Indian Institute of Technology Bhubaneswar

Security & Forensics Lab II

Laboratory Experiment No. 2

(Held on 14/01/2022)

Submitted By:

Ayush Modi

21CS06005

M. Tech, CSE (1st year)

Aim

The aim of the experiment is to hide the message in an audio file and extract the hidden message from the encoded audio file.

Theory

Steganography is the technique of hiding data secretly within an ordinary, non-secret, file or message in order to avoid detection. The use of steganography can be combined with encryption as an extra step for hiding or protecting data. Steganography can be used to conceal almost any type of digital content, including text, image, video or audio content; the data to be hidden can be hidden inside almost any other type of digital content.

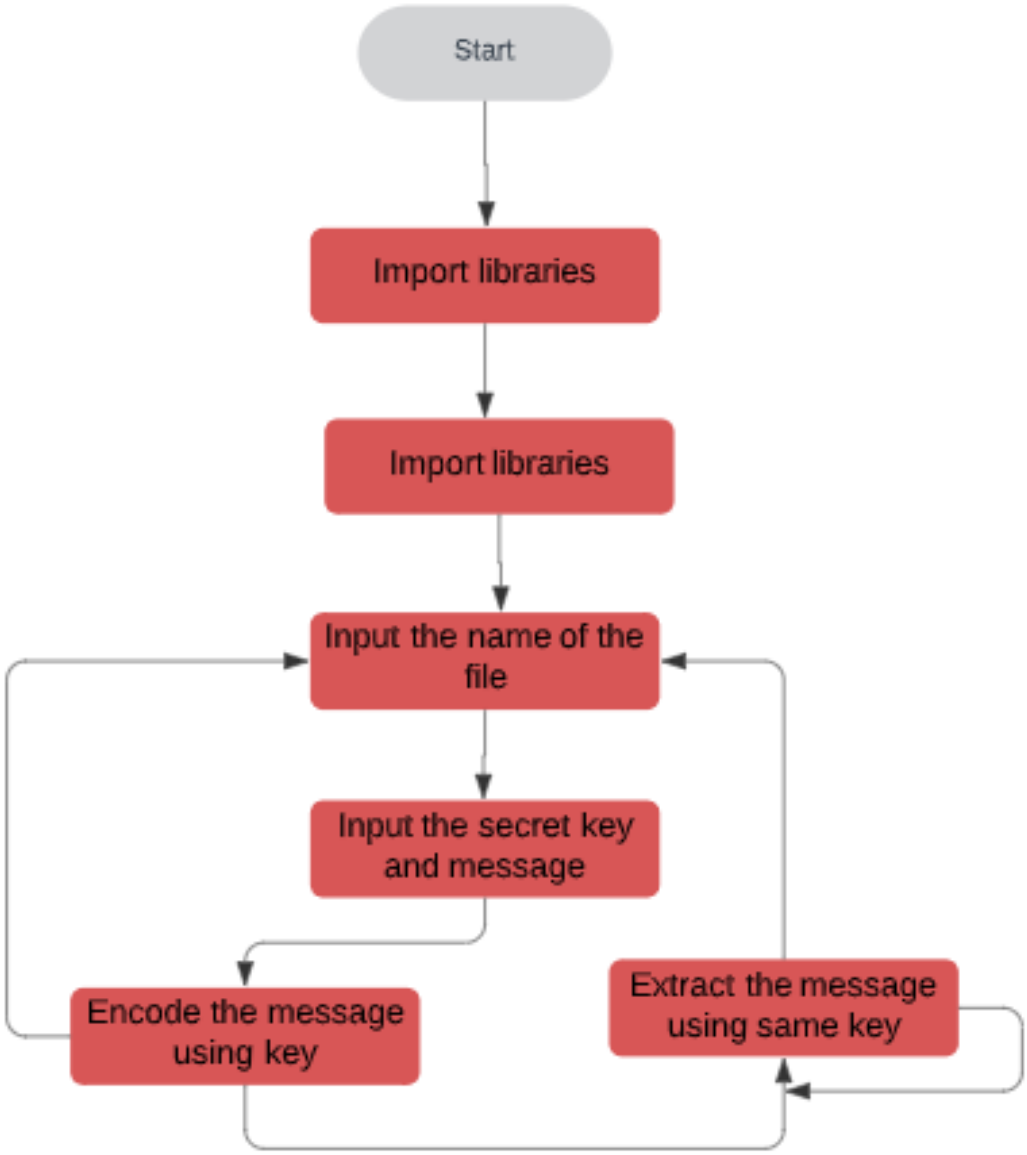
In modern digital steganography, data is often encrypted and then inserted, using a special algorithm, into data that is part of a particular file such as a JPEG image, audio, or video file. The secret message can be embedded into ordinary data files in many different ways. By applying the encrypted data to this redundant data in some inconspicuous way, the result will be an audio file that appears identical to the original audio but that has "noise" patterns of regular, unencrypted data.

Steganography with LSB –
LSB-Steganography is a steganography technique in which we hide messages inside an image/audio by replacing least significant bit of image/audio file with the bits of message to be hidden.

Inputs

1. **Filename -** which is to be used to hide the message**.**
2. **Key -** An alpha-numeric key is used to determine the location of the message bits.
3. **Message -** Alpha-numeric message (up to 5,00,000 characters).

Flow Chart


Start
Import libraries
Import libraries
Input the name of the file
Input the secret key and message
Encode the message using key
Extract the message using same key

Hiding the message

This is a symmetric key algorithm, i.e. the same key is used to hide the data and then extract the hidden data.

Every character of the message (including symbols and numbers) is converted to its 8-bit binary code using ASCII values. Similarly, every character of the key is converted to its 8-bit binary code using ASCII values.

For example Binary equivalent of ***Hello*** is:

01001000011001010110110001101100011011111

Now, we traverse the frames from left to right from left to right. While traversing, we check the value of the key at key index. If it's 1, we replace the LSB of the frame with the value of the message at message index. When the message is over, we stop traversing and give the output audio.

For example: Let's say we wish to hide the below data in an audio file.

Message - *Steganography is the art of concealing information. In computer science, it refers to hiding data within a message or file. It serves a similar purpose to cryptography, but instead of encrypting data, steganography simply hides it from the user.*
(Binary - 01000011……)

Key - *Hello (Binary -* **01001000011001010110110001101100011011111***)*

While traversing the frames, we ignore the frame as key[0] is 0 and we move to the next frame. Key[1] is 1. So, we put the message[0] bit (i.e. 0) in the LSB of message 219 (at audio[1]).

216 **219** 217 176 **177** 175 218 216 216
(Original Audio)
v
v
v

216 **218** 217 176 **177** 175 218 216 216
(Audio with message encoded)

Binary Representation:

11011000 **11011011** 11011001 10110000 **10110001** 10101111 11011010 11011000

(Original Audio)

v

v

v

11011000 1101101**0** 11011001 10110000 1011000**1** 10101111 11011010 11011000

(Audio with message encoded)

We continue putting the message bits in the audio frames until all the message is encoded.

Algorithm -

Convert the key and message into their corresponding binary representation.
Initialize message_index to 0
Initialize key_index to 0

```
for(frame from 0 to #Frames) {  // Loop over each frame of the audio file until all the
                                // message bits are encoded or till the last frame
   Increment key_index by 1
      If(key[key_index % key_length] == 1) {
        If(message_index < message_length) {
           LSB of the current frame is replaced with the message[message_index]
           Increment the message_index by 1
        }
        Else
            Exit        // All the message has been successfully hidden in the audio file.
      }
      Else
        Continue;     // Skip the current frame
}
```

Extracting the hidden message from the Audio -

This is a symmetric key algorithm, so the same key is required to extract the hidden data as the one which was used to encode the data in the audio.

Every character of the key is converted to its 8-bit binary code using ASCII values.

For example Binary equivalent of **Hello** is:

01001000011001010110110001101100011011111

Now, we traverse the frames from left to right. While traversing, we check the value of the key at key index. If it's 1, we append the LSB of the frame to the newChar. If the length of the newChar is 8, a new character is converted to an integer and then to Character. This character is added to the result.

For example: Let's say the output audio frames are as follows:

216 **218** 217 176 **177** 175

v
v
v

11011000 **11011010** 11011001 10110000 **10110001** 10101111

While traversing the audio, we ignore the first frame as key[0] is 0 and we move to the next frame. Key[1] is 1. So, the newChar is added with 0 as the LSB of the frame is **0**. Similarly, we add 1 to newChar - **01** for 177.

Algorithm -

Convert the key into its binary representation.
Initialize message_index to 0
Initialize key_index to 0
Initialize extracted_message to empty string // initially the message is empty
Initialize newFrame as 0
Initialize length_newFrame to 0

for(i from 0 to #Frames) { *// Loop over each frame of the audio file until all the // message bits are extracted along with the delimiter or till the last frame*

Increment key_index by 1
If(key[key_index % key_length] == 1) {
If(message_index < message_length) {
Multiply the newFrame with 10 and add the LSB of the current frame
Increment the message_index by 1
}
Else
Exit // Complete message has been successfully extracted.
}
Else
continue; // Skip the current frame
}

Results

- Encoding the message

Encode the message

```
[6]  1 # Input for the filename
     2 fileName = input("Enter the filename: (eg- cover_audio.wav)")
     3 # Input for the key to be used
     4 key = input("Enter the key:")
     5 # Input for the message to be encoded
     6 message = input("Enter the message to be hidden:")
     7
     8 # Function call to encode the message
     9 encodeMessage(fileName, key, message)
```

```
Enter the filename: (eg- cover_audio.wav)cover_audio.wav
Enter the key:akhand
Enter the message to be hidden:ajdhwhajwhdih adwjahgwjidhiuawhiudhuiawhdihauiwhdiuhawiuhdiuhawuihdiuhawiuhduiahwiudhuahwdhiuahwuidhiwahiudhaiwhdihaiuwhduiahwuidhu
```

- Extracting the message

Extract the message

```
1 # Input for the key to be used
2 key = input("Enter the same key which was used to encode the message:")
3
4 # Function call to extract the message
5 extractedMessage, lastIndex = extractMessage(key)
6
7 print('extractedMessage = {}'.format(extractedMessage))
```

```
Enter the same key which was used to encode the message:akhand
extractedMessage = ajdhwhajwhdih adwjahgwjidhiuawhiudhuiawhdihauiwhdiuhawiuhdiuhawuihdiuhawiuhduiahwiudhuahwdhiuahwuidhiwahiudhaiwhdihaiuwhduiahwuidhuiawhdoiawoi
```

- Plots

Below are the plots of the sounds for each frame. The last plot shows the difference between the two sounds.

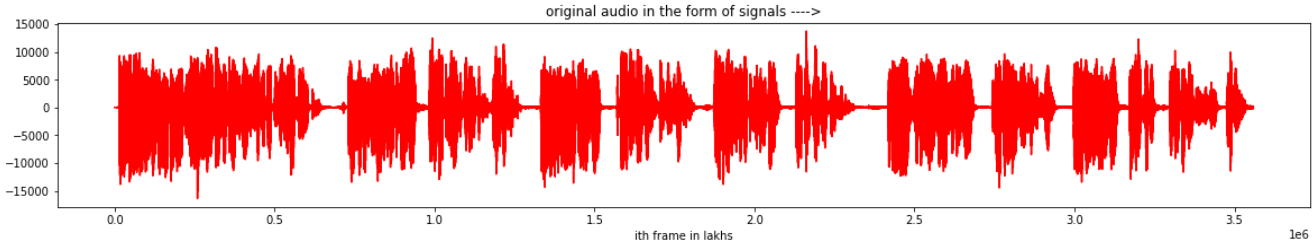


Fig-1: Frame-wise sound of the original audio

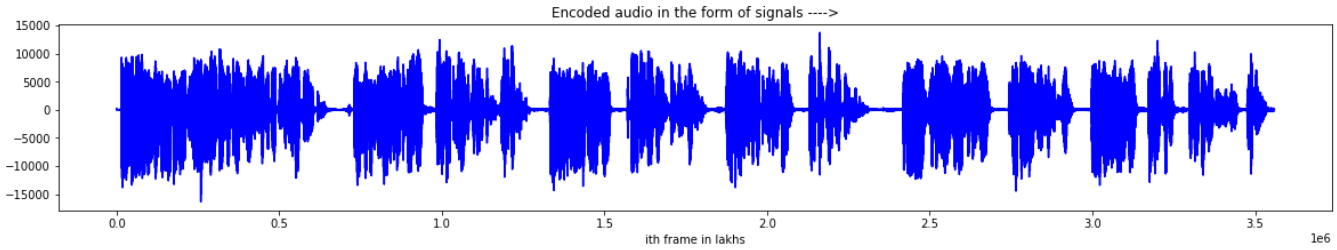


Fig-2: Frame-wise sound of the encoded audio

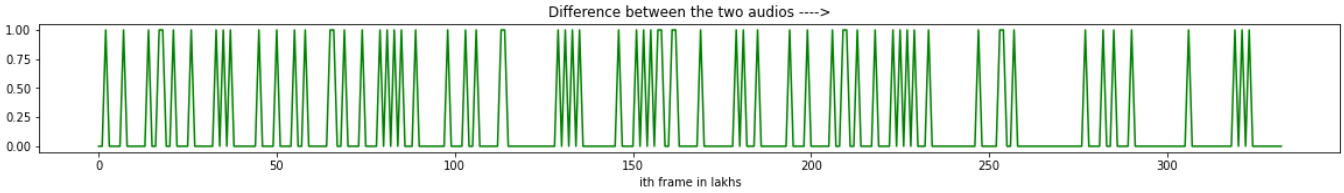


Fig-1: Frame-wise difference between the two sounds

Conclusion

The message was hidden in the audio. Then using the same key, data was extracted from the encoded audio file.

Code

Google drive link -
https://drive.google.com/drive/folders/1Ww97VY8V7zR6tdMjOpLZWqHGCk5Za6mr?usp=sharing

Github link to the file –
https://github.com/amyush/Audio-steganography.git