D
10	0x200E

Zero width character table

```
def txt_encode(text):
    l=len(text)                 #Length of message to be encoded
    while i<l:
        t=ord(text[i])          # getting ascii value of each character
        if(t>=32 and t<64):     # if ascii is between 32 and 64
            t1=t+48             #increment the ascii value by 48
            t2=t1^170           #xoring with 170 (binary value-10101010)
            res = bin(t2)[2:].zfill(8)      #converting obtained value to 8bit binary value
            add+="0011"+res     #adding 0011 to making it 12 bit
        else:                   #for any other cases
            t1=t-48             #decrement the ascii value by 48
            t2=t1^170           #xoring with 170 (binary value-10101010)
            res = bin(t2)[2:].zfill(8)      #converting obtained value to 8bit binary value
            add+="0110"+res     #adding 0110 to making it 12 bit
        i+=1
    res1=add+"111111111111"     #adding 111111111111 as delimiter to find the end point
    HM_SK=""
    ZWC={"00":u'\u200C',"01":u'\u202C',"11":u'\u202D',"10":u'\u200E'}
                                                #assigning every 2bit combination with a zero width character
    file1 = open("covertext.txt","r+")          #opening cover file for the message
    file3= open(nameoffile,"w+", encoding="utf-8") #creating stego_file for storing
    word=[]
    for line in file1:
        word+=line.split()    #storing every word from the cover file
    while(i<len(res1)):
        s=word[int(i/12)]
        j=0
        while(j<12):
            x=res1[j+i]+res1[i+j+1]        #taking 2bit at a time
            HM_SK+=ZWC[x]    #comparing with every 2bit combination and storing apprpriate zwc
            j+=2
        s1=s+HM_SK           #embedding secret message every charater zwc at end of every word from the coverfile
        file3.write(s1)      #writing in the stegofile
        i+=12
    t=int(len(res1)/12)
    while t<len(word):       #for writing remaining words of coverfile into stegofile
        file3.write(word[t])
        t+=1
```

Decode:

After receiving a stegofile , the extraction algorithm discovers the contractual 2-bit of each ZWCs , every 12 bit from end of the word in the stego file and then the binary equivalent is completely extracted and delimiter discussed above helps us in getting to the end point. Now we divide the 12 bit into two parts first 4 bit and another 8bit on which we do the xor operation with 170(binary

value 10101010). Now according to the first 4bit if its is "0110" we increment it by 48 else we decrement by 48. At last we convert the ascii value into its equivalent character to get the final hidden message from the stego file

```
def decode_txt_data():
    ZWC_reverse={u'\u200C':"00",u'\u202C':"01",u'\u202D':"11",u'\u200E':"10"}
                                                #reversing every 2bit combination with a zero width character accordingly
    file4= open(stegofile,"r", encoding="utf-8")    #opening stego file for the extraction message
    for line in file4:
        for words in line.split():
            T1=words
            binary_extract=""
            for letter in T1:                       #selecting every character for a word from stegofile
                if(letter in ZWC_reverse):          #checking for matching zero width character
                    binary_extract+=ZWC_reverse[letter]  #storing respective bits
            if binary_extract=="111111111111":      #checking with delimiter if the message reaches its end point break
                break
            else:
                temp+=binary_extract                #else store it
    a=0
    b=4
    c=4
    d=12
    final=''                #for storing final decoded secret message
    while i<len( temp):
        t3=temp[a:b]        #accessing first 4 bit of the 12 bit for each character
        a+=12
        b+=12
        i+=12
        t4=temp[c:d]        #accessing remaining 8 bit of the 12 bit for each character
        c+=12
        d+=12
        if(t3=='0110'):                             #if first 4 bit is 0110
            for i in range(0, len(t4), 8):          #for converting 8 bit into its ascii value
                temp_data = t4[i:i + 8]
                decimal_data = BinaryToDecimal(temp_data)
            final+=chr((decimal_data ^ 170) + 48)   #xoring with 170 (binary value-10101010) and incrementing by 48
        elif(t3=='0011'):                           #if first 4 bit is 0011
            for i in range(0, len(t4), 8):
                temp_data = t4[i:i + 8]
                decimal_data = BinaryToDecimal(temp_data)
            final+=chr((decimal_data ^ 170) - 48)   #xoring with 170 (binary value-10101010) and decrementing by 48
```

4.3 **Audio Steganography**

<u>Encode:-</u>

We will be using Cover Audio as a Cover file to encode the given text. Wave module is used to read the audio file. Firstly we convert our secret message to its binary equivalent and added delimiter '*****' to the end of the message. For encoding we have modified the LSB Algorithm, for that we take each frame byte of the converting it to 8 bit format then check for the 4th LSB and see if it matches with the secret message bit. If yes change the 2nd LSB to 0 using logical AND operator between each frame byte and 253(11111101). Else we change the 2nd LSB to 1 using logical AND operation with 253 and then logical OR to change it to 1 and now add secret message bit in LSB for achieving that use logical AND operation between each frame byte of carrier audio and a binary number of 254 (11111110). Then logical OR operation between modified carrier byte and the next bit (0 or 1) from the secret message which resets the LSB of carrier byte.

```
def encode_aud_data():
    import wave
    song = wave.open(nameoffile, mode='rb')    #opening the cover audio

    frame_bytes=bytearray(list(song.readframes(song.getnframes())))#reading each frame and converting to byte array
    data = 'secret message'   #secret message
    data = data + '*****'     #adding delimiter at the end of the message
    #converting text to bit array
    result = []
    for c in data:
        bits = bin(ord(c))[2:]
        bits = '00000000'[len(bits):] + bits #modify the carrier byte according to the text message such that we overwrite
                                             #each character bit to the end of 8 bit format
        result.extend([int(b) for b in bits])
    j = 0
    for i in range(0,len(result),1):
        res = bin(frame_bytes[j])[2:].zfill(8)          #converting the frame bytearray into its 8 bit binary format
        if res[len(res)-4]== result[i]:                 # checking if the 4th-lsb matches with secret message bit
            frame_bytes[j] = (frame_bytes[j] & 253)     #we perform logical and between each frame byte and 253
                                                        #which set the 2nd lsb to 0 in frame byte
        else:                                   #if the above condition fails
            frame_bytes[j] = (frame_bytes[j] & 253) | 2 #we perform logical and between each frame byte and 253
                                                        #and then doing or operator with 2 to set 2nd lsb to 1

            frame_bytes[j] = (frame_bytes[j] & 254) | result[i] #again we perform logical and between each frame byte and 254
                                                                #which sets lsb to 0 then we do or operation with message bit
                                                                #to store it in lsb
        j = j + 1
    frame_modified = bytes(frame_bytes)                 #getting the modified bits
    stegofile=input("\nEnter name of the stego file (with extension) :- ")
    with wave.open(stegofile, 'wb') as fd:              #writing bytes into a new wave audio file
        fd.setparams(song.getparams())
        fd.writeframes(frame_modified)
    song.close()
```

Decode:-

We start the extraction process by reading each frame and converting it to byte array. After that we check 2nd LSB if it is 0 or 1. If the bit is 1 we store the LSB of the frame byte else we store the 4th LSB, we keep this process until we reach the delimiter and then we break from the loop, then convert the message into characters and print it.

```
def decode_aud_data():
    import wave
    song = wave.open(nameoffile, mode='rb')        #opening the stego audio

    frame_bytes=bytearray(list(song.readframes(song.getnframes())))  #reading each frame and converting to byte array
    extracted = ""      #for storing the extracted bit
    p=0                 #counter
    for i in range(len(frame_bytes)):
        if(p==1):       #checks if the recovered message has reached delimiter(end point) we break
            break
        res = bin(frame_bytes[i])[2:].zfill(8)     #converting the frame bytearray into its 8 bit binary format
        if res[len(res)-2]==0:                     #checking 2nd lsb if it is 0
            extracted+=res[len(res)-4]             #we add 4th lsb to extracted
        else:
            extracted+=res[len(res)-1]             #else add lsb
    #Converting the decoded bits to Characters
        all_bytes = [ extracted[i: i+8] for i in range(0, len(extracted), 8) ]

        decoded_data = ""
        for byte in all_bytes:
            decoded_data += chr(int(byte, 2))
            if decoded_data[-5:] == "*****":       #Checking if we have reached the delimeter which is "*****"
                print("The Encoded data was :--",decoded_data[:-5])   # we print the hidden message separating the delimiter
                p=1            #change counter to 1
                break
```

4.4 **Video Steganography**

In video steganography we have used combination of cryptography abd Steganography. We encode the message through two parts

We convert plaintext to cipher text for doing so we have used RC4 Encryption Algorithm. RC4 is a stream cipher and variable-length key algorithm. This algorithm encrypts one byte at a time. It has two major parts for encryption and decryption:-

KSA(Key-Scheduling Algorithm)- A list S of length 256 is made and the entries of S are set equal to the values from 0 to 255 in ascending order. We ask user for a key and convert it to its equivalent ascii code. S[] is a permutation of 0,1,2....255, now a variable j is assigned as j=(j+S[i]+key[i%key_length) mod 256 and swap S(i) with S(j) and accordingly we get new permutation for the whole keystream according to the key.

PRGA(Pseudo random generation Algorithm (Stream Generation)) - Now we take input length of plaintext and initiate loop to generate a keystream byte of equal length. For this we initiate i=0, j=0 now increment i by 1 and mod with 256. Now we add S[i] to j amd mod of it with 256 ,again swap the values. At last step take store keystreambytes which matches as S[(S[i]+S[j]) mod 256] to finally get key stream of length same as plaintext.

Now we xor the plaintext with keystream to get the final cipher.

```
def KSA(key):#Key-Scheduling Algorithm
    key_length = len(key) #length of key
    S=list(range(256))   #The entries of S are set equal to the values from 0 to 255 in ascending order (a general permutation)
    j=0
    for i in range(256):
        j=(j+S[i]+key[i % key_length]) % 256     #assigning value to j to get different permutation according to the key
        S[i],S[j]=S[j],S[i]                      #swap S[i] and S[j]
    return S                                     #return a different permutation of 0-255 for this perticular key
```

```
def PRGA(S,n):#Pseudo random generation algorithm (Stream Generation)
    i=0
    j=0
    key=[]
    while n>0:                              #n is length of our plain text
        n=n-1
        i=(i+1)%256                          # increment i and mod with 256 so it won't get out of bound
        j=(j+S[i])%256                       # now add S[i] value to j and the mod 256
        S[i],S[j]=S[j],S[i]                  #swaping the values
        K=S[(S[i]+S[j])%256]                 #generates keystreambyte by adding value at that particular index
        key.append(K)
    return key
```

```
def preparing_key_array(s):
    return [ord(c) for c in s]                  #converts character of s into its ordinal code
```

```
def encryption(plaintext):
    key=input()                                         #enter the key
    key=preparing_key_array(key)                        #Key converted to ordinal characters

    S=KSA(key)                                          #calling KSA for a special permutation of 0-255 for this key

    keystream=np.array(PRGA(S,len(plaintext)))          #Keystreambyte generation using PRGA
    plaintext=np.array([ord(i) for i in plaintext])     #converts plaintext to its ordinal code

    cipher=keystream^plaintext              #xoring every byte of keystream with plaintext to get cipher text
    ctext=''
                                            #cipher text is generated
    for c in cipher:
        ctext=ctext+chr(c)
    return ctext
```

Now for the Steganography part we will be using Modified LSB Algorithm where we overwrite the LSB bits of the selected frame (given by the user) from the cover video, with the bit of text message character. At the end of text message we add a delimiter to the message string as a endpoint which comes useful in decoding function. We encode data in order of Red, then Green and then Blue pixel for the entire message of the selected frame.

```
def embed(frame):
    data=input()                            #Enter the data to be Encoded in Video
    data=encryption(data)                   #data is encrypted using RC4 encryption algorithm
    if (len(data) == 0):
        raise ValueError()                  #Data entered to be encoded is empty

    data +='*^%^*'                          # add delimiter as checkpoint

    binary_data=msgtobinary(data)           #msg to binary
    length_data = len(binary_data)
    index_data = 0
    # message is encoded in the frame using LSB algorithm
    for i in frame:
        for pixel in i:
            r, g, b = msgtobinary(pixel)#converting each pixel to binary data format
            if index_data < length_data:
                pixel[0] = int(r[:-1] + binary_data[index_data], 2) #overwrite the LSB of red pixel with the bit of
                                                                    #text message bit
                index_data += 1
            if index_data < length_data:
                pixel[1] = int(g[:-1] + binary_data[index_data], 2) #overwrite the LSB of green pixel with the bit of
                                                                    #text message bit
                index_data += 1
            if index_data < length_data:
                pixel[2] = int(b[:-1] + binary_data[index_data], 2) #overwrite the LSB of blue pixel with the bit of
                                                                    #text message bit
                index_data += 1
            if index_data >= length_data:               #when message is over it breaks from the loop
                break
    return frame
```

Decode:-

In decode part In the decode part, we take the encoded frame from the stego video, in the frame each pixels last LSB is stored until we get to the delimiter as we reach there we split them by 8 bits and convert them to characters data type now we go to the decryption process where we do the same as encode, make Keystream with help of secret key and using KSA and PRGA and finally xoring with the obtained data from the frame with keystream to get the final decoded secret message

```
def decryption(ciphertext):
    key=input()                                 #enter the key
    key=preparing_key_array(key)                #key converted to ordinal characters

    S=KSA(key)                                  #calling KSA for a special permutation of 0-255 for this key

    keystream=np.array(PRGA(S,len(ciphertext)))        #keystreambyte generation using PRGA
    ciphertext=np.array([ord(i) for i in ciphertext])  #converts ciphertext to its ordinal code

    decoded=keystream^ciphertext                #xoring every byte of keystream with ciphertext to get decoded text
    dtext=''
    #finally decoded
    for c in decoded:
        dtext=dtext+chr(c)
    return dtext
```

```
def extract(frame):
    data_binary = ""                            #storing each LSB bit of every pixel
    final_decoded_msg = ""
    for i in frame:
        for pixel in i:
            r, g, b = msgtobinary(pixel)        #converting this whole string to binary data format
            data_binary += r[-1]                #storing red pixel lsb
            data_binary += g[-1]                #storing green pixel lsb
            data_binary += b[-1]                #storing blue pixel lsb
            total_bytes = [ data_binary[i: i+8] for i in range(0, len(data_binary), 8) ] #split string bits into 8 bit
            decoded_data = ""                   #store extracted bits
            for byte in total_bytes:
                decoded_data += chr(int(byte, 2))# convert binary to character
                if decoded_data[-5:] == "$^$^$": #reaching end by help of delimiter
                    for i in range(0,len(decoded_data)-5):
                        final_decoded_msg += decoded_data[i]
                    final_decoded_msg = decryption(final_decoded_msg) #decrypting the message received and storing it
                    return
```

5. Result and output

```
                    STEGANOGRAPHY

                    MAIN MENU

1. IMAGE STEGANOGRAPHY {Hiding Text in Image cover file}
2. TEXT STEGANOGRAPHY {Hiding Text in Text cover file}
3. AUDIO STEGANOGRAPHY {Hiding Text in Audio cover file}
4. VIDEO STEGANOGRAPHY {Hiding Text in Video cover file}
5. Exit

Enter the Choice: 1
```

5.1 Image Steganography

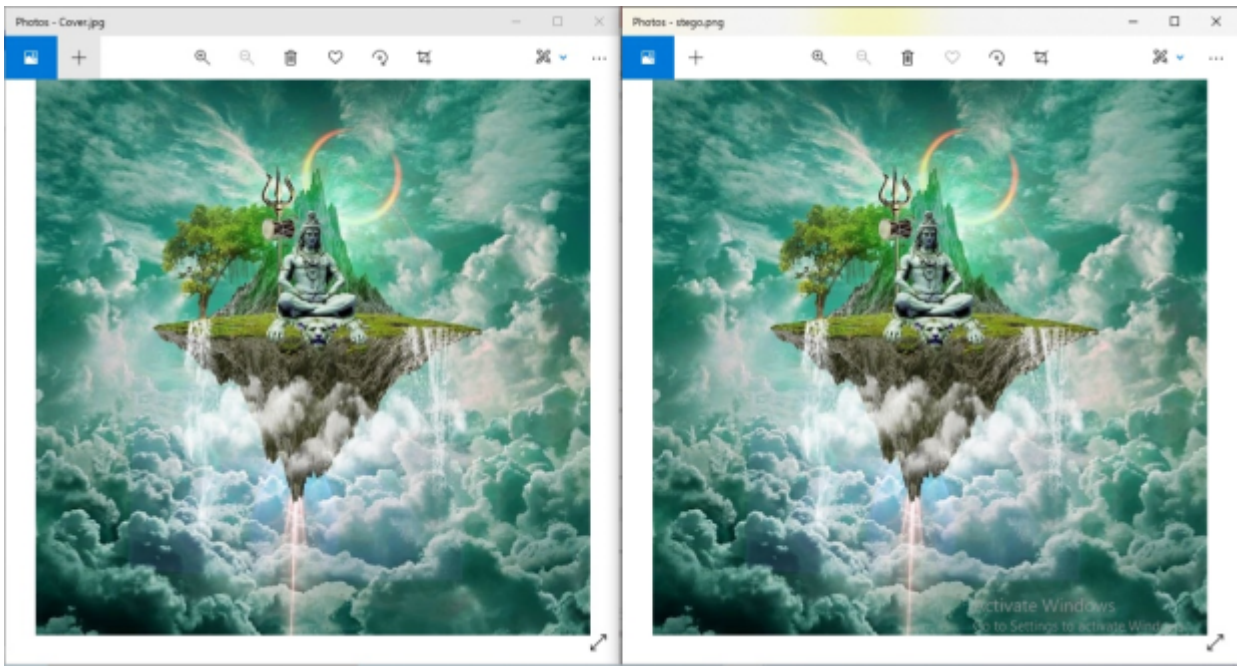
Encoding the message in image file

```
                IMAGE STEGANOGRAPHY OPERATIONS

1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice: 1

Enter the data to be Encoded in Image :This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1 batch 1.Priyansh S
harma 2.Koustubh sinha 3.Vaibhav Kansal

Enter the name of the New Image (Stego Image) after Encoding(with extension):stego.png

Maximum bytes to encode in Image : 203136


010101000110100001101001011100110010000001101001011100110010000001101111011101010111001000100000011011010110100101101110011011
101110010001000000111000001110010011011110110101001100101011000110111010000100000011011110110111000100000011101000110100000110
010100000011101000110111101110000011010010110001100100000001000100101001101010100010001010100011101000001010011100100111101000
111010100100100000101010000010010000101100100100010001011100010000001010011010001010100111010010000000110101001000000100011001
001001101111011011010010000001100110001100010010000001100010011000010111010001100011011010000010000000110001001011100101000001
100100110100101111001011000010110111001110011010100000100000010100110111010000110000101110010011011010110000100100000011001000
101110010010110110111011101101010111001101110100011101010110001001101000001000000111001101101001011011100110100001100001001000000
011001100101110010101100110000101101001011000100110100001100001011101100010000001001011011000010110111001110011011000010110110
001010100101111000101010010111100010101
```

The Length of Binary data 1056

Encoded the data successfully in the Image and the image is successfully saved with name stego.png

Preview of Cover image and Stego image:



Decoding the message from Image file:

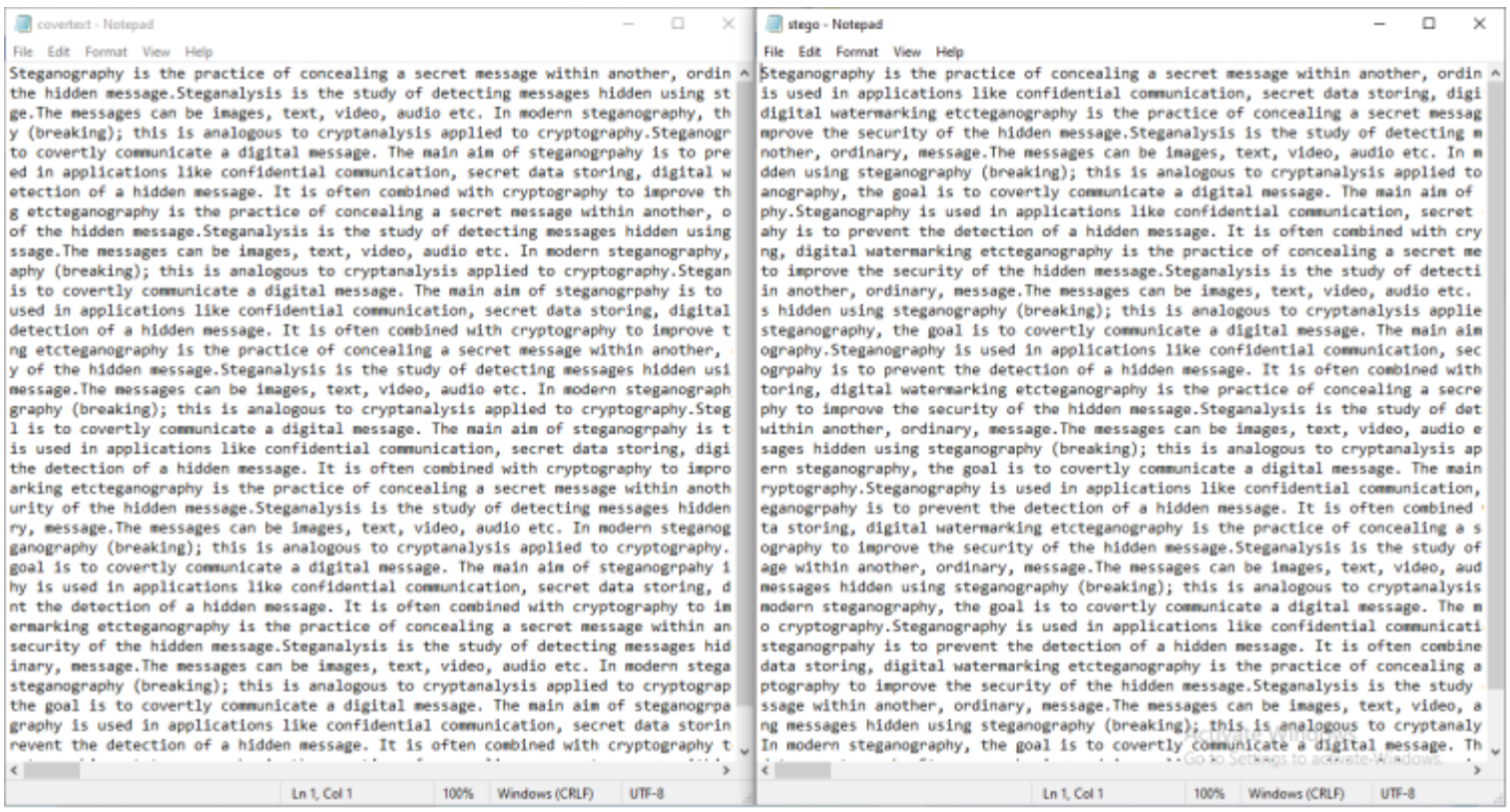
```
                IMAGE STEGANOGRAPHY OPERATIONS

1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice: 2
Enter the Image you need to Decode to get the Secret message :  stego.png


The Encoded data which was hidden in the Image was :--   This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1
batch 1.Priyansh Sharma 2.Koustubh sinha 3.Vaibhav Kansal
```

5.2 Text Steganography

Encoding the message in text file:

```
                TEXT STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:1
Maximum number of words that can be inserted :-  879

Enter data to be encoded:- This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1 batch 1.Priyansh Sharma 2.Kous
tubh sinha 3.Vaibhav Kansal

Inputed message can be hidden in the cover file

The string after binary conversion appyling all the transformation :- 0110100011100110100100100111010010011011011110100100111111
0100111010010011011011110100100111111110100110100101010110111011101110110111000001111111010011010010111011010010011011010010100001100
1001010101101110100000111111110100110111010100110111010000011010010101011010010000011010011111011010011001011011101110001111111010
0011010010101011010010100001111111010011011101110011010010010011010011111001111111010011011101110011010010101011011101010011010
0100110110100110010011111110100011111110000110100010010110100011100110101111101101011110101101110110100011010110100
1101011110101101000100001101011101101101000101001101011001001101000001100111111100000111111010000111111101001101000100101011011
1111011010110111001111111010001111001111001111111010011010111100011011101000011010010101011010010111001111111010011010011100001
1110010110011111111010011010011000011010011011011011101110011010011001011010010010001111111010001111001011001111111010001101000010
1001101111010000110100100110110111000110110100110111011010010110100011011101001011010010010001111111010011010001001011010010010011101
0011011011011101000011010010111011010011011001111111010001111100100000111111101000110101100010110100101010110111011110110111001101
0110111011100110111011110111010011000011010010010001111111101001101110100101101001001101110100101000110100100100110100110110011111
1101000111100100100111111101000110100011000110100110110110100100110111010011010010010011010011011011011101100000111111101001
1010110001011010011011011010010100011011101001011010011011011010010110111111111111
Length of binary after conversion:-  1536

Enter the name of the Stego Key file after Encoding(with extension):- stego.txt

Stego file has successfully generated
```


Preview of Cover Text file and Stego Text file:



Decoding the message from Text file:

```
                TEXT STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:2

Please enter the stego file name(with extension) to decode the message:- stego.txt

Encrypted message presented in code bits: 0110100011100110100100100110100100110110111010010011111110100110100100110110111010010
0111111101001101001010101101110111011101110100000111111101001101001011101101001001101101001010001101001010101101110100000111111
1010011011101010011011101000011010010101011010010000011010011111011010011001011011101110001111111010011010010101011010010100001
1111110100110111011100110100100100110100111110011111110100110111011100110100101010110111010100110100100110110100110010011111110
1000111111100001101000100101101000111001101011111101101011110101101011101101101101000110101101010110101111010110100010000110 1
0111011011010001010011010110010011010000011001111111000001111110100001111111010011010001001011010111111011010110111001111111010
0011110011110011111111010011010111100011011010000110100101010110100101110011111111010011010011100001111001011001111111010011010
1100001101001101101101111011100110100110010110100100100011111110100011110010110011111110100011010001010011011101000011010010011 01
1011100011011010011011011010010100011011101001011010010010001111111010011010001001011010010010011010011011011011101000011010010
1110110100110110011111111010001111001000001111111010001101011000101101001010101110111011110111010010111011011100110111011111010
1001100001101001001000111111110100111011010010110100100111011010010100011010010010011010011011001111111010001111001001001111110 10
0011010001100011010011011011010010011011010011000011010010010011010011011011011011000011111110100111010110001010011011011010
0101000110111010010110100111011011010010
```

Length of encoded bits:- 1524

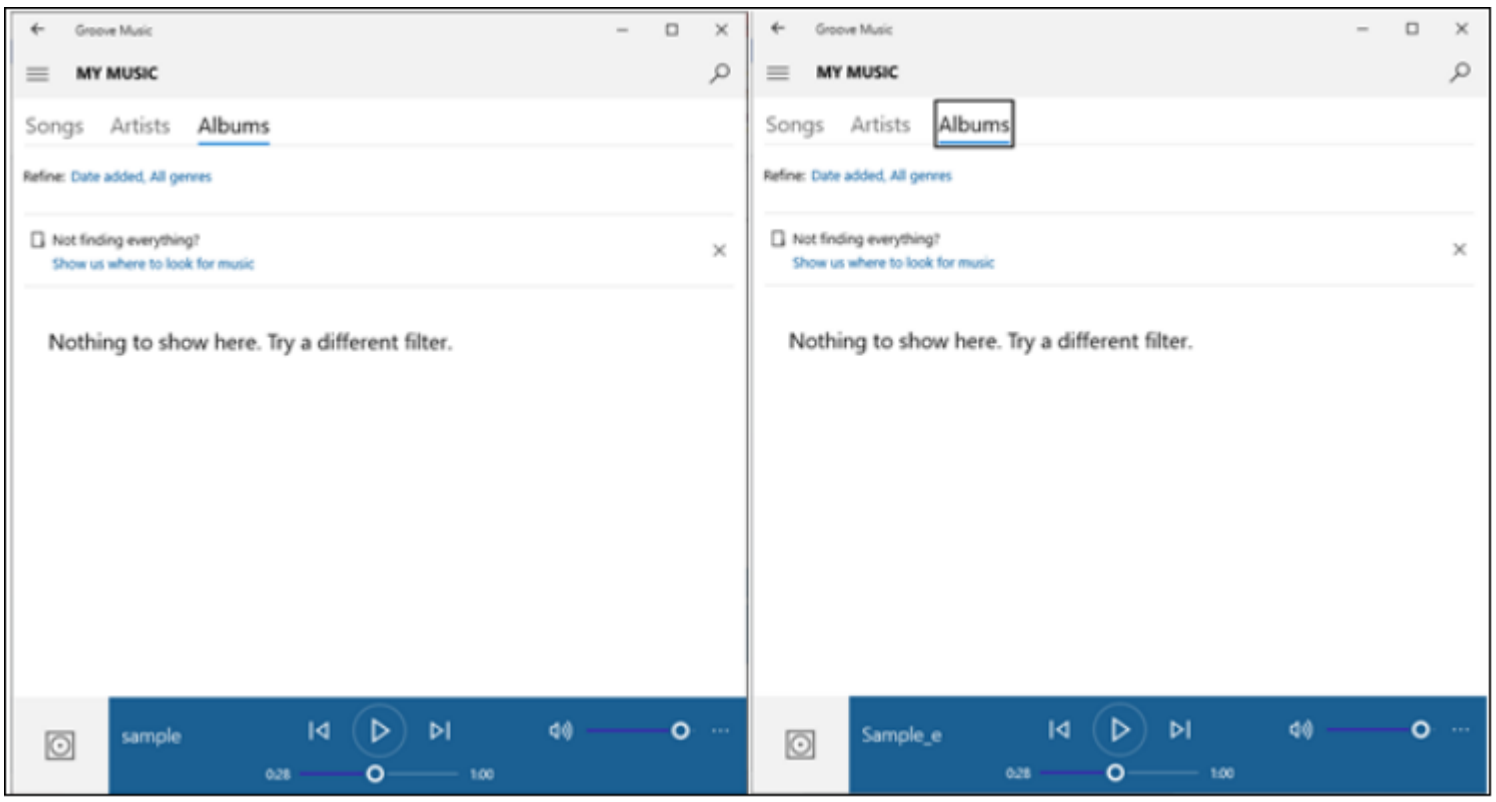
Message after decoding from the stego file:- This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1 batch 1.Priyansh Sharma 2.Koustubh sinha 3.Vaibhav Kansal

5.3 Audio Steganography:

Encoding the message in Audio file:

```
                AUDIO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:1
Enter name of the file (with extension) :- sample.wav

Enter the secret message :- This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1 batch 1.Priyansh Sharma 2.Kou
stubh sinha 3.Vaibhav Kansal

The string after binary conversion :- 0101010001101000011010010111001100100000011010010111001100100000011011110111010101110010
0100000011011010110100101101110011011110111001000100000011100000111001001101111011010100110010101100011011110100001000000011011111
0110111000100000011101000110100001100101001000000111010001101111011100000110100101100011001000000010001001010011010101000100010
1010001110100000101001110010011110100011101010010010000010101000001001000010110010010001000101110001000000101001101000101010011
0100100000001101010010000001000110011100100110111101101101001000000110011000110001001000000110001001100001011101000110001101101
0000010000000110001001011100101000001110010011010010111100101100001011011100111001101100010111010000100000010100111011010000110000101111
0010011011010110000100100000001100100010111001001011011011110111010101110011011101000111010101100010011010000100000011100110111
0100101101110011010000110000100100000001100110010111001010110011000010110100101100010011010000110000101110110001000000100101101
1000010110111001110011011000010110110 0
```

Preview of Cover Audio file and Stego Audio file:



Decoding the message from Audio file:

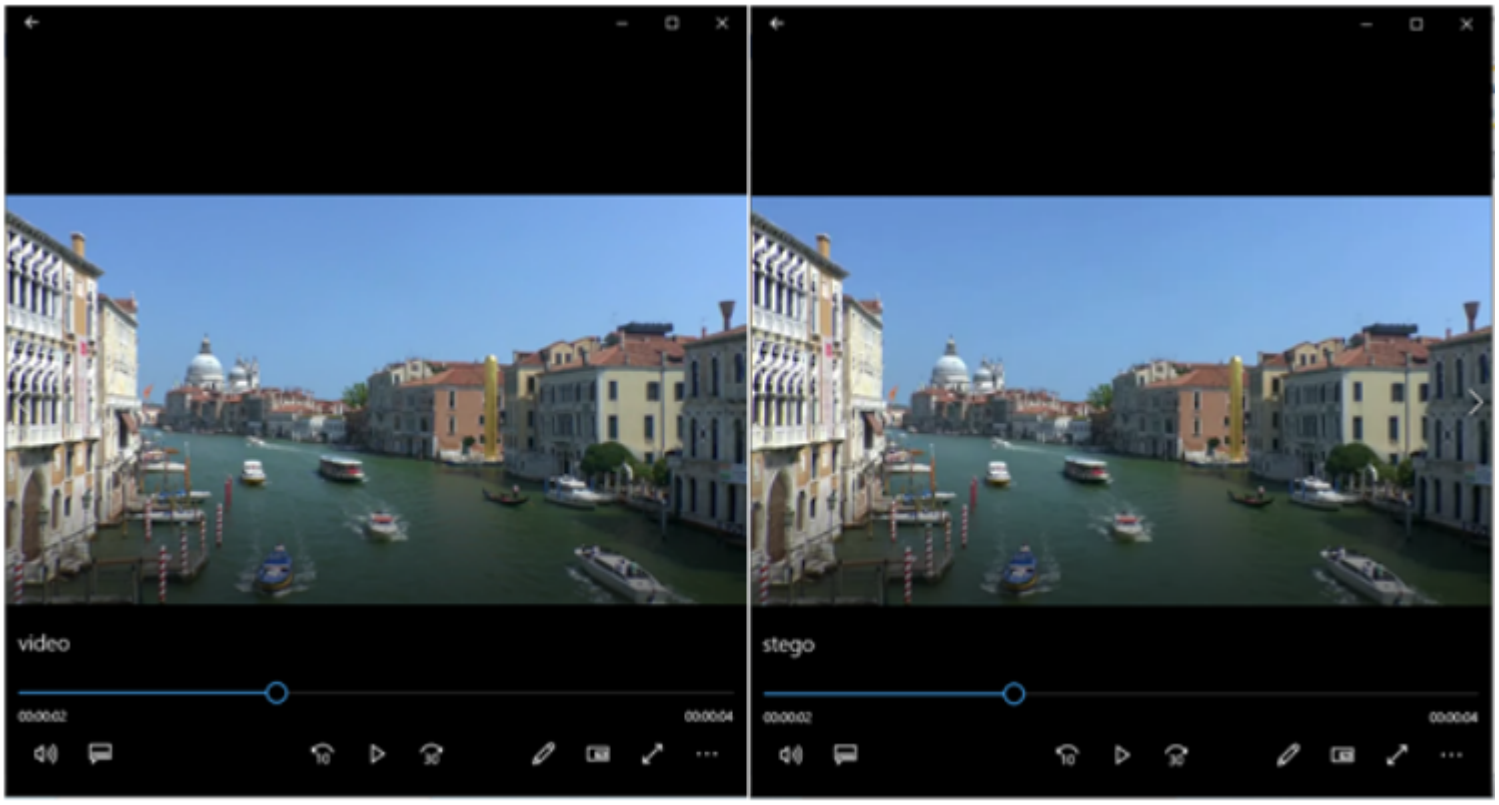
```
                AUDIO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:2
Enter name of the file to be decoded :- stego.wav
The Encoded data was :-- This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1 batch 1.Priyansh Sharma 2.Koustu
bh sinha 3.Vaibhav Kansal
```

5.4 **Video Steganography:**

Encoding the message in Video file:

```
                VIDEO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:1
Total number of Frame in selected Video : 172
Enter the frame number where you want to embed data :
6

Enter the data to be Encoded in Video :This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1 batch 1.Priyansh S
harma 2.Koustubh sinha 3.Vaibhav Kansal
Enter the key :
key
□(kUþù§□^6úÇ-/□□zæ4□q :  _□]□□j'>z[{4|□ýT□□Ä□□□BÓÇÓ□□□û4IÙþÀi□□`□ºöb3iÄ¨|ho5ü'¤ *u□¶§□yÌ□Ì¶□U°ÙF
j□÷□□Ö+TnÅã□$9□□.□¢Ù9□M□+Þ□åZ&

Encoded the data successfully in the video file.
```

Preview of Cover Video file and Stego Video file:



Decoding the message from Audio file:

```
                VIDEO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:2
Total number of Frame in selected Video : 172
Enter the secret frame number from where you want to extract data
6
Enter the key :
key


The Encoded data which was hidden in the Video was :--
 This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1 batch 1.Priyansh Sharma 2.Koustubh sinha 3.Vaibhav Kansa
l
```

Conclusion:

Steganography can protect data by hiding it but using it alone may not guarantee total protection. It is possible that by using a steganocryption technique, enemy detects presence of text message in theimage file and then he/she may succeed in extracting information from the picture, which can be disastrous in real life situations. This is same for plainencryption. In this case by seeing the meaningless appearing sequence of bit enemy can detect that some illegal message is being sent (unless he/she is afool), and we may land –up in a problematic situation.However, if one uses both methods, this will lead to 'security in depth'. The message should first be encoded using a strong encryption algorithm and then embedded into a carrier.

Reference:

[1] G. Sahoo and R.K. Tiwari " Designing an Embedded Algorithm for Data Hiding using Steganographic Technique by File Hybridization", IJCSNS International Journal of Computer Science and Network Security, VOL. 8 No. 1 January 2008.

[2] Donovan Artz " Digital Steganography: Hiding Data within Data", IEEE Internet computing, pp.75-80 May –June 2001.

[3] Mitchell D.Swanson, Bin Zhu and Ahmed H. Tewfik "TRANSPARENT ROBUST IMAGE WATERMAKING" IEEE 0-7803-3258-x/96. 1996

[4] M.M Amin, M. Salleh, S. Ibrahim, M.R. Katmin, and M.Z. I. Shamsuddin " Information hiding using Steganography" IEEE 0-7803-7773-March 7, 2003.