Multi-Media Steganography using PRGA and KSA algorithms

Project Report

Submitted in fulfilment for the J-Component of CSE3501 – Information Security, Analysis and Audit

in

B.Tech. (Computer Science)

by

Nilaa (20BDS0311)

Syed (20BDS0394)

Aadesh (20BDS0168)

Pranav Saravanan (20BCE0974)

Pradeep Kumar (20BDS0183)

Under the guidance of

Prof. Sivakumar Sir

SCOPE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Fall Semester 2022-2023

Index

Sl.no	*Topic*	*Page no*
1	Abstract	3
2	Introduction	3-6
3	Working of the model	6-16
4	Screenshots	6-16
5	Result and Outputs	16-22
6	Conclusion	22

ABSTRACT

Steganography is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity. Consequently, functions provided by Stefano only hide messages, without encryption. Steganography is often used with cryptography.

We have implemented Steganography in 4 different Types of media

1. Image
2. Text
3. Audio
4. Video

INTRODUCTION

Steganography is the practice of representing information within another message or physical object, in such a manner that the presence of the information is not evident to human inspection. In computing/electronic contexts, a computer file, message, image, or video is concealed within another file, message, image, or video. The word steganography comes from Greek steganographia, which combines the words steganós meaning "covered or concealed", and -graphic meaning "writing".

The first recorded use of the term was in 1499 by Johannes Trithemius in his Steganographia, a treatise on cryptography and steganography, disguised as a book on magic. Generally, the hidden messages appear to be (or to be part of) something else: images, articles, shopping lists, or some other cover text. For example, the hidden message may be in invisible ink between the visible lines of a private letter. Some implementations of steganography that lack a shared secret are forms of security through obscurity, and key-dependent steganographic schemes adhere to Kerckhoffs's principle.

The advantage of steganography over cryptography alone is that the intended secret message does not attract attention to itself as an object of scrutiny. Plainly visible encrypted messages, no matter how unbreakable they are, arouse interest and may in themselves be incriminating in countries in which encryption is illegal.

Whereas cryptography is the practice of protecting the contents of a message alone, steganography is concerned with concealing the fact that a secret message is being sent and its contents.

Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program, or protocol. Media files are ideal for steganographic transmission because of their large size. For example, a sender might start with an innocuous image file and adjust the color of every hundredth pixel to correspond to a letter in the alphabet. The change is so subtle that someone who is not specifically looking for it is unlikely to notice the change.

Steganography has been widely used for centuries. Some examples include:

Hidden messages on a paper written in secret inks.

Hidden messages distributed, according to a certain rule or key, as smaller parts (e.g. words or letters) among other words of a less suspicious cover text. This particular form of steganography is called a null cipher.

Messages written in Morse code on yarn and then knitted into a piece of clothing worn by a courier.

Messages written on envelopes in the area covered by postage stamps.

In the early days of the printing press, it was common to mix different typefaces on a printed page because the printer did not have enough copies of some letters in one typeface. Thus, a message could be hidden by using two or more different typefaces, such as normal or italic.

A microdot camera

During and after World War II, espionage agents used photographically-produced microdots to send information back and forth. Microdots were typically minute (less than the size of the period produced by a typewriter). World War II microdots were embedded in the paper and covered with an adhesive, such as collodion that was reflective and so was detectable by viewing against glancing light. Alternative techniques included inserting microdots into slits cut into the edge of postcards.

During World War II, Velvalee Dickinson, a spy for Japan in New York City, sent information to accommodation addresses in neutral South America. She was a dealer in dolls, and her letters discussed the quantity and type of doll to ship. The stegotext was the doll orders, and the concealed "plaintext" was itself encoded and gave information about ship movements, etc. Her case became somewhat famous and she became known as the Doll Woman.

During World War II, photosensitive glass was declared secret[by whom?], and used for transmitting information to Allied armies.

Jeremiah Denton repeatedly blinked his eyes in Morse code during the 1966 televised press conference that he was forced into as an American prisoner-of-war by his North Vietnamese captors, spelling out "T-O-R-T-U-R-E". That confirmed for the first time to the US Naval Intelligence and other Americans that the North Vietnamese were torturing American prisoners-of-war.

In 1968, crew members of the USS Pueblo intelligence ship, held as prisoners by North Korea, communicated in sign language during staged photo opportunities, to inform the United States that they were not defectors but captives of the North Koreans. In other photos presented to the US, crew members gave "the finger" to the unsuspecting North Koreans, in an attempt to discredit photos that showed them smiling and comfortable.

In 1985, a klezmer saxophonist smuggled secrets into and out of the Soviet Union by coding them as pitches of musical notes in sheet music.

WORKING OF THE MODEL AND SCREENSHOTS

1.1 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 text message we **push a delimiter to the message string** as a checkpoint useful in decoding function. We encode data in order of Red, then Green and then Blue pixel for the entire message.

```python
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)
```

Image Steganography Encoding Algorithm

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
```

Image Steganography Decoding Algorithm

1.2 Text Steganography

Encode:

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
```

Text Steganography Encoding Algorithm

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

```python
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
```

Text Steganography Decoding Algorithm

1.3 Audio Steganography

Encode:-

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.

```python
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()
```

Audio Steganography Encoding Algorithm

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.

```python
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
```

Audio Steganography Decoding Algorithm

1.4 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.

```python
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
```

```python
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
```

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

```python
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
```

Video Steganography Encoding Algorithm

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

```python
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
```

```python
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
```

RESULTS AND OUTPUTS

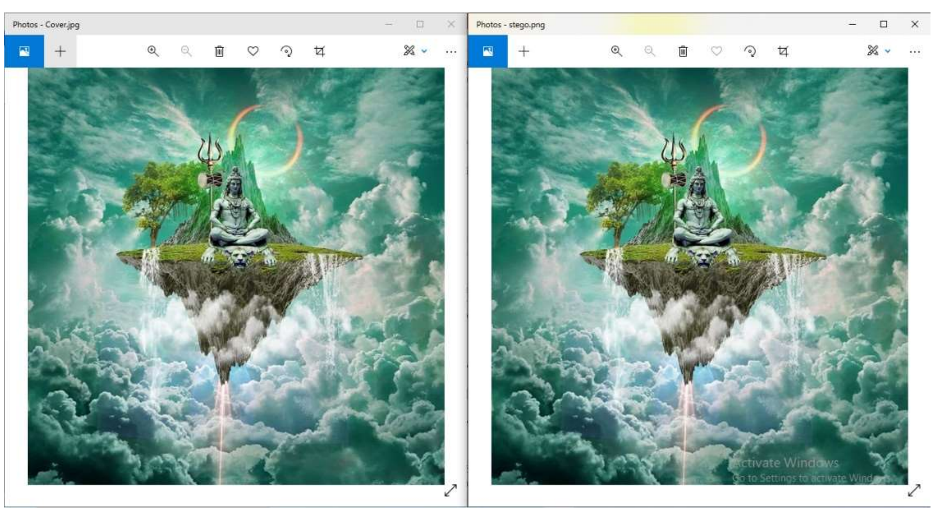
```
                STEGANOGRAPHY

                  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
```

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


0101010001101000011010010111001100100000011010010111001100100000011011110111010101110010001000000110110101101001011011100110111
1011100100010000001110000011100100110111101101010011001010110001101110100001000000110111101101110001000000111010001101000011001
0100100000011101000110111101110000011010010110001100100000001000100101001101010100010001010100011101000001010011100100111101000
1110101001001000001010100000100100001011001001000100010111000100000010100110100010101001101001000000011010100100000010001100111
0010011011110110110100100000011001100011000100100000011000100110000101110100011000110110100000100000001100010010111001010000011
1001001101001011110010110000101101110011100110110100000100000010100110110100001100001011100100110110101100001001000000011001000
1011100100101101101111011101010111001101110100011101010110001001101000001000000111001101101001011011100110100001100001001000000
0110011001011100101011001100001011010010110001001101000011000010111011000100000010010110110000101101110011100110110000101101100
0010101001011110001010100101111000101010

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:

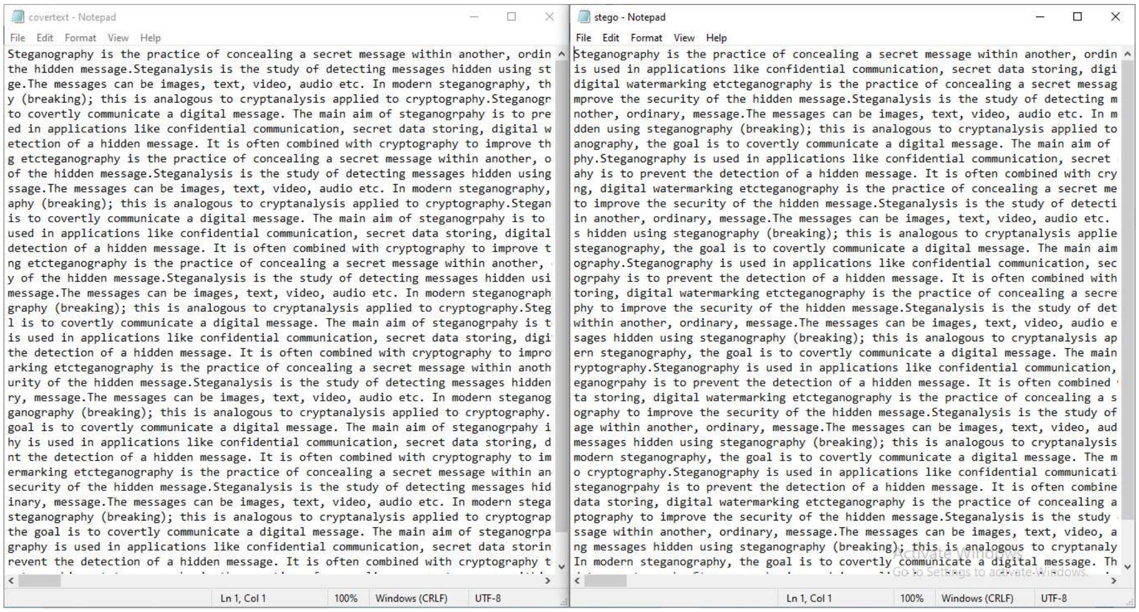
```
                IMAGE STEGANOGRAPHY OPERATIONS

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
```

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 :- 01101000111001101001001001101001001101101110100100111111
0100110100100110110111010010011111101001101001010101101110111011011101000001111110100110100101110110100100110110100101000110
1001010101101110100000111111010011011101010011011101000011010010101011010010000011010011111011010011001011011101110001111110
0011010010101011010010100001111110100110111011100110100100100110100111110011111101001101110111001101001010101101110101001101
0100110110100110010011111101000111111100001101000100101101000111001101011111101101011110101101011101101101011010001101011010
1101011110101101000100001101011101101101000101001101011001001101000001100111111100000111111010000111111010011010001001011010
1111011010110111001111111010001111001111001111111010011010111100011011101000011010010101011010010111001111111010011010011100
1110010110011111110100110100110000110100110110110111011100110100110010110100100100011111110100011100101100111111010001101000
1001101110100001101001001101101110001101101001101101101001010001101110100101101001001000111111101001101000100101101001001001
0011011011011101000011010010111011010011011001111111010001111001000001111110100011010110001011010010101011011101111011011101
0110111011100110111011110110100110000110100100100011111110100110111010010110100100110110100101000110100100100110100110110011
1101000111100100100111111010001101000110001101001101101101001001101101001100001101001001001101001101101101110110000111111010
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:

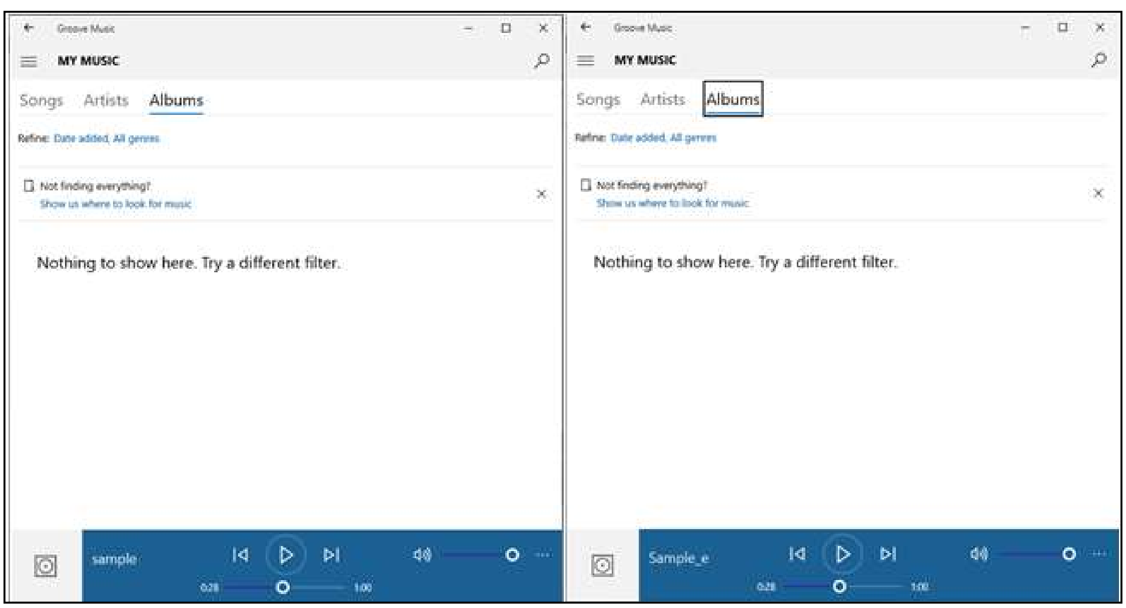
```
                TEXT STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:2

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

Encrypted message presented in code bits: 0110100011100110100100100110100100110110111010010011111110100110100100110110111010010
01111111010011010010101011011101111011011101000001111111010011010010111011010010011011010010100011010010101011011101000001111111
10100110111010100110111010000110100101010110100100000110100111110110100110010110111011100011111110100110100101010110100101000001
11111101001101110111001101001001001101001111100111111101001101110111001101001010101101110101001101001001101101001100100111111110
10001111111000011010001001011010001110011010111111011010111101011010111011011010110100011010110101011010111101011010001000011101
01110110110100010100110101100100110100000110011111110000011111101000011111110100110100010010110101111110110101101110011111111010
00111100111100111111101001101011110001101110100001101001010101101001011100111111101001101001110000111100101100111111110100110100
11000011010011011011011101110011010011001011010010010001111111010001111001011001111111010001101000101001101110100001101001001101
10111000110110100110110110100101000110111010010110100100100011111110100110100010010110100100100110100110110110111010000110100101
11101101001101100111111101000111100100000111111010001101011000101101001010101101110111101101110100101101110111001101110111110110
10011000011010010010001111111010011011101001011010010011011010010100011010010010011010011011001111111010001111001001001111111010
00110100011000110100110110110100100110110100110000110100100100110100110110110111011000011111110100110101100010110100110110110101
0101000110111010010110100110110110100100110

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.Pri
yansh Sharma 2.Koustubh sinha 3.Vaibhav Kansal
```


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 :- 01010100011010000110100101110011001000000110100101110011001000000110111101110101011100100
0100000011011010110100101101110011011110111001000100000011100000111001001101111011010100110010101100011011101000010000001101111
0110111000100000011101000110100001100101001000000111010001101111011100000110100101100011001000000010001001010011010101000100010
1010001110100000101001110010011110100011101010010010000010101000001001000010110010010001000101110001000000101001101000101010011
0100100000001101010010000001000110011100100110111101101101001000000110011000110001001000000110001001100001011101000110001101101
0000010000000110001001011100101000001110010011010010111100101100001011011100111001101101000001000000101001101101000011000010111
0010011011010110000100100000001100100010111001001011011011110111010101110011011101000111010101100010011010000010000001110011011
0100101101110011010000110000100100000001100110010111001010110011000010110100101100010011010000110000101110110001000000100101101
10000101101110011100110110000101101100

Length of binary after conversion :-  1016

Enter name of the stego file (with extension) :- stego.wav

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

- Preview of Cover Audio file and Stego Audio file:



- Decoding the message from Audio file:

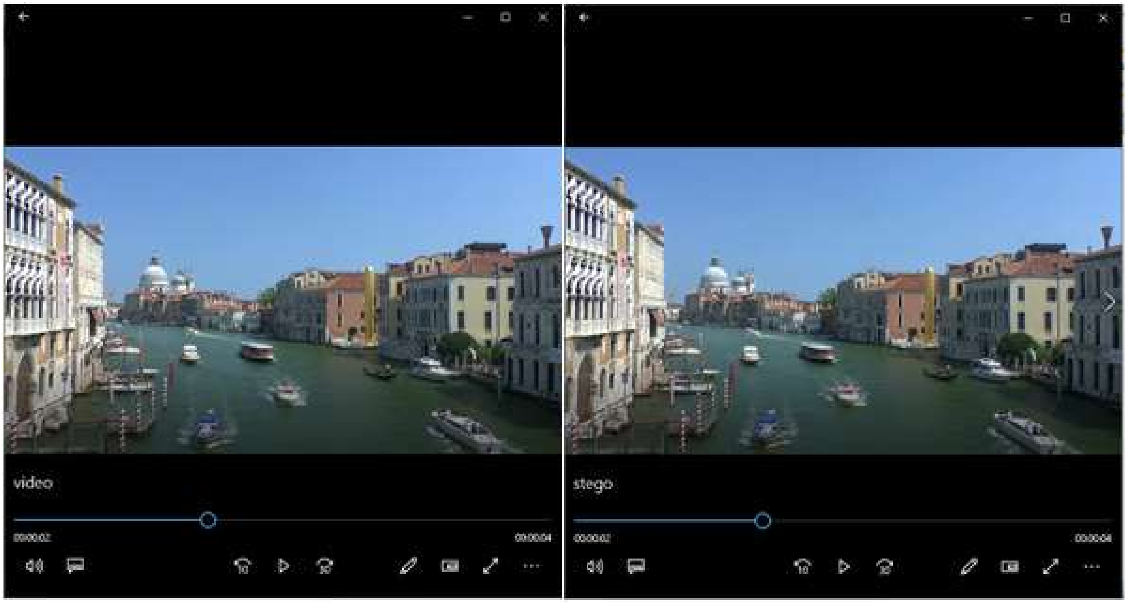
```
              AUDIO STEGANOGRAPHY OPERATIONS
1. Encode the Text message
2. Decode the Text message
3. Exit
Enter the Choice:2
Enter name of the file to be decoded :- stego.wav
The Encoded data was :-- This is our minor project on the topic "STEGANOGRAPHY". SEM 5 From f1 batch 1.Priyansh Sharma 2.Koustu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

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▯▯▯û4IÙþÀì▯▯`▯ºöb3iÄ¨|ho5û'¤ *u▯¶§▯yÌ▯Ì¶▯UºÙF
j▯÷▯▯Ö+TnÂã▯$9▯▯.▯¢Ü9▯M▯+Þ▯åZ&

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

- Preview of Cover Video file and Stego Video file:



16

- 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

As proved in the above section, the Steganographic model is successful.