A secure method to transmit highly confidential information using - Compression, Encryption and Pixel Value Differencing

Group Members

Joby Mathew	CSU 152 24	MDL15CS047
Merin Francis	CSU 152 34	MDL15CS070
Ria Rajan Alappat	CSU 152 39	MDL15CS087
Tony Josi	CSU 152 56	MDL15CS113

Overview

The project aims at securing confidential data communicated via emails to be secured using a new methodology that combines Compression, Encryption and Pixel Value Differencing. The process of securing the data involves: Arithmetic coding was applied on secret message for lossless compression, which provided 22% higher embedding capacity. The compressed secret message is subjected to AES encryption; this provides higher security in the cases of steganalysis attacks. After compression and encryption, LSB substitution and PVD are applied.

Proposed method:

- Compression using ARITHMETIC CODING
- Encryption and decryption using AES
- Embedding procedure using PVD and LSB substitution
- Extraction procedures
- Sending mail with user interface

Module 1

- Arithmetic Coding Features
  - Efficient
  - Lossless compression
- Input File Format
  - Text File
- Output File Format
  - Frequency Table
  - Compressed Data

Step 1: Compression using arithmetic coding

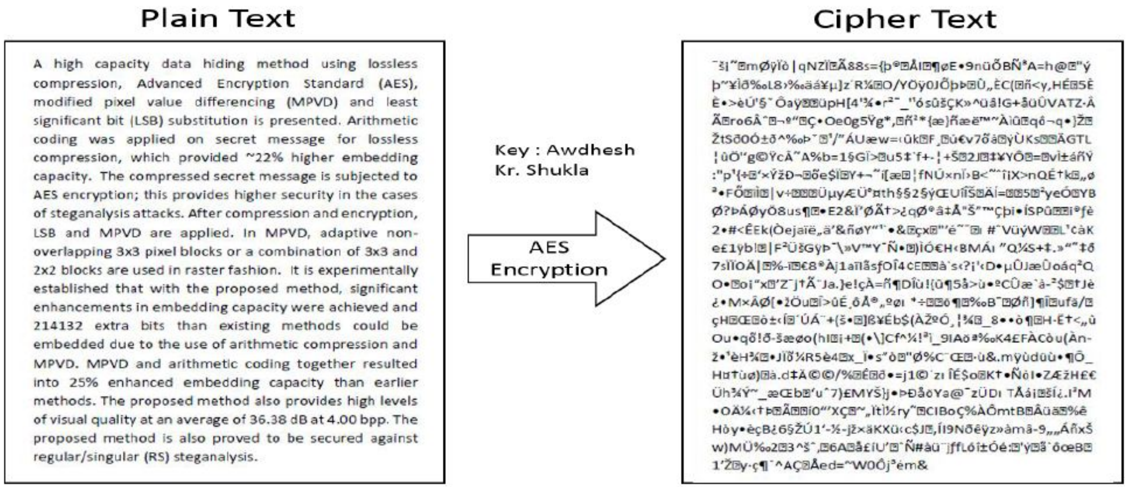
- When a string is converted to arithmetic encoding, here, fewer bits are used for frequently used characters and infrequently used characters are stored with more number of bits.
- This results in fewer bits required for total encoding.
- After compression, output of this step would be a bit stream of compressed secret message.
- Extra 0 bits are added at the end if required,to make the sequence divisible by 8.
- This bit stream is used as input for AES encryption.

Module 2

- **Encryption**
  - Technology Used: AES
  - Language Used: Python
  - External Libraries Used: Crypto
- **Decryption**
  - Technology Used: AES
  - Language Used: Python
  - External Libraries Used: Crypto

Step 2: Encryption and decryption using AES

- Compressed text file is taken as the input to AES.
- AES uses 128-bit data and variable length keys. AES is an iterative cipher, the number of rounds in AES vary with the length of the key.



Encryption using AES

Input: 128-bit data, 128-bit key
Output: Cipher text

Steps involved (For each round, repeat the following steps):
a. SubBytes()
b. ShiftRows()
c. MixColumns()
d. AddRoundkey()

Decryption using AES:

For each round, with the state and key as inputs (except for last round), following steps are repeated:

Step 1: Inverse_SubBytes()

Step 2: Inverse_ShiftRows()

Step 3: Inverse_MixColumns()

Step 4: Inverse_AddRoundkey()

```python
import base64,contextlib,sys
import hashlib
from Crypto import Random
from Crypto.Cipher import AES

class AESCipher(object):

    def __init__(self, key):
        self.bs = 32
        self.key = hashlib.sha256(key.encode()).digest()

    def encrypt(self, raw):
        raw = self._pad(raw)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return base64.b64encode(iv + cipher.encrypt(raw))

    def decrypt(self, enc):
        enc = base64.b64decode(enc)
        iv = enc[:AES.block_size]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return self._unpad(cipher.decrypt(enc[AES.block_size:])).decode('utf-8')

    def _pad(self, s):
        return s + (self.bs - len(s) % self.bs) * chr(self.bs - len(s) % self.bs)

    @staticmethod
    def _unpad(s):
        return s[:-ord(s[len(s)-1:])]
```

```python
def main(args):
    enc = AESCipher("jmrt")
    ipfile,opfile = args
    with open(ipfile, "r") as input:
        data = input.read()
        encdata = enc.encrypt(data)
    #print(encdata)
    outp = open(opfile,"w")
    outp.write(encdata)
    outp.close()



if __name__ == "__main__":
    main(sys.argv[1: ])
```

Module 3

- **Steganography**
  - Technology Used: Pixel Value Differencing
  - Language Used: Python
  - External Libraries Used: PIL
- **Inverse Steganography**
  - Technology Used: Pixel Value Differencing
  - Language Used: Python
  - External Libraries Used: PIL

Embedding Procedure:

- LSB+PVD method is applied to embed the message in cover image.
- the cover image is divided into non-overlapping pixel blocks of 3x3 or 3x3 + 2x2 pixel blocks as per the cardinality of the cover image
- In every 3x3 sized block one pixel is taken as the reference pixel ac and we embed 3 pixels directly into LSB of the reference pixel ac to form ac' and the optimize it to form ac".
- Calculate the difference di, for all pixel values except for ac. di = | ac - ai

Table 1 : Range Table

	Lower level		Higher level		
Range	R1	R2	R3	R4	R5
Lower-Upper bound	[0-15]	[16-31]	[32-63]	[64-127]	[128-255]
No of bits (t_i)	3 bits	4 bits	5 bits	5 bits	5 bits

- There is no reference pixel for2x2 sized blocks
- Bits of the secret message is embedded in every pixel ai to form ai' and then optimised to form a"
- Optimization technique is different for reference pixel and other pixels.

Extraction Procedure

- Read the cover image I and divide it into 3x3 + 2x2 non overlapping blocks.
- Extract secret bits from LSB of the reference pixel sc
- Extract all bits embedded in ai si
- Concatenate all bits sc and si to obtain the real secret message.

Embedding data into cover image

- LSB substitution and PVD are applied. In PVD, adaptive non-overlapping 3x3 pixel blocks or a combination of 3x3 and 2x2 blocks are used in raster fashion.
- The cover image is divided and substituted with data bits depending on the difference between pixel values.
- In the embedding method, the cover image is divided into non-overlapping pixel blocks of 3x3 or 3x3 + 2x2 pixel blocks as per the cardinality of the cover image. That is, if the dimensions of the cover image are not divisible by 3x3 pixel blocks, the remaining pixels are taken 2x2 pixel blocks.

Embedding

Embedding Capacity Calculation — Before embedding data into cover image the embedding capacity is calculated by *calcCapacity()* function

Then for each char read from the i/p file its bits are substituted with LSB's of the pixels depending on PVD.

Number of LSB’s substituted based on PVD:

- If PVD < 16:
  - Bits substituted = 3
- If 16 < PVD < 32
  - Bits substituted = 4
- Else:
  - Bits substituted = 5

Basic Algorithm for Embedding Data:

- Divide pixels into group of 3*3 matrix.
- Calculate PVD between ref. Pixel and neighbors.
- Depending upon PVD substitute LSBs with data values.
- Save the new image with altered LSBs.

Extraction

During Extraction the cover image is taken as input and extraction is performed on the cover image to obtain the final data recovered.

LSBs are read from the pixel values and combined to form each character of the encrypted file.

Basic Algorithm for Extracting Data:

- Loop through the pixel values of the input image
- Collect the LSB values
- Combine bits to form the characters in the encrypted file
- Save the recovered data into a new file

Embedding Module

```python
# Calculate embedding capacity of the given cover image
def calcCapacity():
    global capacity

    # Divide pixels to [3 x 3] matrix
    for i in range(0, lix * 3, 3):
        for j in range(0, liy * 3, 3):

            # Obtain pixel values of ref. pixel
            rref, gref, bref = pix[i + 1, j + 1]

            # For all pixels in the matrix
            for k in range(i, (i + 3)):
                if k >= hi:
                    break
                for l in range(j, (j + 3)):
                    if k == i + 1 and l == j + 1:
                        continue
                    if l >= wi:
                        break

                    # Calculate the difference in pixel values
                    r, g, b = pix[k, l]
                    rdif = r - rref
                    gdif = g - gref
                    bdif = b - bref
                    rdif = abs(rdif)
                    gdif = abs(gdif)
                    bdif = abs(bdif)

                    # Cumulative capacity
                    capacity = (
                        capacity + classify(rdif) + classify(gdif) + classify(bdif)
                    )
```

```python
# Classify pixels based on the difference in pixel value to the number of bits to be substituted to LSB
def classify(pvd):
    nbits = 0
    if pvd < 16:
        nbits = 2
    elif 16 < pvd < 32:
        nbits = 3
    else:
        nbits = 4
    return nbits
```

```
            # For all pixels in the matrix
            for k in range(i, (i + 3)):
                if k >= hi:
                    break
                for l in range(j, (j + 3)):
                    if k == i + 1 and l == j + 1:
                        continue
                    if l >= wi:
                        break

                    # Calculate pixel value difference
                    r, g, b = pix[k, l]
                    rdif = r - rref
                    gdif = g - gref
                    bdif = b - bref
                    rdif = abs(rdif)
                    gdif = abs(gdif)
                    bdif = abs(bdif)

                    # Till embedding gets completed
                    if completed == 0:
                        newr = embedbits(k, l, "r", classify(rdif), r)
                    if completed == 0:
                        newg = embedbits(k, l, "g", classify(gdif), g)
                    if completed == 0:
                        newb = embedbits(k, l, "b", classify(bdif), b)

                    # Embedding completed
                    if completed == 1:

                        # Assign modified pixel values
                        pix[k, l] = (newr, newg, newb)

                        # Save embedded image
                        im.save("protest.png")

                        # Close log file
                        lg.close()
```

Extraction

```python
# File Objects creation
im = Image.open(sys.argv[1])
outp = open(sys.argv[2], "w")
lg = open("embedlog.log", "r")

# Initialisation
pix = im.load()
temp = 1
chrtr = ""

# Main Function
def main():
    global chrtr, temp
    while True:

        # Read each line from log file
        st = lg.readline()

        # Check if log file reached its end
        if len(st) == 0:
            # Write extracted data to file
            # print(chr(int(chrtr, 2)))
```

```python
        # Check if log file reached its end
        if len(st) == 0:
            # Write extracted data to file
            # print(chr(int(chrtr, 2)))
            outp.write(chr(int(chrtr, 2)))
            break

        # Unpack line read from log file to variables
        i, j, pixel, diff, pad, charNum = st.split()

        # Process variables
        i = int(i)
        j = int(j)
        diff = int(diff)
        pad = int(pad)
        charNum = int(charNum)
        r, g, b = pix[i, j]

        # Check if a new character in embed log is reached
        if temp != charNum:
            # print(chr(int(chrtr, 2)), end="")
            outp.write(chr(int(chrtr, 2)))
            chrtr = ""

```

```
        if temp != charNum:
            # print(chr(int(chrtr, 2)), end="")
            outp.write(chr(int(chrtr, 2)))
            chrtr = ""

        # If embedded pixel is red
        if pixel == "r":
            binr = bin(r)
            chrtr += binr[(len(binr) - diff) :]

        # If embedded pixel is green
        if pixel == "g":
            binr = bin(g)
            chrtr += binr[(len(binr) - diff) :]

        # If embedded pixel is blue
        if pixel == "b":
            binr = bin(b)
            chrtr += binr[(len(binr) - diff) :]

        # Unpad if padding is done
        if pad != 0:
            chrtr = chrtr[: (len(chrtr) - pad)]
```

Module 4

- **Sending mail with attached cover image**
  - Technology Used: SMTP
  - Language Used: Python
  - External Libraries Used: smtplib
- **Receiving mail with attached cover image**
  - Technology Used: SMTP
  - Language Used: Python
  - External Libraries Used: smtplib

```python
def send_mail(send_from, send_to, subject, text, passw, files=None):

    smtp = smtplib.SMTP("smtp.gmail.com: 587")
    smtp.starttls()
    smtp.login("tordown97@gmail.com", passw)


    msg = MIMEMultipart()
    msg["From"] = send_from
    msg["To"] = COMMASPACE.join(send_to)
    msg["Date"] = formatdate(localtime=True)
    msg["Subject"] = subject

    msg.attach(MIMEText(text))

    for f in files or []:
        with open(f, "rb") as fil:
            part = MIMEApplication(fil.read(), Name=basename(f))

        part["Content-Disposition"] = 'attachment; filename="%s"' % basename(f)
        msg.attach(part)

    smtp.sendmail(send_from, send_to, msg.as_string())
    smtp.close()
```

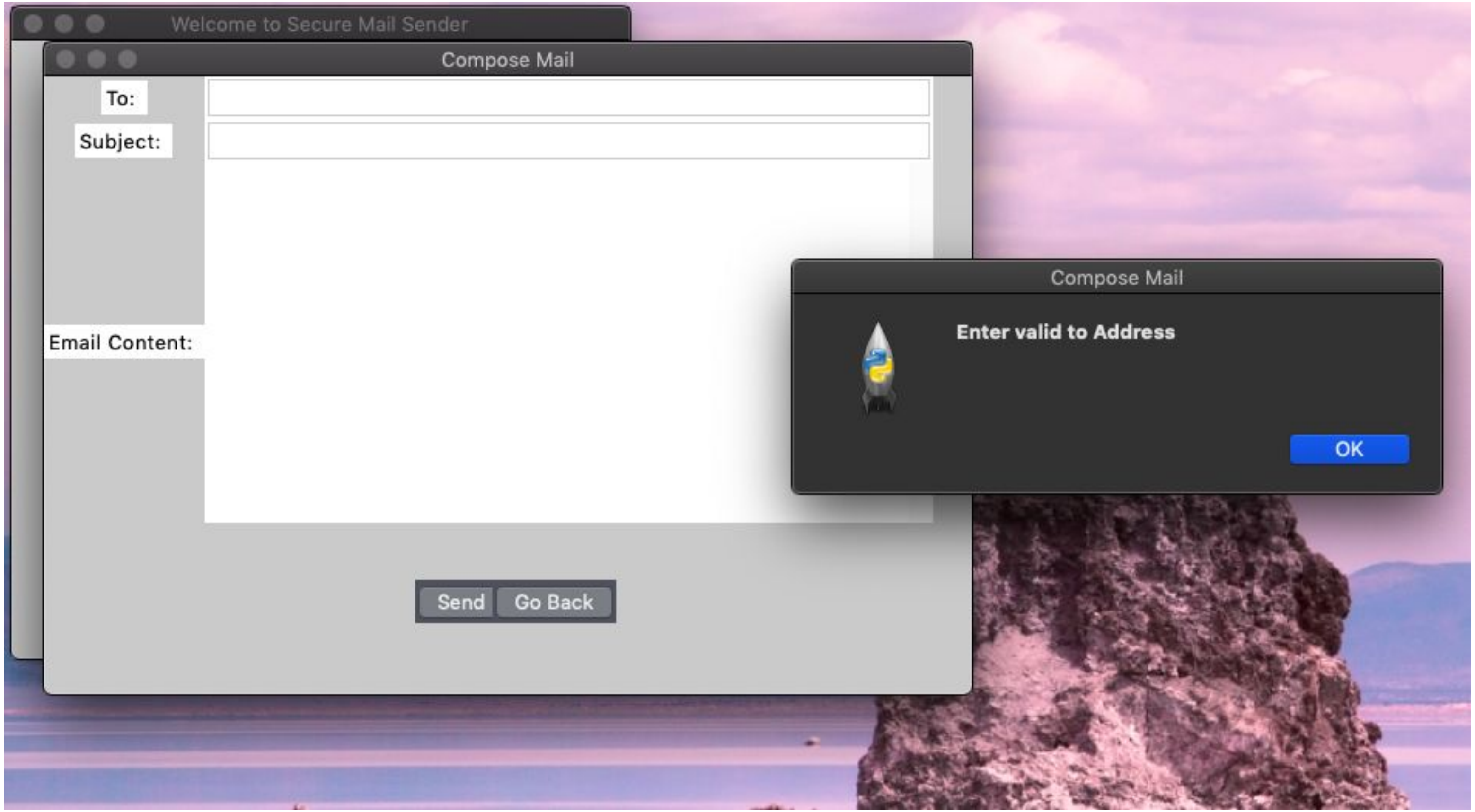
GUI

```python

window = tk.Tk()
window.title("Welcome to Secure Mail Sender")
window.geometry("400x400")
window.configure(background="grey")


def closeWindow():
    window.destroy()


btn = ttk.Button(window, text="Compose Mail", command=sendMailCallBac
    relx=0.5, rely=0.35, anchor=tk.CENTER
)
btn = ttk.Button(window, text="Extract Data from Cover Image", comman
    relx=0.5, rely=0.55, anchor=tk.CENTER
)
btn = ttk.Button(window, text="Quit", command=closeWindow).place(
    relx=0.5, rely=0.75, anchor=tk.CENTER
)
window.mainloop()

```

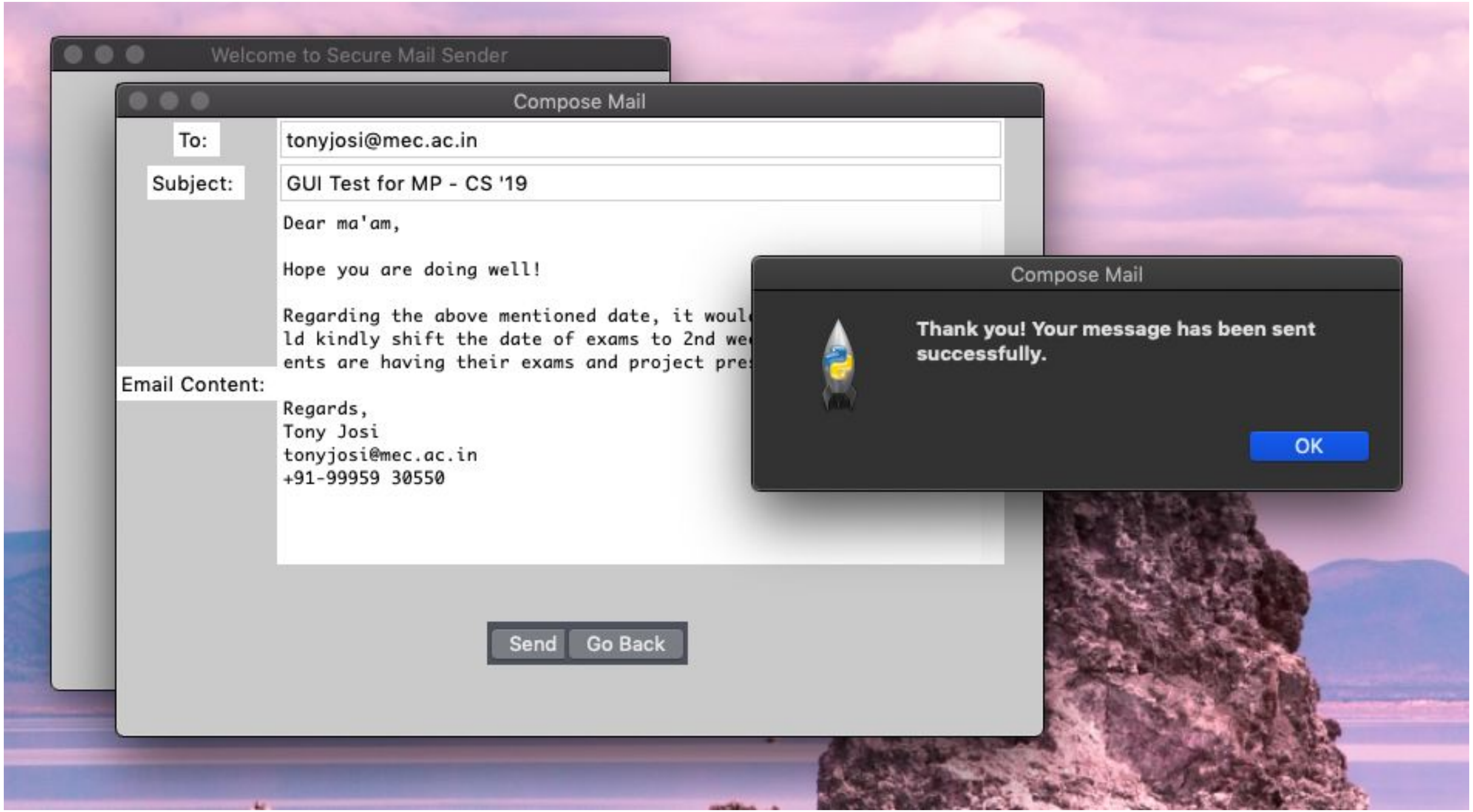

Welcome to Secure Mail Sender
Compose Mail
Extract Data from Cover Image
Quit

Basic UI

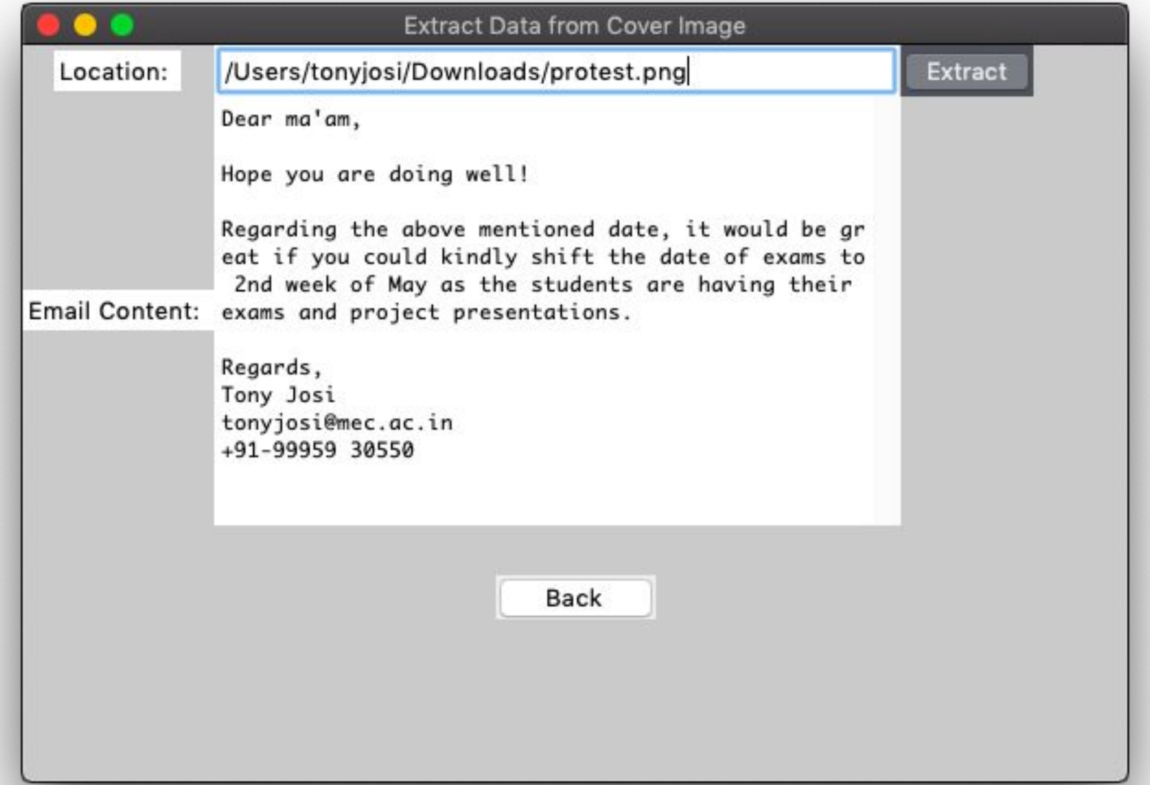
Compose Mail



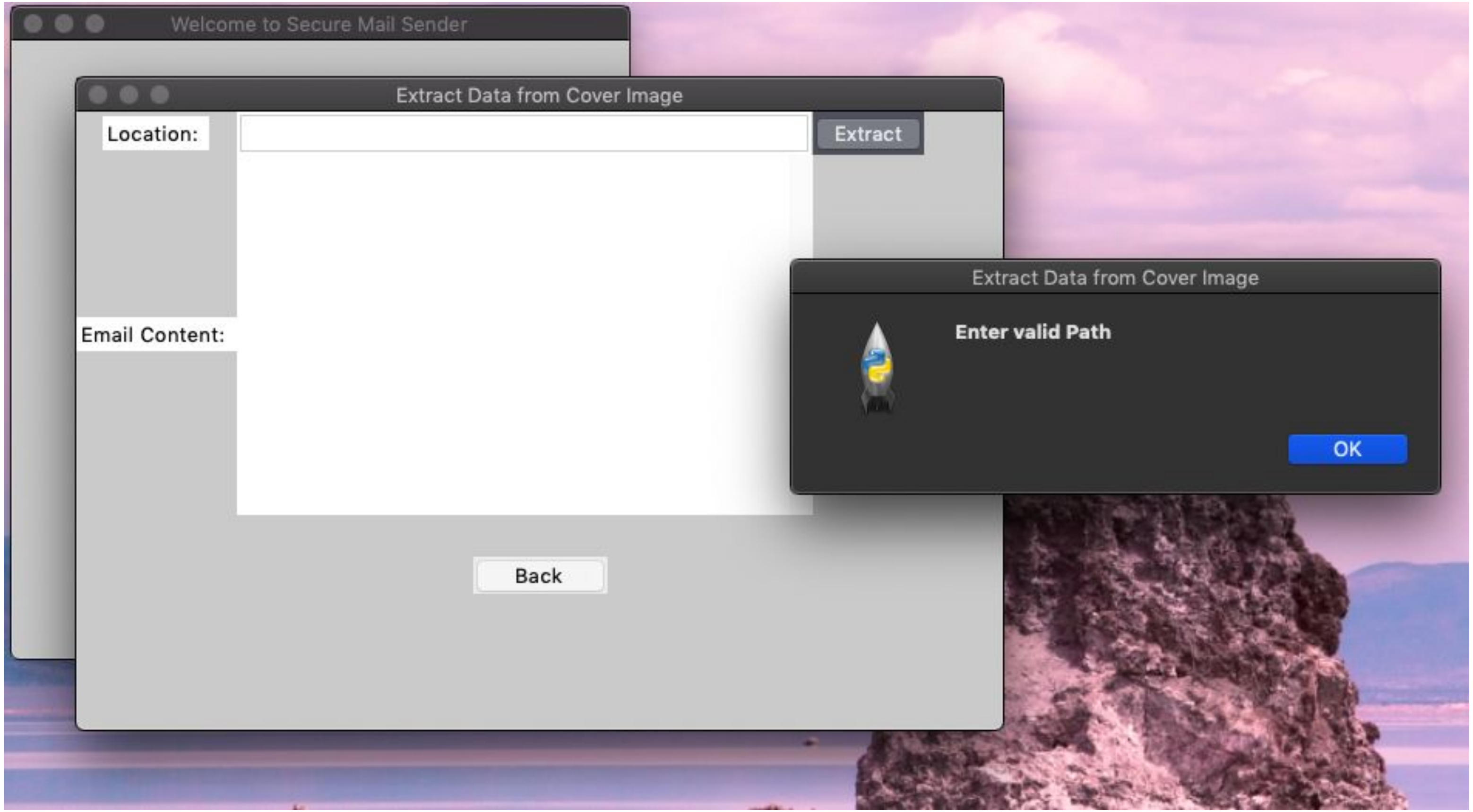
Alert Boxes for Compose Mail Window



Alert Boxes for Successfully Sending of Mail



Extracting Encrypted data from Cover Image



Alert Boxes for Extract Data Window

Testing

Unit Testing

Compression Module - A text file is given as input to the module and compression is performed an average 50% compression is achieved.

Encryption Module - The unit is tested by passing a text file as input and AES encrypted output file is obtained.

Embedding Module - A cover image and text file is given as input and embedded cover image is obtained as output.

Sending Mails - The module is tested by sending mails to the given ID’s with given files as attachments.

Integration Testing

We integrated our modules one by one and tested the partially integrated system.

System Testing

All modules were integrated at the end of integration testing and the entire system was tested by sending a textual message to a given mail ID using the mentioned processing steps.

Results

Original and Embedded Image Comparison

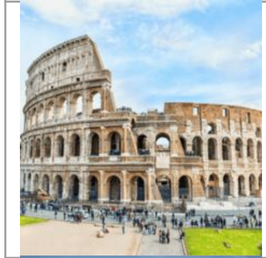
Original	Embedded	Quality Parameters
		Mean Square Error: 0.1996641975308642 Peak Signal to Noise Ratio: 55.1278016384 dB Mean Structural Similarity Index: 0.999256585841
		Mean Square Error: 0.49475555555555556 Peak Signal to Noise Ratio: 51.1868968132 dB Mean Structural Similarity Index: 0.999662989898
		Mean Square Error: 0.40630123456790124 Peak Signal to Noise Ratio: 52.0423221891 dB Mean Structural Similarity Index:0.999469287703
		Mean Square Error: 0.16094814814814815 Peak Signal to Noise Ratio: 56.0639437676 dB Mean Structural Similarity Index: 0.99883054049333

Image Quality Comparison with Varying Input File Size done on Same Cover Image

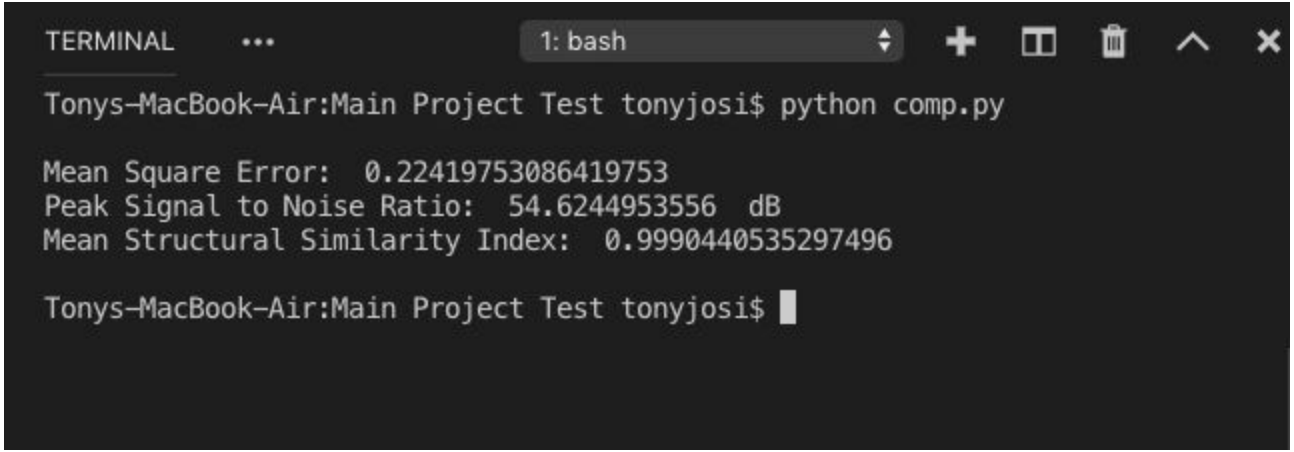
Test Image Info.:

- Image Size - 84 KB
- Image Dim. - 225 * 225
- Color Space - RGB

Test 1 - Input Size: 644 Bytes

```
TERMINAL   •••        1: bash

Tonys-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error:  0.11156543209876543
Peak Signal to Noise Ratio:  57.6555070905  dB
Mean Structural Similarity Index:  0.9996067211756375

Tonys-MacBook-Air:Main Project Test tonyjosi$
```

Test 2 - Input Size: 1289 Bytes



Test 3 - Input Size: 1934 Bytes

```
TERMINAL   •••                1: bash

Tonys-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error:  0.3808
Peak Signal to Noise Ratio:  52.3238342116  dB
Mean Structural Similarity Index:  0.9985222271170396

Tonys-MacBook-Air:Main Project Test tonyjosi$
```

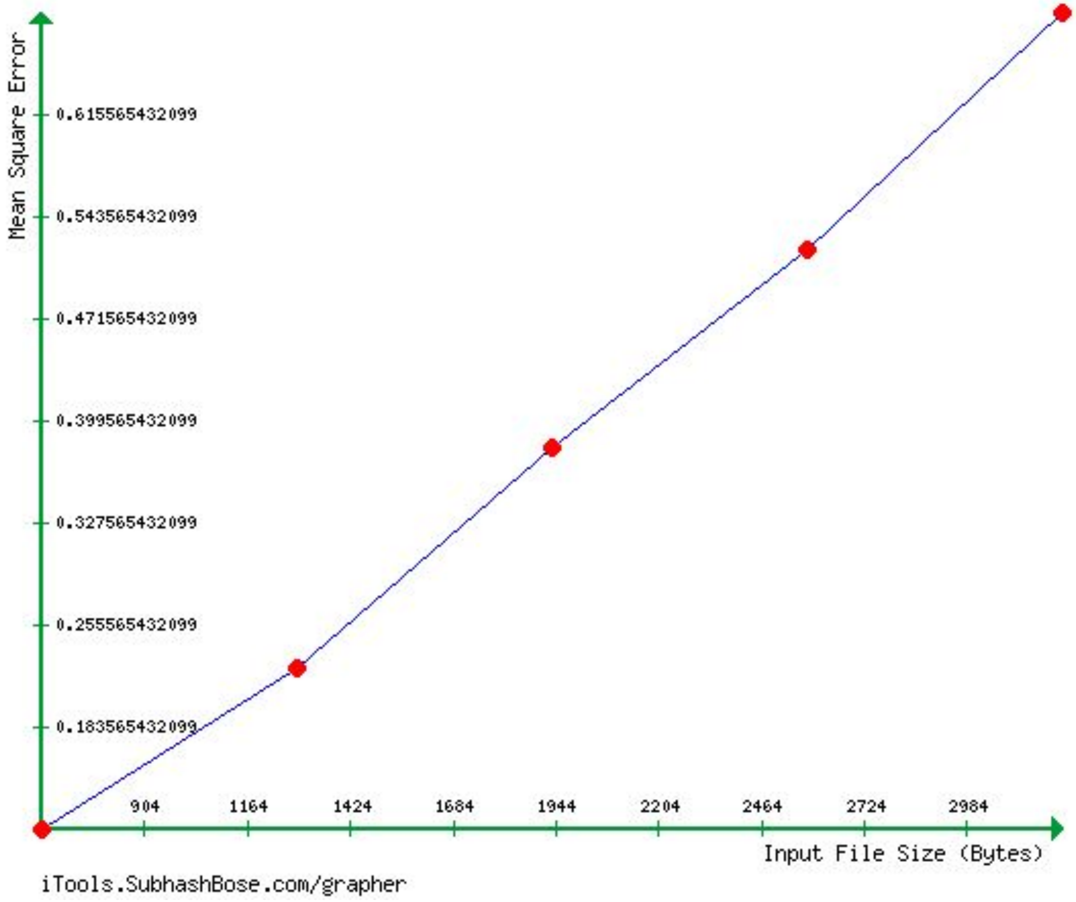
Test 4 - Input Size: 2579 Bytes

```
TERMINAL   •••                1: bash

Tonys-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error:  0.5209481481481482
Peak Signal to Noise Ratio:  50.9628586232  dB
Mean Structural Similarity Index:  0.9981800838607368

Tonys-MacBook-Air:Main Project Test tonyjosi$ 
```

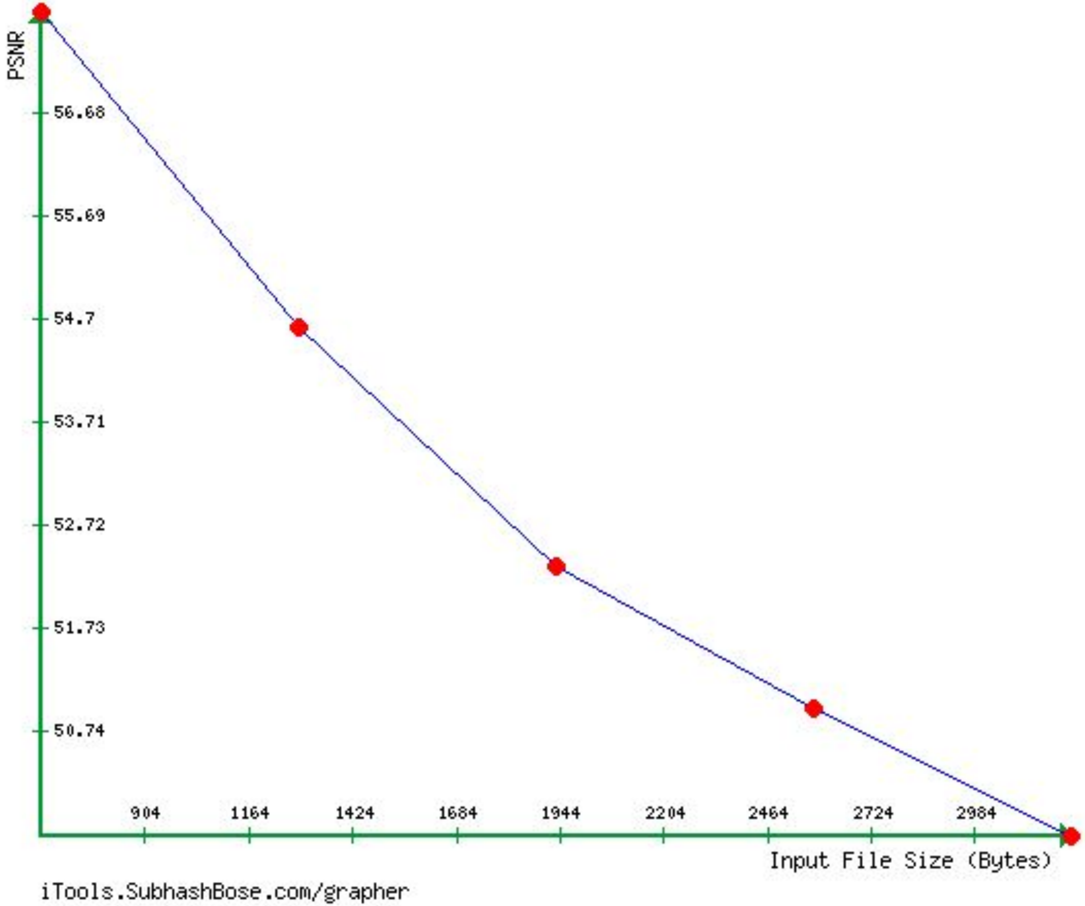
Test 5 - Input Size: 3224 Bytes

```
TERMINAL   •••        1: bash

Tonys-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error:  0.6882962962962963
Peak Signal to Noise Ratio:  49.7530492823  dB
Mean Structural Similarity Index:  0.9978115343534433

Tonys-MacBook-Air:Main Project Test tonyjosi$
```

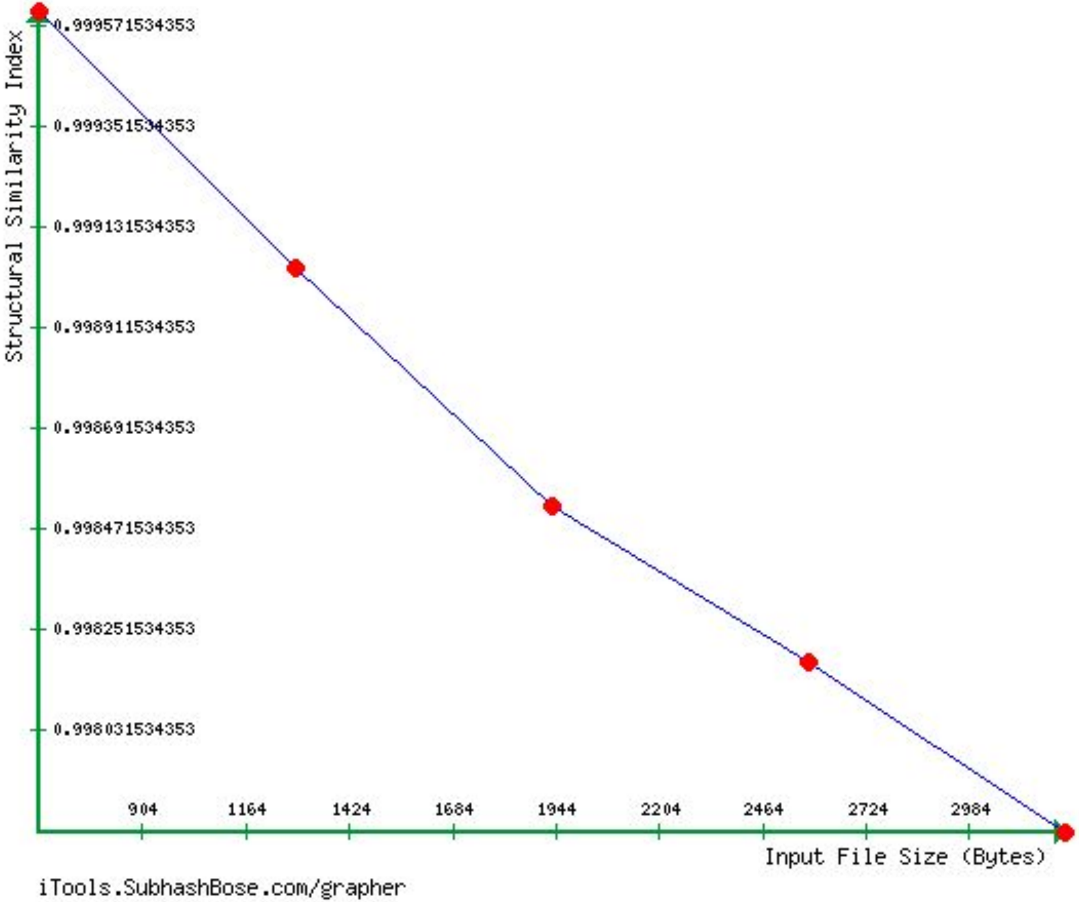
Plots for Image Quality Comparison



Mean Square Error (MSE)



Peak Signal to Noise Ratio (PSNR)



Structural Similarity

Conclusion

- A fully functional methodology for sending highly confidential information via e-mails using cryptographic and stenographic methods has been developed.
- Arithmetic coding was applied on secret message for lossless compression, which provided 22% higher embedding capacity. Then the data is undergone encryption using AES which provides higher security in the cases of steganalysis attacks.
- The encrypted data is then embedded to a cover image using a new steganographic method - PVD (Pixel Value Differencing) with LSB substitution.
- The interface for sending mails is developed using Python GUI module Tkinter

Future Scope

- In this project, we have considered both the message and the key to be of 128 bits. The strength of the AES algorithm may be enhanced by increasing the key length from 128 bits to 512 bits. As a result, the number of rounds is increased in order to provide a stronger encryption method for secure communication.
- Enhancements in image based steganography can be done using soft computing techniques such as Neural based steganography, Fuzzy and Genetic algorithms based approaches.
- System can be modified to develop features like retrieving mails automatically from the receiver.

Thank You