INFORMATION SECURITY PROJECT

Dual-Layer Security of Data using LSB Image Steganography Method and AES Encryption Algorithm

Group Members:

Vatsal Gupta 17104060
Arjav Jain 17104070
Ananya Sharma 17104038

Title

Dual-Layer Security of Data using LSB Image Steganography Method and AES Encryption Algorithm

Introduction

Cryptography concerns itself with hiding the contents of a message. In contrast, steganography concerns itself with hiding the fact that there is even a message at all, which discourages potential code-breakers from looking deeper.

Steganography is the art of concealing information within different types of media objects such as images or audio files, in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message. By default, steganography is a type of security through obscurity.

In our project, we propose a python steganography module to store messages or binary files inside an image. We have further increased the security using an AES-256 encryption algorithm, a popular algorithm in symmetric key cryptography

Working Principle

The technique used in this project works on the principle that the human eye cannot differentiate among pixels unless the difference in the RGB values of the pixels is significant.

One byte of data consists of 8 bits - eight digits of either 0 or 1, e.g. 00011010. If this byte represents the colour of a pixel in an image, we could change the rightmost digit and only very imperceptibly change the colour of that pixel.

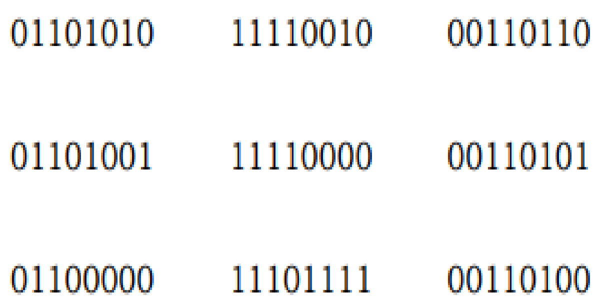
Hence the data which is to be encrypted can be encoded in the least significant bits of a pixel of an image, so the overall image does not change significantly, and hence the message can be transferred undetected. The least significant bits can be extracted later for decryption.

Further, AES is used to encrypt/decrypt the data before using LSB steganography for enhanced security.

Algorithms Used

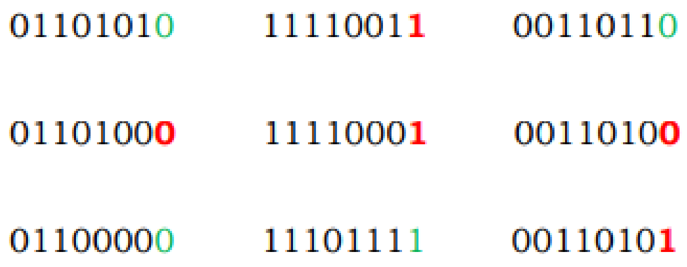
- **Least Significant Bit Algorithm:**

  Steganography for encrypting text and image into an image that uses a modified version of LSB (Least Significant Bit) algorithm. In this technique, the binary representations of the secret data have been taken, and the LSB of each byte is overwritten within the image. If 24-bit colour images are used, then the quantity of modification will be small. As an example, suppose that we have three neighbouring pixels (nine bytes) with the following RGB encoding:



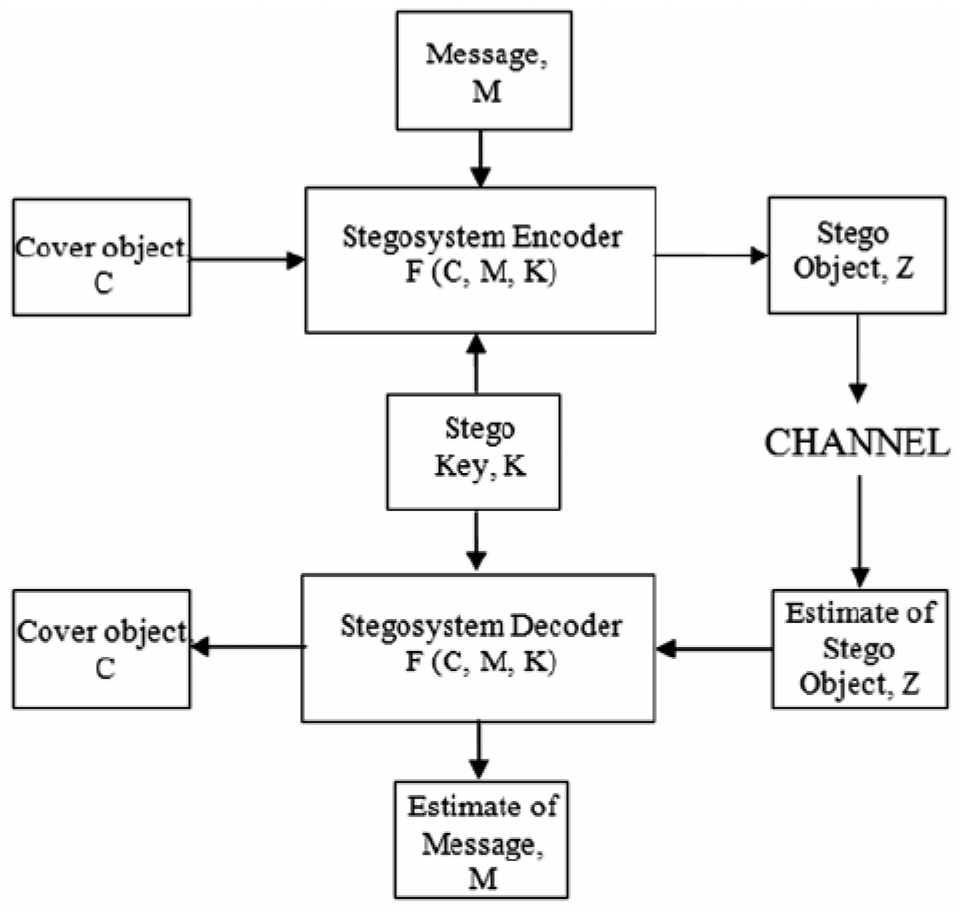
01101010	11110010	00110110
01101001	11110000	00110101
01100000	11101111	00110100

Now if we wish to embed the following 9 bits of compressed secret information: 010010011. If we insert these 9 bits over the LSB of the 9 bytes above, we get the following sequence of bits (where bits in red colour have been modified):



01101010	1111001**1**	00110110
0110100**0**	1111000**1**	0011010**0**
01100000	11101111	0011010**1**

We have successfully hidden 9 bits but at the cost of only modifying 5, or roughly 50% of the LSB bits. Now, for extracting the embedded message from the image, we calculate the pixels of the image and loop through the pixels of the image until we find the 8 consecutive zero. Then Pick up the LSB from each pixel element and then convert it into the character.



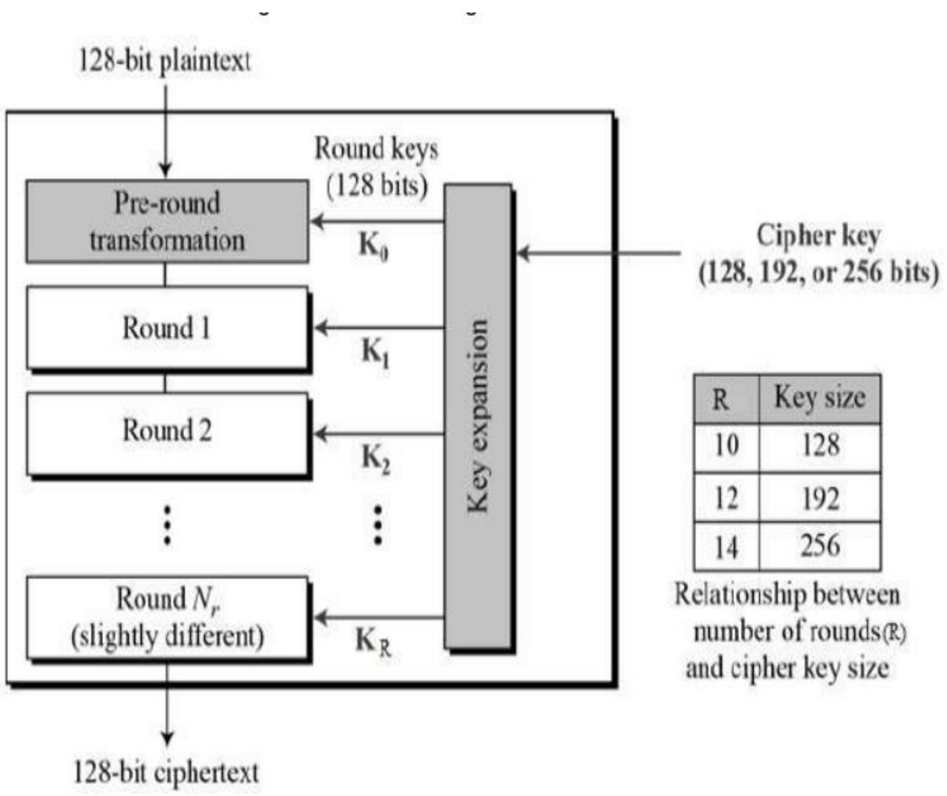
- **<u>Advanced Encryption Standard Algorithm:</u>**

  AES makes use of an iterative approach rather than the Feistel cypher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

  Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

  Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

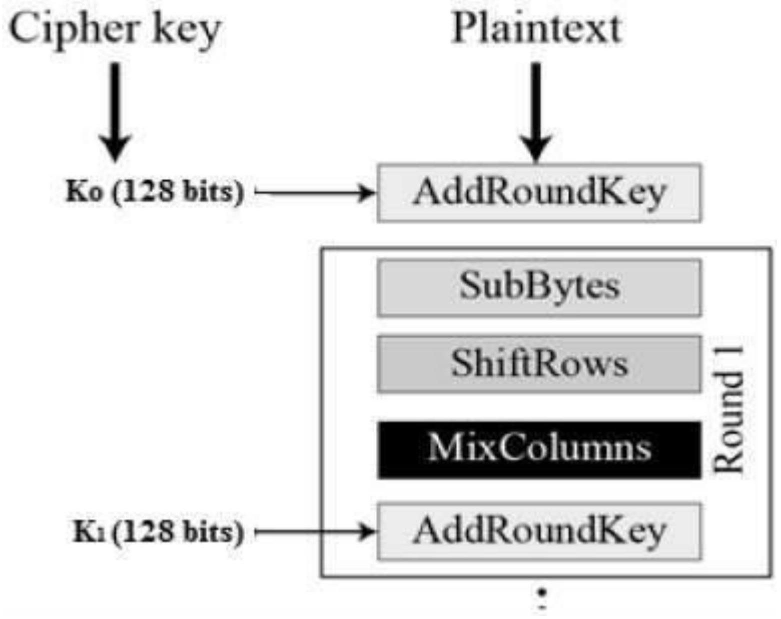
The schematic diagram of the AES algorithm's working is given in the following illustration −


128-bit plaintext
Round keys (128 bits)
Pre-round transformation
K0
Cipher key (128, 192, or 256 bits)
Round 1
K1
Round 2
K2
Key expansion
Round Nr (slightly different)
KR
128-bit ciphertext
Relationship between number of rounds(R) and cipher key size

R	Key size
10	128
12	192
14	256

(A) Encryption Process

Here, we restrict to the description of a typical round of AES encryption. Each round comprises of four sub-processes. The first round process is depicted below −



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of the row. The shift is carried out as follows −

- The first row is not shifted.
- The second row is shifted one (byte) position to the left.
- The third row is shifted two positions to the left.
- The fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

Each column of four bytes is now transformed using a particular mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round, then the output is the cypher text. Otherwise, the resulting 128 bits are interpreted as 16 bytes, and we begin another similar round.

<u>(B) Decryption Process:</u>

The process of decryption of an AES cypher text is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns

- Shift rows
- Byte substitution

Since sub-processes in each round are in a reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms need to be separately implemented, although they are very closely related.

Languages and Environment Used:

Python (3.7) in Spyder IDE

Project Description

In our project, we aimed at secure delivery of the data from the sender side to the receiver side without compromising on the quality of the data.

We have performed encryption of a binary message using AES algorithm and sent it across via a carrier medium by inserting it in it using LSB steganography

We have used the carrier medium -a jpg image.

The various functionalities of our project and their respective uses are described below:

1. CryptoSteganography class

This class which contains the necessary functions for hiding the message (def hide()) and retrieving the message (def retrieve()). These functions implement the AES and LSB algorithm in the required order. We have imported the necessary modules for them.

2. Hiding a message string in an image

This program takes input from the user for the message to be encrypted and hide it in an image(carrier) using the hide function from the cryptoSteganography class. After encryption, a new file is created which contains the carrier image having the encrypted message in it. For the decryption, the user is asked for a password to decrypt the file, and maximum two trials are given for successful entry of the password, after that the stored message is displayed to the user using retrieve function from the above class.

3. Hiding an MP3 file in an image

This program has the same functioning as the one above; the difference here is that the message to be encrypted is an mp3 file.

The cryptoSteganoraphy class is common to both the programs.

Project Motivation and Papers Referred:

Steganography is a suitable method for embedding sensitive information behind some cover media. LSB based steganography in combination with AES will provide a good security model for hiding data. AES, which is a relatively new encryption technique, is preferred over DES due to its simplicity and its speed.

1.Enhanced Security of Data using Image Steganography and AES Encryption Technique

Panghal, S., Kumar, S., & Kumar, N. (2016). Enhanced security of data using image steganography and AES encryption technique. *International Journal of Computer Applications*, *42*.

2.Image Steganography Using AES Encryption and Least Significant Nibble

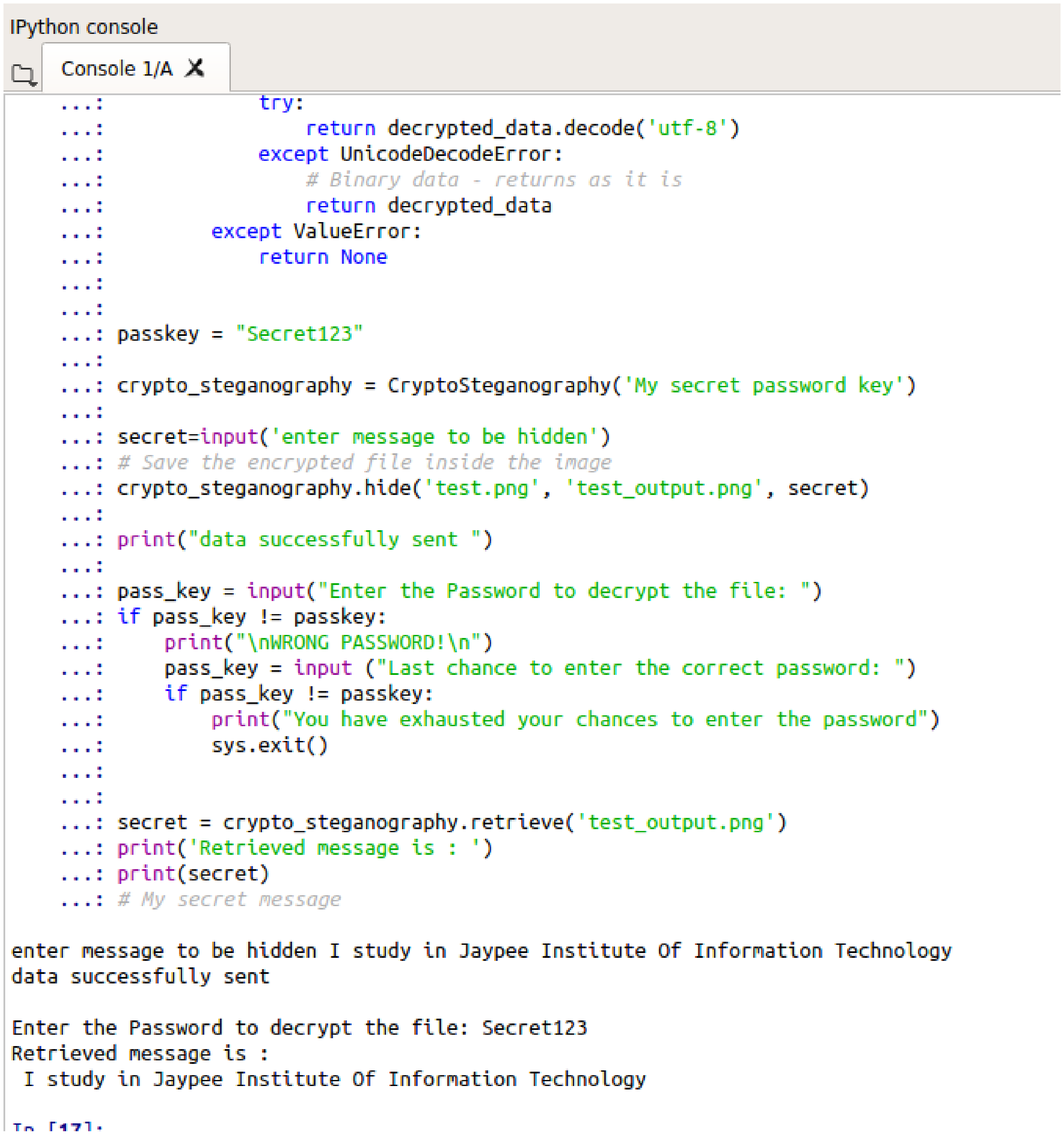
Sheth, U., & Saxena, S. (2016, April). Image steganography using AES encryption and least significant nibble. In *2016 International Conference on Communication and Signal Processing (ICCSP)* (pp. 0876-0879). IEEE.

3.The Process of Encoding and Decoding of Image Steganography using LSB Algorithm

Reddy, R. C., & Ramani, R. A. (2012). The Process of Encoding and Decoding of Image Steganography using LSB Algorithm. *International Journal of Computer Science Engineering & Technology*, *2*(11).

Screenshots of the running program

1. To store a message string inside an image.py



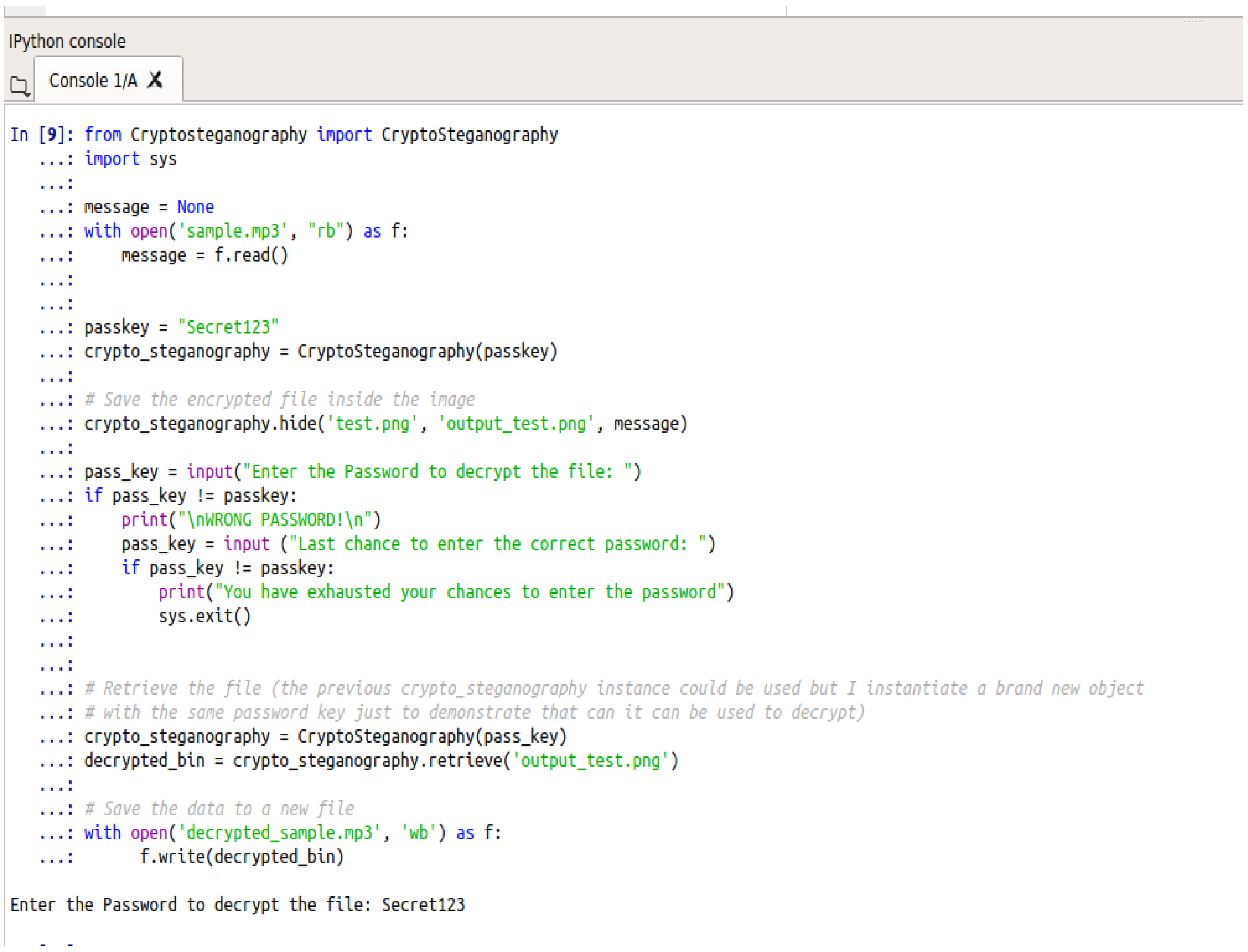
```
   ...:             try:
   ...:                 return decrypted_data.decode('utf-8')
   ...:             except UnicodeDecodeError:
   ...:                 # Binary data - returns as it is
   ...:                 return decrypted_data
   ...:         except ValueError:
   ...:             return None
   ...:
   ...:
   ...: passkey = "Secret123"
   ...:
   ...: crypto_steganography = CryptoSteganography('My secret password key')
   ...:
   ...: secret=input('enter message to be hidden')
   ...: # Save the encrypted file inside the image
   ...: crypto_steganography.hide('test.png', 'test_output.png', secret)
   ...:
   ...: print("data successfully sent ")
   ...:
   ...: pass_key = input("Enter the Password to decrypt the file: ")
   ...: if pass_key != passkey:
   ...:     print("\nWRONG PASSWORD!\n")
   ...:     pass_key = input ("Last chance to enter the correct password: ")
   ...:     if pass_key != passkey:
   ...:         print("You have exhausted your chances to enter the password")
   ...:         sys.exit()
   ...:
   ...:
   ...: secret = crypto_steganography.retrieve('test_output.png')
   ...: print('Retrieved message is : ')
   ...: print(secret)
   ...: # My secret message

enter message to be hidden I study in Jaypee Institute Of Information Technology
data successfully sent

Enter the Password to decrypt the file: Secret123
Retrieved message is :
 I study in Jaypee Institute Of Information Technology

In [17]:
```

2. To store a binary file inside an mp3 file



IPython console

Console 1/A

```
In [9]: from Cryptosteganography import CryptoSteganography
   ...: import sys
   ...:
   ...: message = None
   ...: with open('sample.mp3', "rb") as f:
   ...:     message = f.read()
   ...:
   ...:
   ...: passkey = "Secret123"
   ...: crypto_steganography = CryptoSteganography(passkey)
   ...:
   ...: # Save the encrypted file inside the image
   ...: crypto_steganography.hide('test.png', 'output_test.png', message)
   ...:
   ...: pass_key = input("Enter the Password to decrypt the file: ")
   ...: if pass_key != passkey:
   ...:     print("\nWRONG PASSWORD!\n")
   ...:     pass_key = input ("Last chance to enter the correct password: ")
   ...:     if pass_key != passkey:
   ...:         print("You have exhausted your chances to enter the password")
   ...:         sys.exit()
   ...:
   ...:
   ...: # Retrieve the file (the previous crypto_steganography instance could be used but I instantiate a brand new object
   ...: # with the same password key just to demonstrate that can it can be used to decrypt)
   ...: crypto_steganography = CryptoSteganography(pass_key)
   ...: decrypted_bin = crypto_steganography.retrieve('output_test.png')
   ...:
   ...: # Save the data to a new file
   ...: with open('decrypted_sample.mp3', 'wb') as f:
   ...:       f.write(decrypted_bin)

Enter the Password to decrypt the file: Secret123
```