

## **CRPTOGRAPHY:**

Cryptography is basically a method in which you will be encrypting a message that a user wants to send it to someone. This method allows us to form an encrypted text (**cipher text**) that hides the message.

The working of cryptography is done via following steps:

Firstly, the textual confidential data which the user wants to send is converted to numerical form.

Secondly, this numeric form that has the message, is applied into the algorithm along with the encryption key, which we can say, is like a string of bits.

Together, when applied into the encryption algorithm, it forms **cipher-text**. The cipher-text that is generated is then sent to the receiver through the network.

Reaching the other side, the receiver then decrypts the cipher-text in a similar way. Here, he uses the cipher-text and decryption key as the inputs for the decryption algorithm and decrypts it back to the numeric form.

Finally in the end the numeric form is converted to the intended message.

There are two types of algorithms in Cryptography.

**Symmetric Algorithms:** The algorithms that use the same key for both encryption and decryption. This means the sender and the receiver have the same type of key that is used to encrypt and decrypt the text. It is known for its speed.

**Asymmetric Algorithms:** These are the ones that have different keys. This simply means that both sender and receiver will be having different keys for encrypting and decrypting the text. It is known for more security.

## **STEGANOGRAPHY:**

Steganography is a method in which you hide the message intended to send to someone in an image. The image might look the same after using steganography but a point to remember is that it has the message. Steganography hides the identity of the message in the cover.

The confidential data is taken as input from the user along with the cover image in which the data is to be hidden.

These act as an input for the Steganographic Encoder which encodes the given message into the image file, resulting the output as the **STEGO-IMAGE**.

Now the receiver is provided with stego-image which he/she decodes it with the Steganographic Decoder and finally gets the intended message.

## **PROPOSED SYSTEM:**

### **i) Overview:**

A GUI is developed by a secure communication system using concepts of cryptography and steganography with three objectives in mind:

1. Developing a method to encode data into a cipher text.
2. Developing a method to encode data into a cover image and decode the data from the stego-image.
3. Combining these both to form a secured communication system.

### **ii) Algorithms used:**

The proposed system deals with the cryptographic and steganographic algorithms.

In our system, for symmetric cryptography technique we used **RSA** algorithm, and for asymmetric cryptography we used **AES** algorithm.

And coming for steganography we used the **LSB technique**.

### **AES ALGORITHM:**

It is a symmetric algorithm. Where the key is the same for both the sender and the receiver. It works on **block cipher technique** which means that size of the plain text and cipher text must be the same. Here an input key is given into the algorithm which is of the same size as the plain text.

The working of AES algorithm is as simple as stated. There are basically, three types of bits that support this algorithm: **128 bits, 192 bits and 256 bits**. It states that the algorithm has multiple rounds which are for processing the key lengths. The number of rounds depends on the key size being used. A 128-bit key size dictates ten rounds, a 192-bit key size dictates 12 rounds, and a 256-bit key size has 14 rounds.

Steps in each round are:

- Substitution of the bytes: the bytes of the block text are substituted based on rules dictated by predefined S-boxes
- Shifting the rows: all rows except the first are shifted by one
- Mixing the columns: the Hill cipher is used to jumble up the message more by mixing the block's columns.
- Adding the key round: the message is XORed with the respective round key.

The model mentioned above just inverses itself when decrypting.

## **RSA ALGORITHM:**

RSA is a block-cipher type algorithm that converts plain text to cipher text. The **RSA algorithm** is an asymmetric cryptography algorithm; this means that it uses a *public* key and a *private* key. As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.

The RSA algorithm has steps that are described as below. The steps mentioned below highlights the working of RSA

- Selection of two prime numbers **p** and **q** where p not equal to q.
- Calculating  $n = p * q$ . (where n is the **block size**)
- Calculating  $\phi(n) = (p-1)*(q-1)$ .
- Selecting **e** such that, e is respectively prime to  $\phi(n)$ .; which means,  $(e, \phi(n)) = 1$  and  $1 < e < \phi(n)$ . (Where e is the **Encryption Key**)
- Calculate  $d = e^{-1} \bmod \phi(n)$  or  $ed = 1 \bmod \phi(n)$ . (where d is the **Decryption Key**)
- **Public Key** = {e, n}; **Private Key** = {d, n}.
- Find out **cipher-text** using the formula,  $C = P^e \bmod n$ , where  $P < n$ . (where P is the **Plain Text**)
- $P = C^d \bmod n$ . This is how we obtain the **Plain Text**.

## **LSB TECHNIQUE ALGORITHM**

In LSB steganography technique, the information hider embeds the secret information in the least significant bits of a media file. In an image file each pixel is comprised of three bytes of data corresponding to the colours red, green, and blue. The pixel value is converted to binary.

The confidential data is converted to its binary format and a temporary variable is assigned to null. If the message bit and LSB of the pixels are found similar then temporary variable is assigned to 0 and if different it is set to 1.

The setting of temp is done by considering XOR of message bit and the LSB of the pixel by updating the pixel of output image to input image pixel value and summation of the temporary variable.

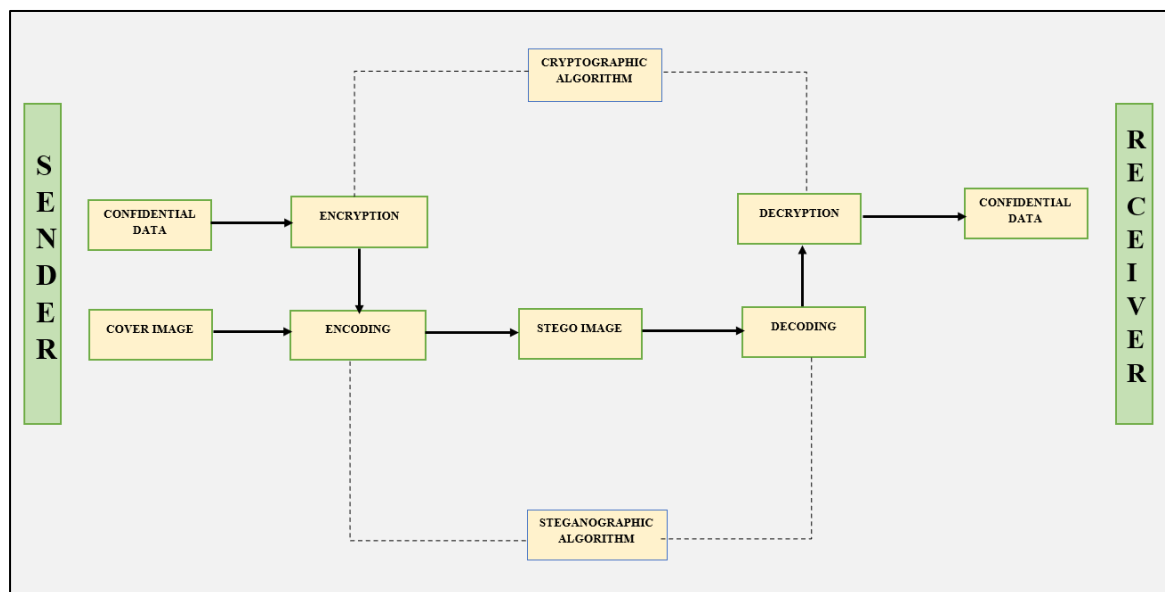
### **iii) System Architecture**

The main architecture of the process for communication between 2 parties is as follows

1. The sender selects a cover image for encoding the message in it.

2. The sender encrypts the confidential data with suitable cryptographic algorithms and encodes the encrypted data in the cover image with a public key using LSB technique and transmits it to the receiver on an unreliable channel.
3. The receiver receives the stego-image through email and decodes the encrypted text from it using the private key.
4. The encrypted text is decrypted using the same cryptographic algorithm and confidential message is obtained.

## ARCHITECTURE DIAGRAM



### iv) Functional Architecture and Module Description:

There are 4 modules used in the model:

1. **Encryption Module:** This module encrypts plain text to cipher text. In this model we have used the AES and RSA encryption algorithm.
2. **Encode Module:** This module is responsible for hiding the cipher text to the cover image by the use of steganography techniques.
3. **Decode Module:** This module extracts the hidden cipher text from the stego image which is sent by the sender.
4. **Decryption Module:** This module performs AES/RSA decryption by using the private key and extracts the plain text from the cipher text.
5. **SMTP:** The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

### v) Innovative idea

The unique approach of combining cryptography and steganography to come up with an even more secure way of data exchange is the innovation behind this application.

Security is divided into layers such that intruders need to pass all layers to get inside the system. Therefore, more layers mean more protection. The GUI is user friendly with a human centric approach to design so that non-tech savvy users can also use this secure way of data communication easily.

## IMPLEMENTAION:

The implementation has been done majorly in two phases here that is cryptography implementation and steganography implementation. These two major phases have more sub-phases in them for implementation.

A GUI is developed using Python PySimpleGUI. which includes four layouts namely: Encrypt, Decrypt, Keys, Mail.

For the implementation of cryptography, firstly we need to generate the keys to proceed further. So, the first part of the implementation is the GENERATING OF KEYS. In the KEY layout the user is asked for selecting the key size and output folder. It has a button Generate keys, which on clicking generates the pair of keys to the output folder selected.

The key generation code is implemented by the following code. The key is generated using the function .generate() embedded in the Crypto.Cipher and Crypto.PublicKey package.

```
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.PublicKey import RSA

def GenerateKeyPair(dir = '', size = 3072):
    keyPair = RSA.generate(size)
    privateKey = keyPair.export_key()
    with open(dir + '/' + str(size) + '_private.pem', 'wb') as outputFile:
        outputFile.write(privateKey)

    publicKey = keyPair.publickey().export_key()
    with open(dir + '/' + str(size) + '_public.pem', 'wb') as outputFile:
        outputFile.write(publicKey)
```

The input is also taken as an image to encode the text into this cover image. The image can be of the format jpg. The image is then passed through following pre-processing steps using PIL (Python Imaging library)

- Image reading
- Converting into binary

```

image = Image.open(imagePath, 'r')
def GenerateData(data, bin = False):
    newData = []
    for i in data:
        newData.append(format(ord(i), '08b'))
    return newData

```

Once the keys are generated, we can proceed for the encryption process. Coming to layout Encrypt, the user is given various choices for encrypting, the choices include:

- to perform only symmetric cryptography
- to perform only asymmetric cryptography
- to perform symmetric cryptography and steganography
- to perform asymmetric cryptography and steganography
- to perform only steganography

The user is then asked to select the output folder, the public key and the image file accordingly to the choice of encryption he wants to do. The user is even given a choice how to encrypt the data, whether he wants to go for text data or file data. After filling all the choices, we can click the Encrypt button which encrypts the data and generates the encrypted file in the selected output folder and gives a confirmation if it's done or not. The code implementation of the cryptography and steganography encryption is stated below.

```

#CODE FOR ENCRYPTION USING AES
def EncryptAES(data, publicKeyPath, outputFilePath = None, header = None, size = 16):
    publicKey = RSA.import_key(open(publicKeyPath).read())
    sessionKey = get_random_bytes(size)
    sessionKeyEncrypted = EncryptRSA(sessionKey, publicKey)
    encryptor = AES.new(sessionKey, AES.MODE_GCM)
    if type(header) == str:
        headerLength = str(len(header))
        if len(headerLength) == 1:
            headerLength = '0' + headerLength
        header = bytes(headerLength + header, 'utf-8')
    elif header is None:
        header = bytes('00', 'utf-8')
    encryptor.update(header)
    if type(data) == str:
        ciphertext, tag = encryptor.encrypt_and_digest(bytes(data, 'utf-8'))

```

```
#CODE FOR ENCRYPTION USING RSA

def EncryptRSA(data, publicKey):

    encryptor = PKCS1_OAEP.new(publicKey)

    if type(data) == str:

        encrypted = encryptor.encrypt(bytes(data, 'utf-8'))

    elif type(data) == bytes:

        encrypted = encryptor.encrypt(data)

    return encrypted
```

```
#CODE FOR ENCODING THE DATA IN IMAGE/ STEGANOGRAPHY

def Encode(imagePath, data, output):

    image = Image.open(imagePath, 'r')

    newImage = image.copy()

    encode_enc(newImage, data)

    newImage.save(output, 'JPG')
```

Coming to layout Decrypt, the user is given various choices for decrypting, the user is asked for the encrypted file, the key and the desired output folder. It also has a textbox where the decrypted data and the confirmation if it is decrypted or not is displayed once we click the decrypt button.

```
#CODE FOR DECRYPTION USING AES

def DecryptAES(inputFilePath, privateKeyPath, outputFilePath = None):

    privateKey = RSA.import_key(open(privateKeyPath).read())

    inputFile = open(inputFilePath, 'rb')

    headerLength = int(inputFile.read(2))

    inputFile.seek(0)

    header, sessionKeyEncrypted, nonce, tag, ciphertext = [inputFile.read(x) for
x in (2 + headerLength, privateKey.size_in_bytes(), 16, 16, -1)]

    sessionKey = DecryptRSA(sessionKeyEncrypted, privateKey)

    decryptor = AES.new(sessionKey, AES.MODE_GCM, nonce)

    decryptor.update(header)

    decrypted = decryptor.decrypt_and_verify(ciphertext, tag)

    if outputFilePath is not None:

        with open(outputFilePath, 'wb') as outputFile:

            outputFile.write(decrypted)

    inputFile.close()

    return decrypted, str(header, 'utf-8')
```

```
#CODE FOR DECRYPTION USING RSA

def DecryptRSA(encrypted, privateKey):

    decryptor = PKCS1_OAEP.new(privateKey)

    decrypted = decryptor.decrypt(encrypted)

    return decrypted
```

```
def Decode(imagePath):

    image = Image.open(imagePath, 'r')

    data = ''

    imgdata = iter(image.getdata())

    while (True):

        pixels = [value for value in imgdata.__next__()[ :3] +
imgdata.__next__()[ :3] + imgdata.__next__()[ :3]]

        binstr = ''

        for i in pixels[:8]:

            if (i % 2 == 0):

                binstr += '0'

            else:

                binstr += '1'

        data += chr(int(binstr, 2))

        if (pixels[-1] % 2 != 0):

            return data
```

Now looking at the mail layout, the main function of it is to transfer the encrypted file to the receiver where the user is asked for encrypted file, his email address and password and the email of the receiver. Once clicked send mail the mail is sent with the encrypted file attached to it. The code implementation of sending mail is as follows.

```
smtpObj=smtpplib.SMTP_SSL("smtp.gmail.com",465)

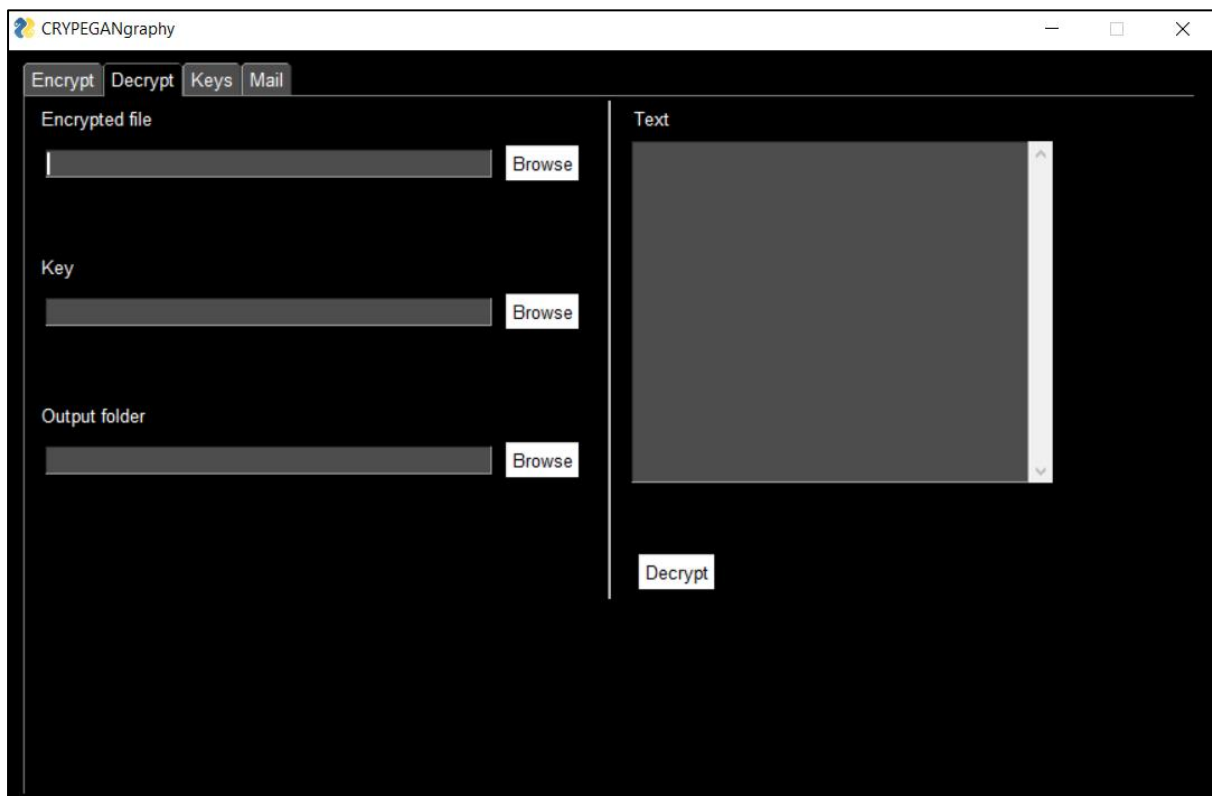
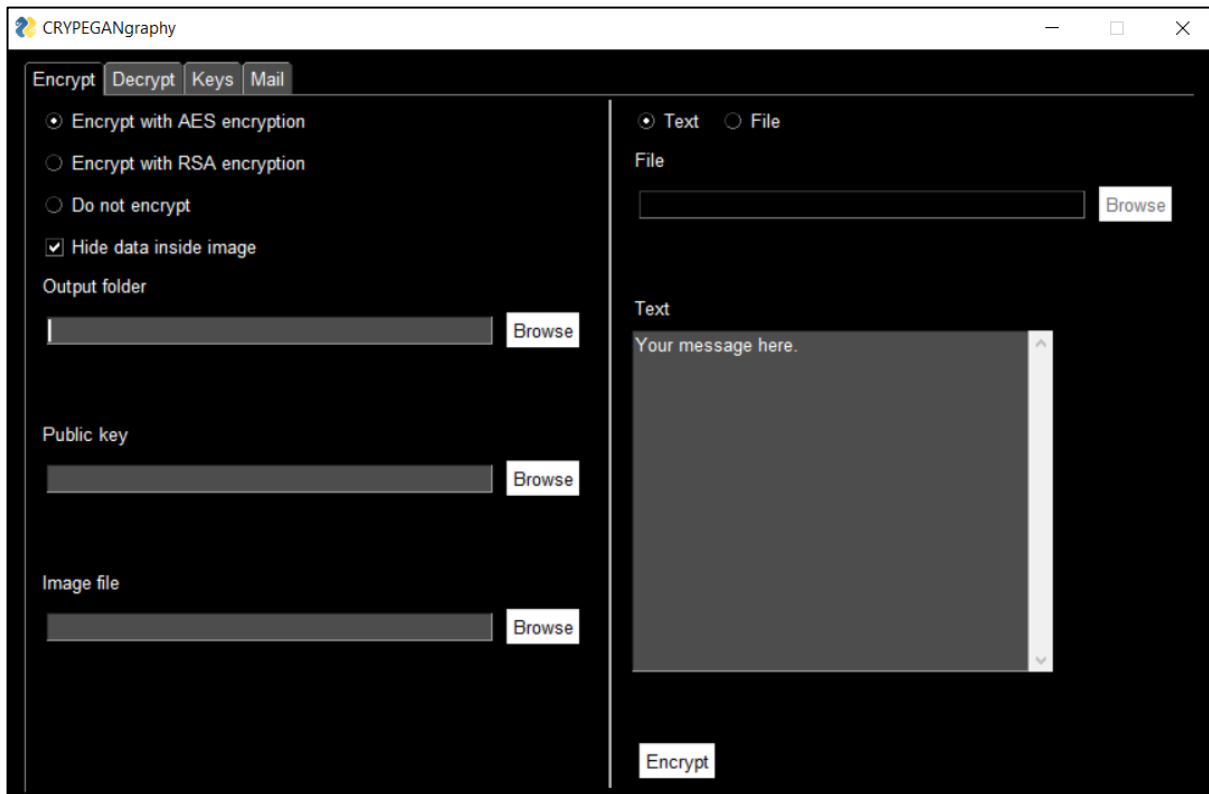
#smtpObj.login("admin123@gmail.com","admin123")
#smtpObj.sendmail('admin123@gmail.com','admin789@gmail.com','encrypted.
txt')

smtpObj.login(values['sender_addr'],values['mail_pwd'])
smtpObj.sendmail(values['sender_addr'], values['mail_addr'], message)

smtpObj.quit()
```



## SCREENSHOT



CRYPEGANgraphy

Encrypt Decrypt Keys Mail

Key size  
3072

Output folder  
 Browse

Generate keys

CRYPEGANgraphy

Encrypt Decrypt Keys Mail

Encrypted file  
 Browse

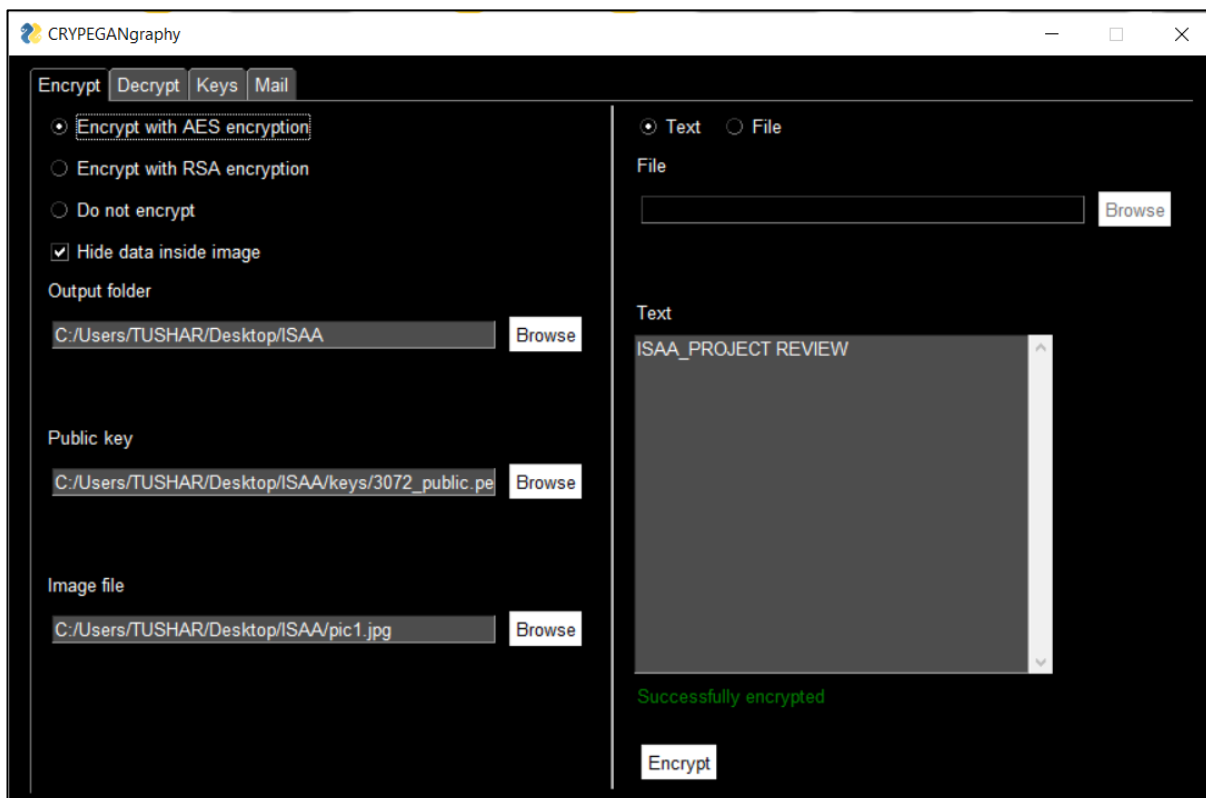
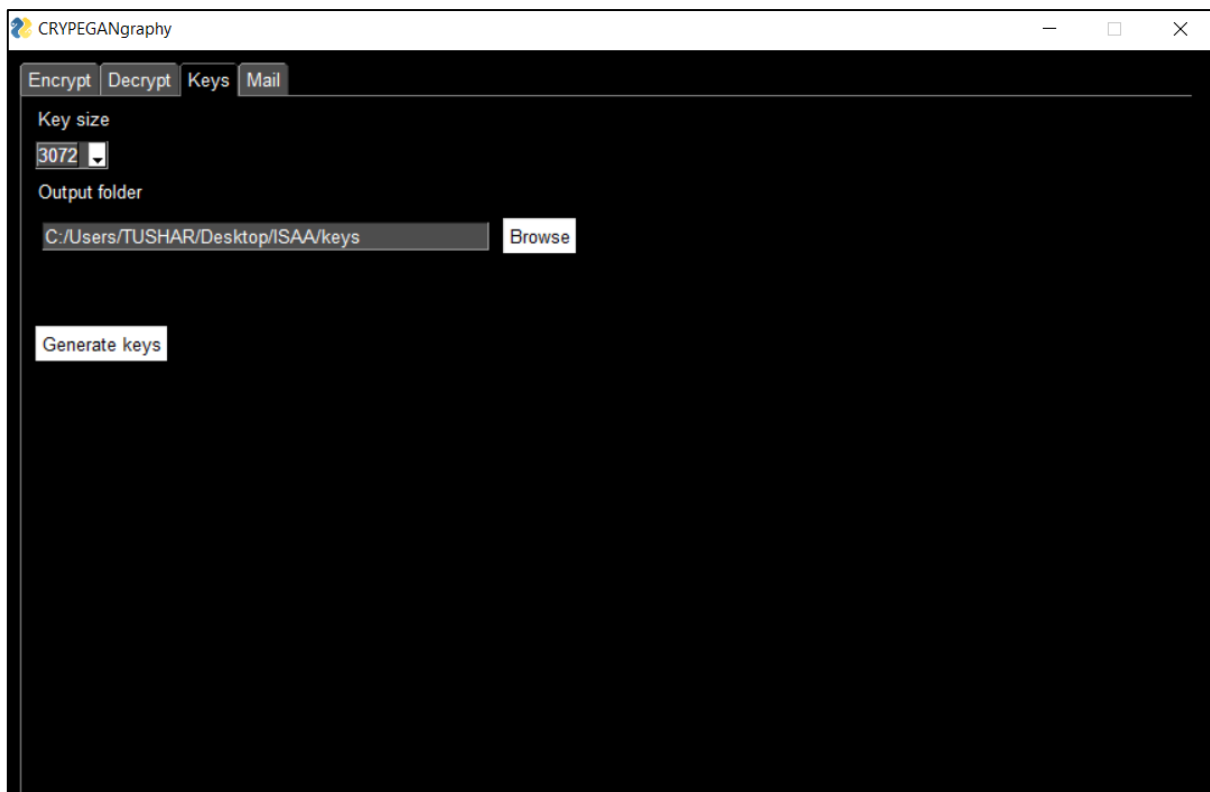
Your Email Address

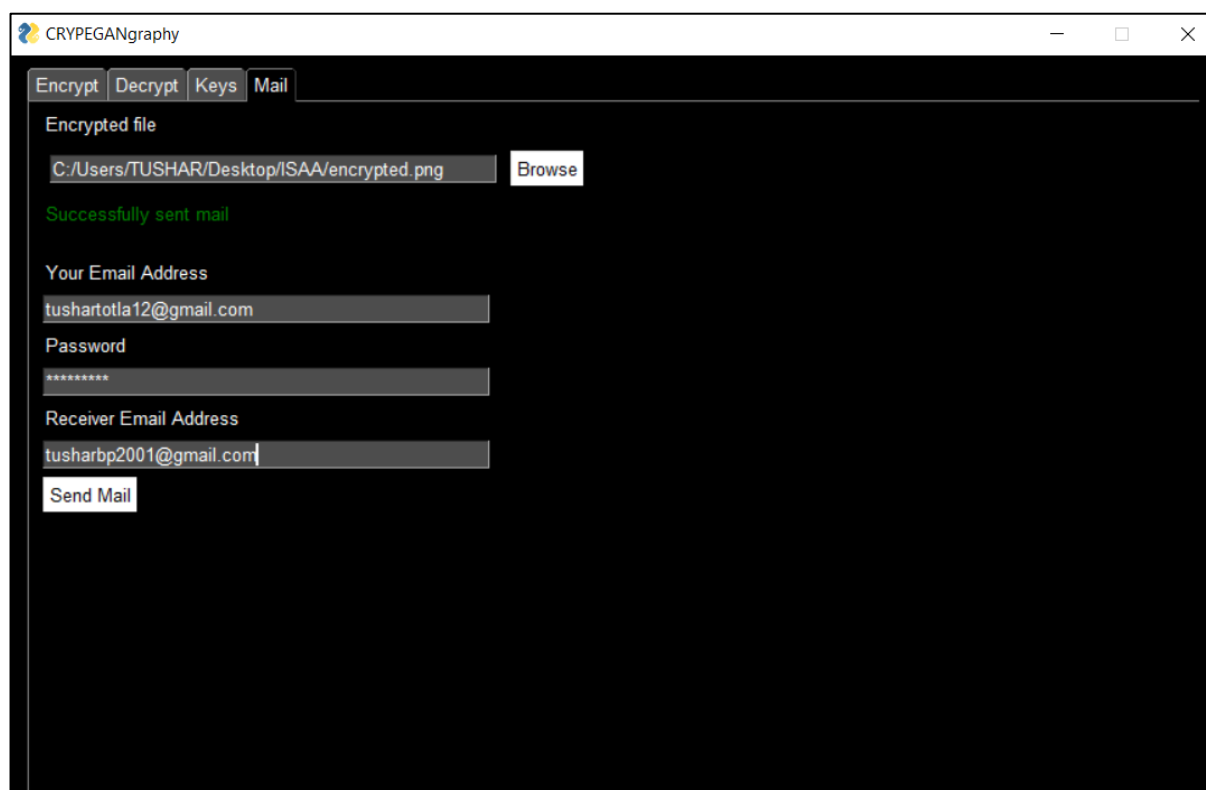
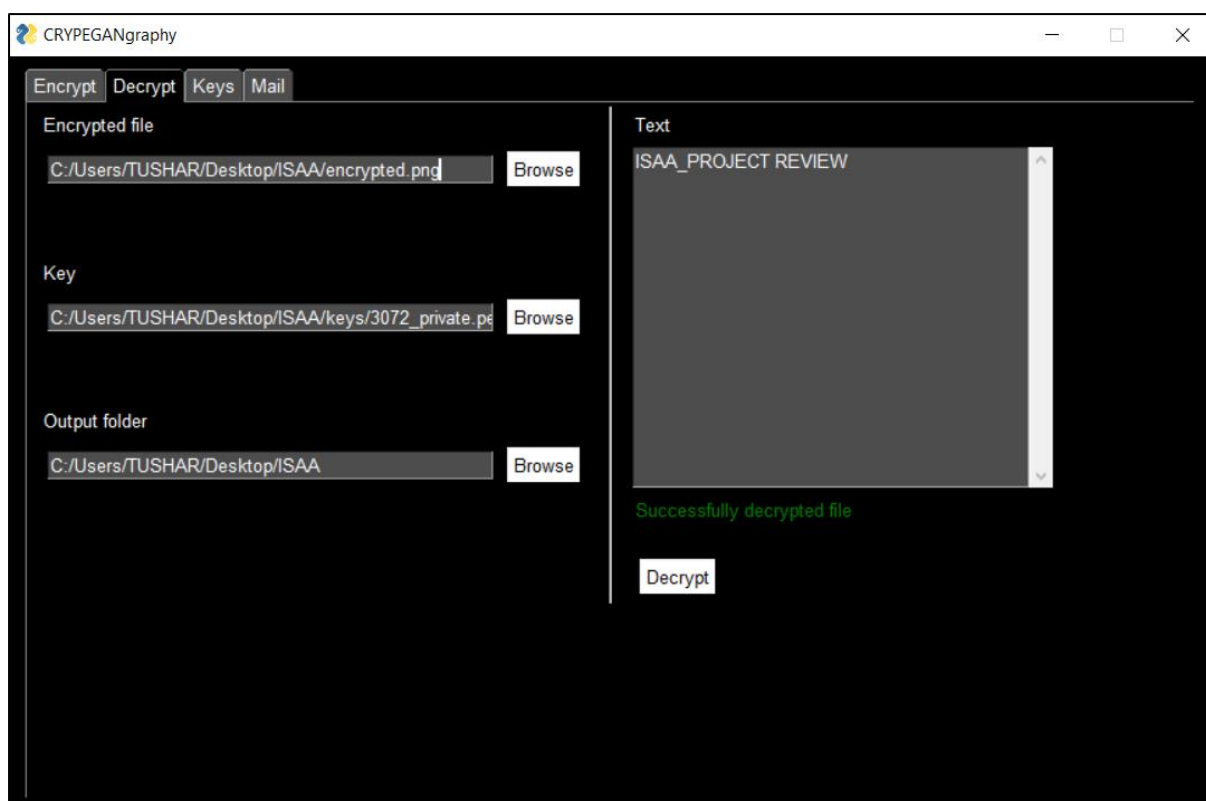
Password

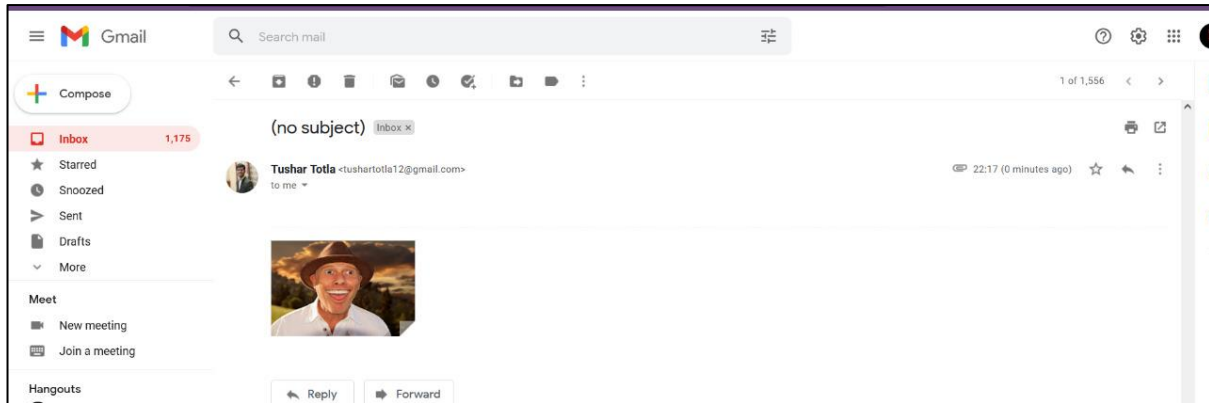
Receiver Email Address

Send Mail

## RESULTS:







*Screenshot of the mail received by the receiver mail.*

### **PERFORMANCE ANALYSIS:**

For evaluation of the proposed steganography technique a performance metric known as PSNR was used to evaluate the distortion between the image before and after the image were encoded with the secret text. Peak signal to noise ratio is used to compute how well the algorithm performed. PSNR computes peak signal to noise ratio, in decibels, between two images. PSNR is used to measure the quality of reconstruction of lossy compression codecs. This ratio is used as a quality measurement between two images, If PSNR ratio is high images are of best quality. Any visual artifacts in the stego-image should not be noticeable to the human eye. Peak Signal-to-Noise Ratio (PSNR), is used to find out whether the stego-image quality is acceptable or not. The PSNR can be calculated by the following equations.

$$\text{PSNR} = 10 \log_2((255)^2 / \text{MSE}) \text{ db}$$

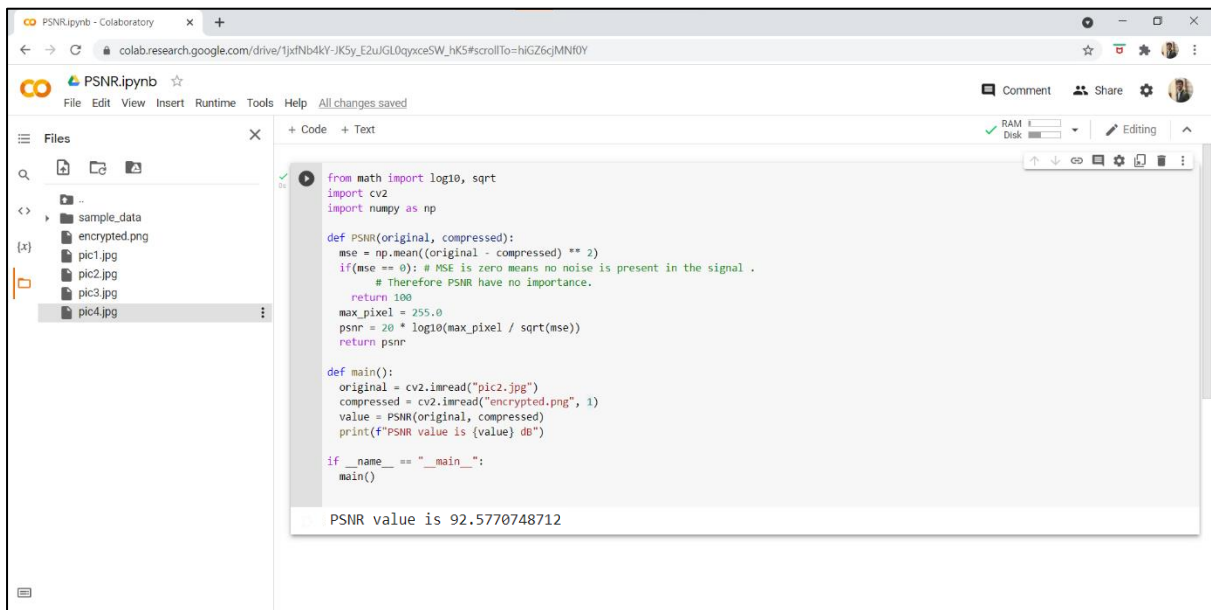
$$\text{MSE} = (1/P*Q) \sum_{M=0}^{P-1} \sum_{N=0}^{Q-1} (P(x, y) - P'(x, y))^2$$

Where, P and Q are image size. In the formula, P(x,y) stands for the original pixel value, and P'(x,y) pixel values of stego-image. The greater the PSNR the lesser the distortion.

The PSNR value approaches infinity as the MSE approaches zero; this shows that a higher PSNR value provides a higher image quality. At the other end of the scale, a small value of the PSNR implies high numerical differences between images.

Case	Image A – Image B	MSE	PSNR
1	Original – Encrypted	High	Low
2	Original – Decrypted	0	Infinite

## PSNR CODE:



```
from math import log10, sqrt
import cv2
import numpy as np

def PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if(mse == 0): # MSE is zero means no noise is present in the signal .
        # Therefore PSNR have no importance.
        return 100
    max_pixel = 255.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr

def main():
    original = cv2.imread("pic2.jpg")
    compressed = cv2.imread("encrypted.png", 1)
    value = PSNR(original, compressed)
    print(f"PSNR value is {value} dB")

if __name__ == "__main__":
    main()
```

PSNR value is 92.5770748712

## TEST IMAGES



*pic1*



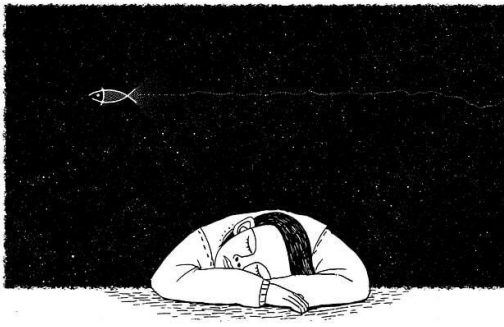
*encrypted*



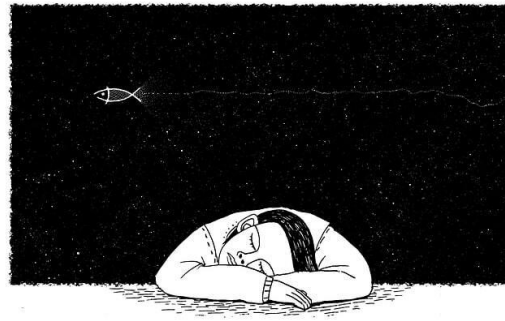
*pic2*



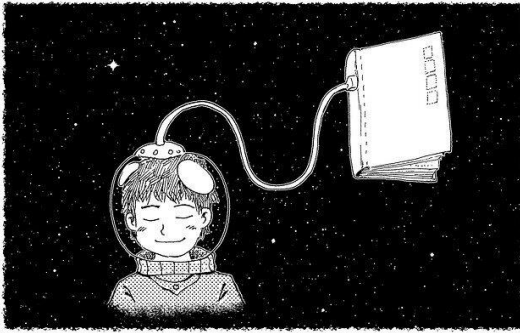
*encrypted*



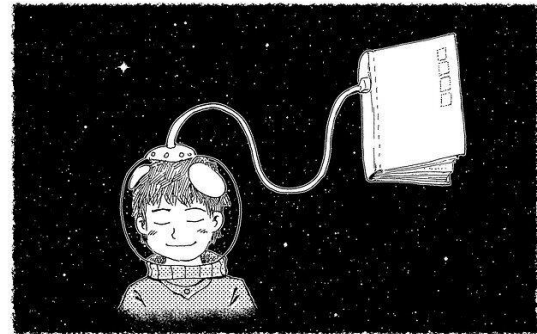
*pic3*



*encrypted*



*pic4*



*encrypted*

As we can see from the above images that there cannot be much difference observed between the two images with naked eyes, thus hiding the information in plain sight.

IMAGE	PSNR VALUE
<b>PIC1</b>	83.68325325
<b>PIC2</b>	92.5770748712
<b>PIC3</b>	90.22543984536
<b>PIC4</b>	85.95438621

The PSNR values obtained for the test images are high enough to prove the effectiveness and efficiency of the steganography algorithm.

According to the calculated PSNR values the proposed RGB steganography technique has proved to be an effective and efficient technique. High PSNR values are proven to the

minimum distortion of the images and thus are not visible to a normal human eye. Thus, the steganography technique used is highly secure. The AES encryption further adds security to the system providing symmetric encryption. The mail feature in the system provides an extended utility to the user to communicate secret messages to another person.

## **CONCLUSION AND FUTURE WORK**

From the whole study, we conclude that using both steganography and cryptography together refers to secure information and communication technique. The model states that the data to be passed on is input for the encryption process and the output of the it is the input for encoding purpose that is the steganography process. While to the other end, the decoding is done first via the steganographic techniques and then the decryption. This brings an extra security to the whole system for secure communication and transfer. The GUI implementation for the whole system giving choice to the user to choose his preferable technique for secure communication is added as a part of innovative idea to it.

Considering the future work, the application can be further improved by adding some more utilities and designing a better UI. Some features like providing more crypto algorithms like DES, Blowfish, SA etc. for encryption of the message can provide more utilities to the application. Overall, the present application is very handy, secured and interactive providing sufficient utilities to the users.