A Progress Report

on

Video Steganography

carried out as part of the course CSE CS3270 Submitted by

Parth Lotte

209301461

VI-CSE

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering

Department of Computer Science & Engineering,

School of Computer Science and Engineering,

Manipal University Jaipur,

March 2023

INTRODUCTION

Steganography is the art of hiding the fact that communication is taking place, by hiding information in other information.

In this project it hides the message within the video file. In this project, the sender selects a cover file (video) with secret text and hide it into the cover file by using different efficient algorithm and generate a stego file of same format as our cover file (video). Then the stego file is sent to the destination with the help of private or public communication networks. On the other side i.e., receiver, the receiver downloads the stego file and by using the appropriate decoding algorithm retrieves the secret text that is hidden in the stego file.

In video steganography we have used combination of cryptography and 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.

MOTIVATION

Video steganography is an exciting field that offers endless possibilities for creative and practical applications. It involves the art of hiding secret messages within video files, making them virtually undetectable to the human eye.

For choosing this project there were many reasons. Firstly, video steganography has numerous practical applications we can use to protect sensitive information like passwords, data files and messages from hackers or any other suspicious entity.

Secondly, we can use it to hide any message, confidential info or any other data by hiding it in the video using steganography techniques .Also it can be used for the military operations.

Overall, a steganography project can be an exciting and interesting field to work on as it not only provides us to work on new technologies like cryptography, message hiding but also offer a chance to explore the technology in the field of security & privacy.

ARCHITECHURE

ALGORITHM DESIGN AND IMPLEMENTATION

In video steganography we have used combination of cryptography and 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 an 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
```

PSEUDOCODE

ALGORITHM USED:

- RC4 Encryption algorithm we used & it is has two parts:
- KSA (Key-Scheduling Algorithm)
- PRGA (Pseudo Random Generation Algorithm)

1. KSA (Key-Scheduling Algorithm)

S = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Initial Permutation

STEP – 1: ADD S[i] to j

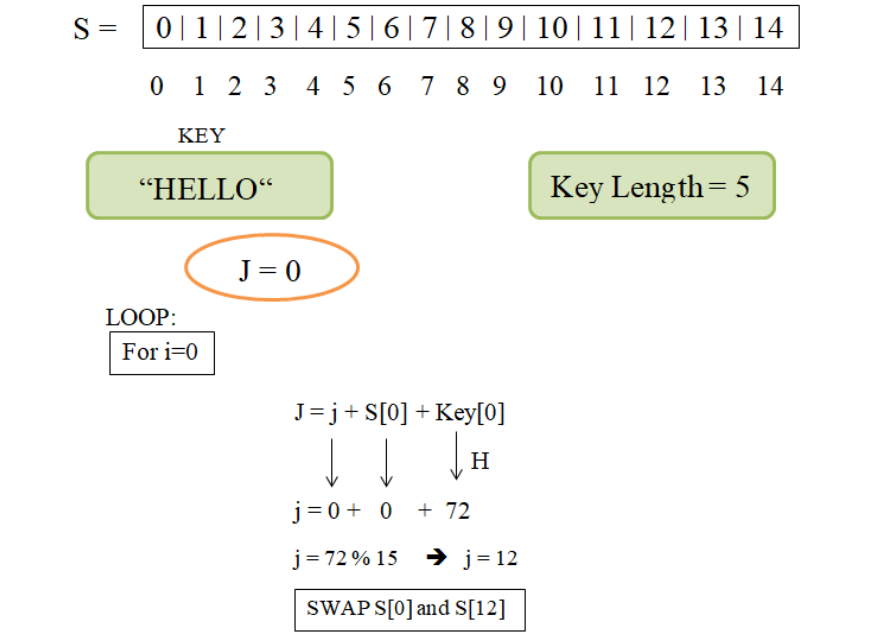
STEP – 2: ADD Key[i] to j

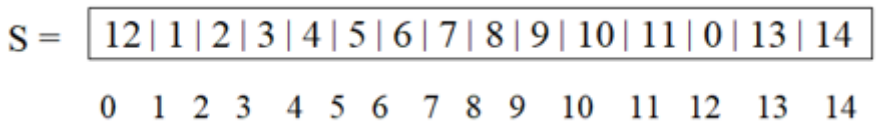
STEP – 3: SWAP S[i] and S[j]

- Initialize array S [] length 256 & entries are in ascending order.
- Ask user for key and convert it into its equivalent ASCII code.
- S [] is a permutation & j is assigned as:

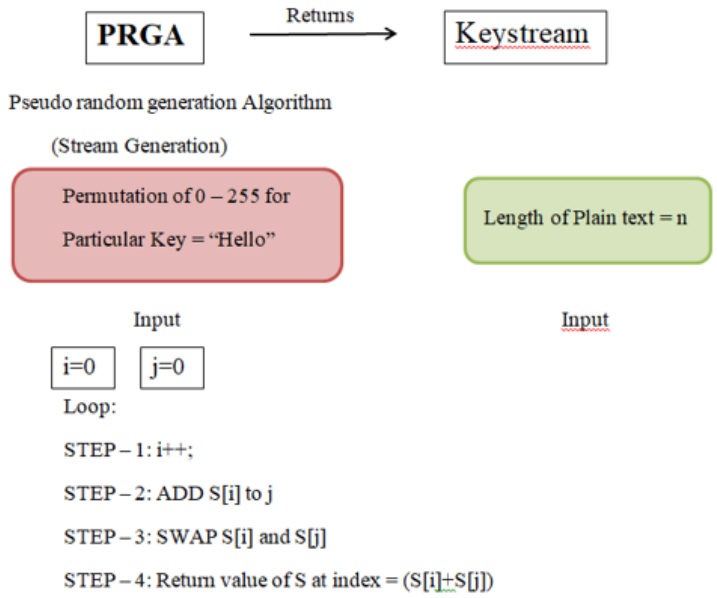
  j=j+S[i]+key[i%keylength] mod 256.

- Swap S[i] with S[j].
- New permutation generated for the whole keystream algorithm.



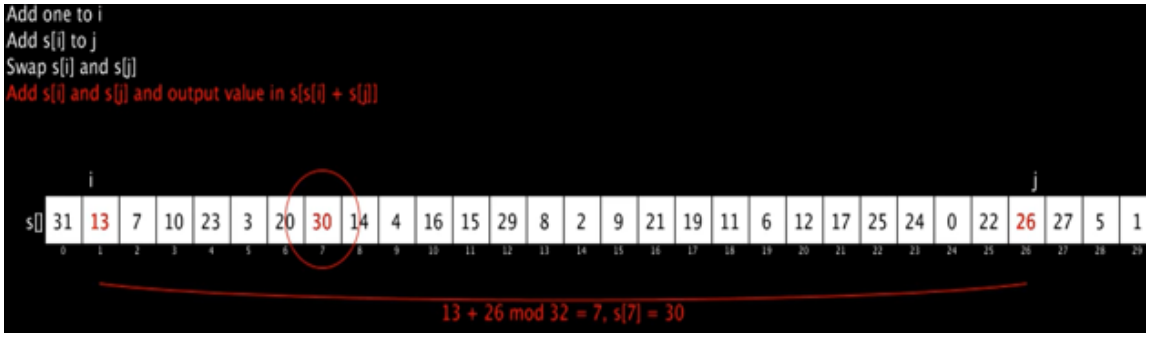


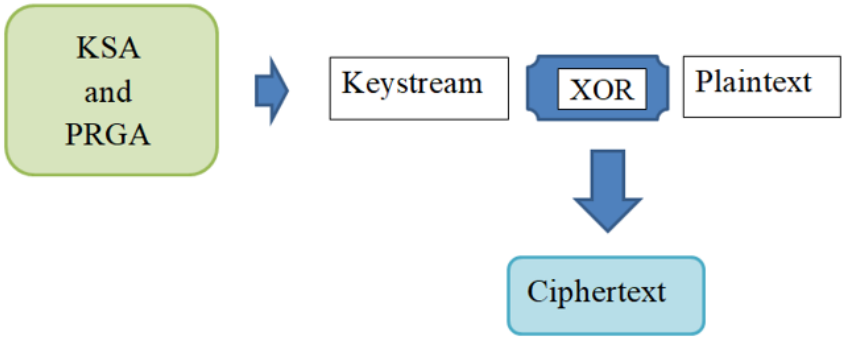
2. PRGA (Pseudo Random Generation Algorithm)



- Input length of Plain Text.
- Loop start to generate a keystream byte of equal length.

- Initialise i=0, j=0 increment i by 1 and mod 256.
- Now add S[i] to j and mod with 32.
- Swap the values.
- Store the keystream bytes which matches S[(S[i]+S[j]) mod 256] to get the keystream of length same as plain text.
- Now XOR the plain text with Keystream to get the cipher.





RESULTS AND OUTPUTS

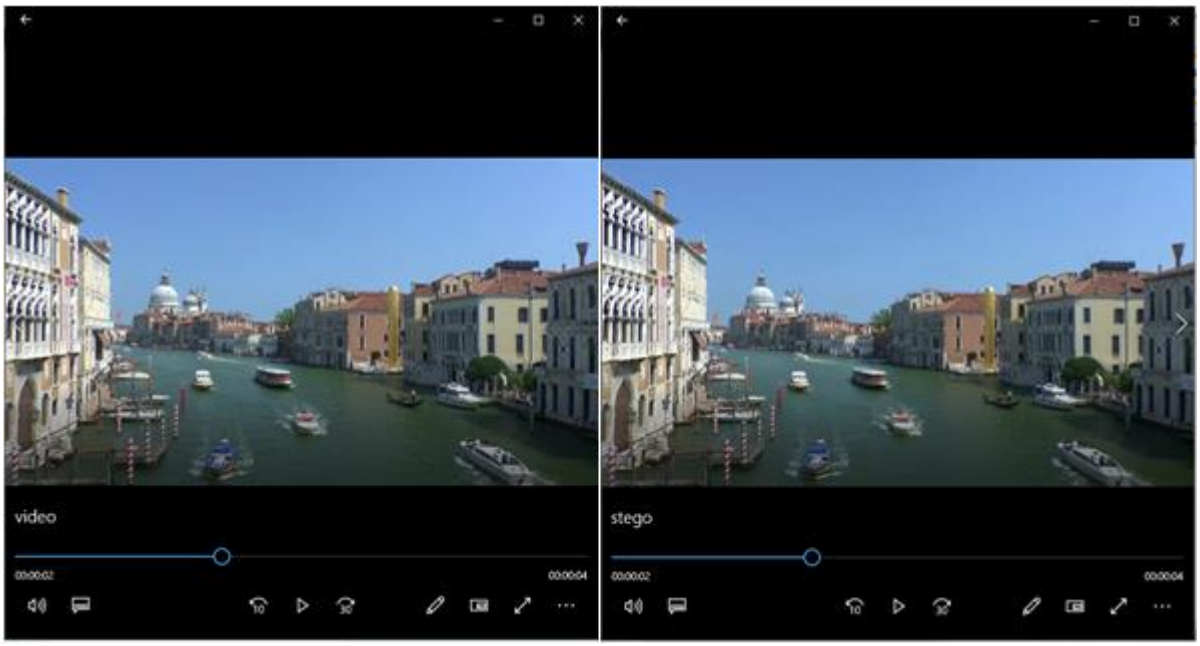
```
                    STEGANOGRAPHY USING VIDEO

                      MAIN MENU

1. VIDEO STEGANOGRAPHY {Hiding Text in Video cover file}
2. Exit

Enter the Choice:
```

ENCODING:

```
Enter the Choice: 1

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

```
Total number of Frame in selected Video : 172
Enter the frame number where you want to embed data :
122

Enter the data to be Encoded in Video :MY NAME IS PARTH LOTTE
Enter the key :
999
The encrypted data is :  ñÍ[ÐØVxó$ÕoiïW
◄t

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

Preview of Cover Video file and Stegno Video file:

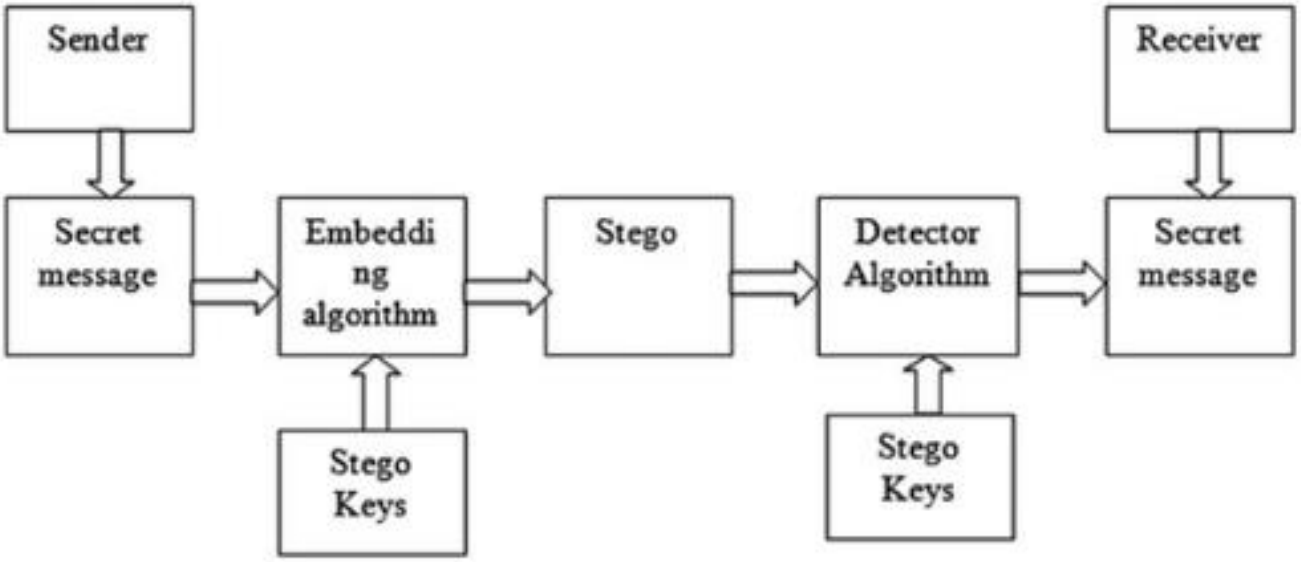


DECODING:

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

```
Total number of Frame in selected Video : 172
Enter the secret frame number from where you want to extract data
122
Enter the key :
999


The Encoded data which was hidden in the Video was :--
 MY NAME IS PARTH LOTTE
```

FLOWCHART


Sender
Receiver
Secret message
Embeddi ng algorithm
Stego
Detector Algorithm
Secret message
Stego Keys
Stego Keys

FUTURE SCOPE

As, Video steganography is a rapidly evolving field with significant potential for future advancements. With further development in this Project " Video Steganography", This Project Can be used by Indian army, RAW, Police and Intelligence agency for Special Emergency operation.

It can also be used in digital forensics to detect tampering and authentication of video content. In future research in this field will help in the development of more advanced and robust steganography algorithms.

We can also use to enhance security and privacy as the use of video steganography can be expanded to develop secure communication systems such as video messaging, which cannot be decode any unauthorized entity.