Python Team Project Report

[Grant Stevenson, Ankur Raghavan, Arjun Narayanan]

[022]: Team [11]

[ENG 13300]

[Ben Manning]

[10/17/24]

1. Project Motivation

Write your project motivation section here. Discuss how image processing and cipher techniques are used in your chosen engineering discipline, citing at least three appropriate references.

In computer engineering, image processing and cipher techniques are often used in areas such as data security, communications, and digital media manipulation. These methods have significant applications in areas like encrypted communication, secure data storage, and biometric systems.

Image processing is frequently used by engineers in Facial recognition systems. Image enhancement is one of the key areas in image processing where techniques such as filtering, histogram equalization, and edge detection are employed to improve image clarity and have drastically improved fields like medical diagnostics (Gonzalez and Woods). For cypher techniques, they are used to encrypt data. Cypher techniques are an important part of network security. With secure transmission protocols like SSL/TLS relying on cipher techniques to ensure data integrity, it is easy to see how useful they can be in high security transmissions used by financial and government institutions (Anderson and Kuhn).

One field of research that combines these two techniques is steganography, which is a method in which an image is used as a cover to hide data within its pixels. This technique provides a secure way to transmit information without raising suspicion. Another area is image encryption, where techniques like chaos theory are used to scramble image data, making it unreadable to unauthorized viewers unless decrypted with the appropriate key.

A study by Jassim and Abbas shows that chaos-based encryption methods applied to images can secure visual information in secure communication or surveillance systems (Jassim and Abbas).

This combination of image processing with cipher techniques can play a crucial role in securing digital media in various fields, such as cybersecurity and entertainment.

2. Project Overview and Methods

Write your project overview and methods section here. This includes a description of algorithms, a summary of background research, and evidence to support the methods chosen for each of the processing steps.

https://www.telsy.com/en/steganography-from-its-origins-to-the-present/

The main purpose of this demonstration is to use python for encrypting and decrypting images using three different ciphers. The first cypher is a Caesar cipher: shifting the message by a certain integer value across the whole message. The second cypher in the program is the Vigenère Cipher: shifts each value in the message independently by the value of the key. Lastly is the XOR cipher; this cipher converts the text into binary and compares each bit in the byte through an XOR gate. The practice of concealing messages or information within data is called steganography.

Steganography has been a technique that has been around since ancient Greece. The first recorded use of steganography was in 440BC when Johannes Triremes hid a message in a book about magic. Once innovations progressed, encrypting messages in books changed into microdots used in world wars. Eventually, computers created a completely distinct way of

communication. Today, steganography is used in copious ways: securing information, protecting confidential data, cybersecurity, etc. Crypto steganography is the type of steganography that is used in the project.

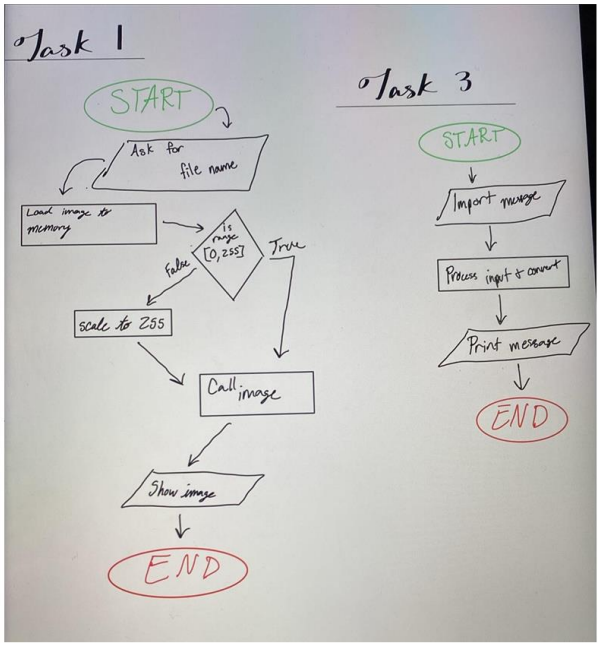
Within the project we used Least significant bits to give out information. The LSB is what stored the binary for the code. Withdrawing the LSB form the picture allowed the team to make a list of bytes that held the information that would be converted back through the cipher to get the encoded message. We convert the LSB into an ascii number then run it though the Ceaser, XOR, or Vigenère cipher. Then once through the cipher we change the value back to ascii, then back to a letter/number.

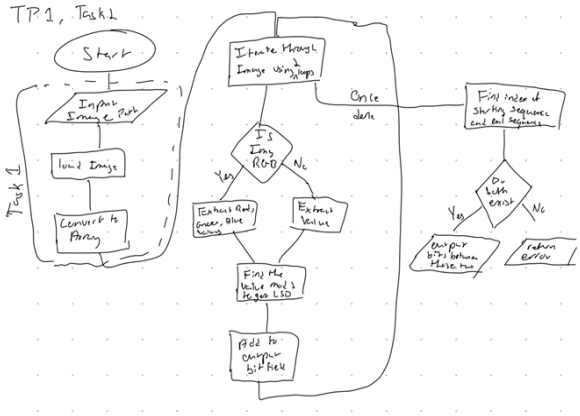
3. Discussion of Algorithm Design

In our design process for the image-based encryption system, we demonstrated a comprehensive approach to problem-solving. We began by defining the problem and gathering specific requirements, which informed our development of a modular system architecture comprising analysis and main functions. This structure allowed us to focus on component development, with our analysis function handling image processing and our main function managing user inputs and overall flow. We implemented an iterative approach evident in the multiple checks and conditional operations within our analysis function, while our accommodation of both color and grayscale images shows our consideration of multiple solution paths. We implemented proactive error handling and risk management by validating image dimensions. We designed a specific algorithm for image data processing, working within 8-bit color depth constraints. Our integration of an external encryption function demonstrates our pragmatic use of existing tools. We addressed output generation through figure plotting, saving, and result printing. Throughout the process, we continuously evaluated our solution against requirements and

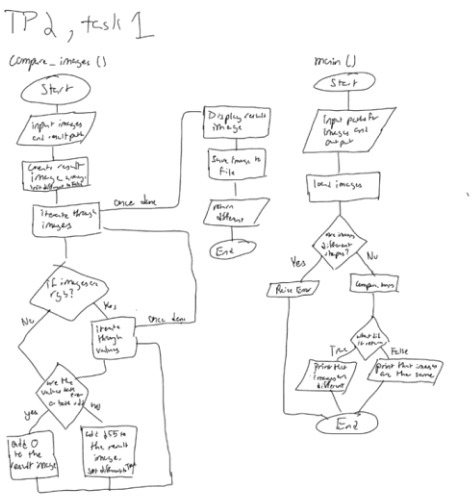
made trade-offs, such as our decision to divide pixel values by 2, balancing encryption complexity and effectiveness. This holistic approach reflects our thoughtful and systematic design process that addresses the challenges of creating an image-based encryption system.

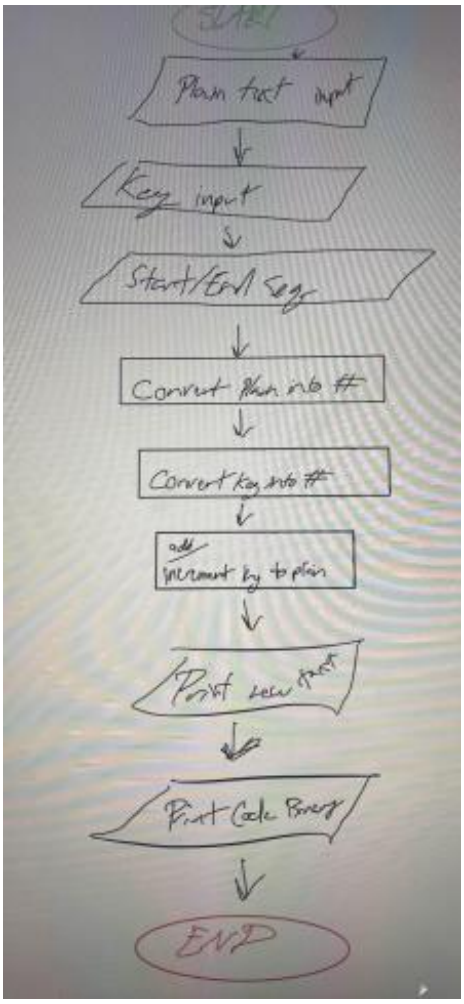
Tp1





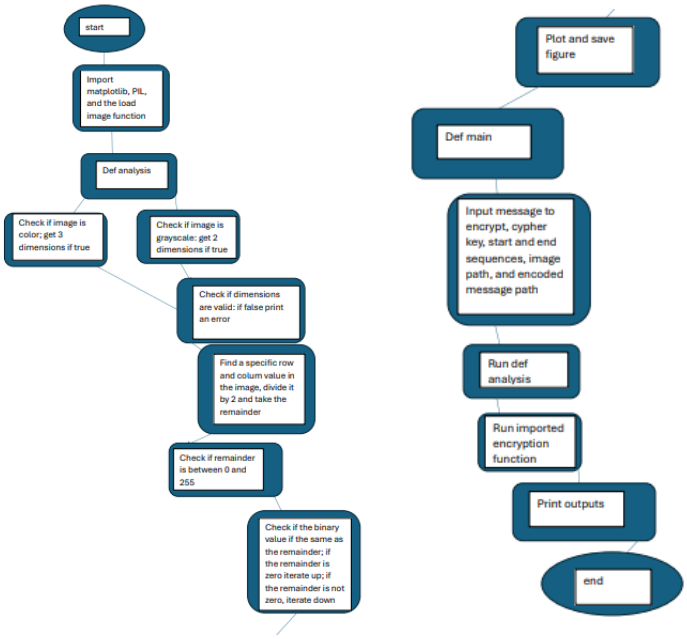
Tp 2



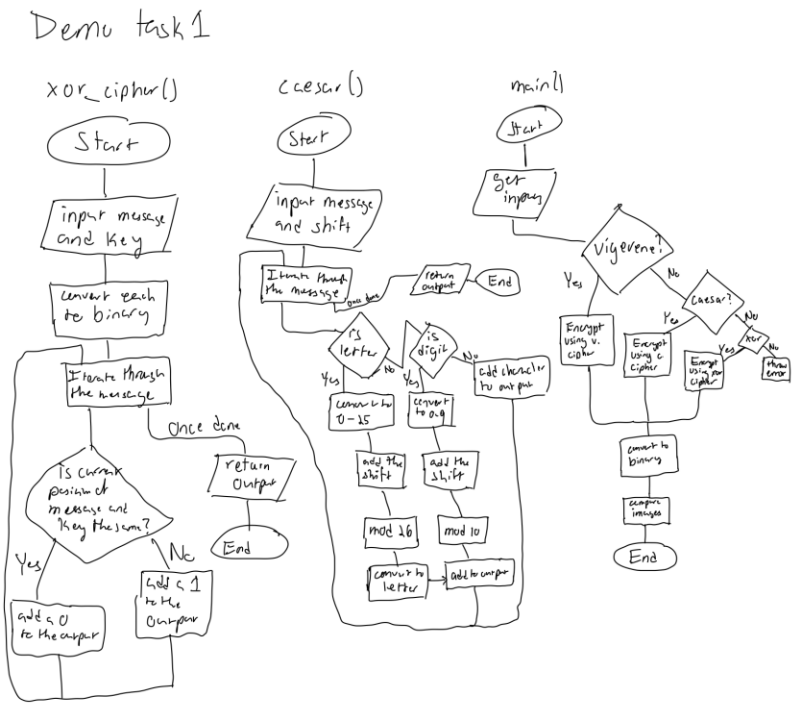


Second is Task 2, First is Task 1

Both Below are Task 3


start
Import matplotlib, PIL, and the load image function
Def analysis
Check if image is color; get 3 dimensions if true
Check if image is grayscale: get 2 dimensions if true
Check if dimensions are valid: if false print an error
Find a specific row and colum value in the image, divide it by 2 and take the remainder
Check if remainder is between 0 and 255
Check if the binary value if the same as the remainder; if the remainder is zero iterate up; if the remainder is not zero, iterate down
Plot and save figure
Def main
Input message to encrypt, cypher key, start and end sequences, image path, and encoded message path
Run def analysis
Run imported encryption function
Print outputs
end

Demo Task 1


Demo task 1
xor_cipher()
Start
input message and key
convert each to binary
Iterate through the message
Once done
return output
End
Is current position of message and key the same?
Yes
No
add a 0 to the output
add a 1 to the output
caesar()
Start
input message and shift
Iterate through the message
return output
End
is letter
is digit
Yes
No
convert to 0-25
convert to 0-9
add the shift
add the shift
mod 26
mod 10
convert to letter
add to output
add character to output
main()
Start
get inputs
Vigenere?
Yes
No
Caesar?
Yes
No
xor
Encrypt using v. cipher
Encrypt using c cipher
Encrypt using xor cipher
throw error
convert to binary
compare images
End

Demo Task 2

4. References

Anderson, Ross, and Markus Kuhn. "Low Cost Attacks on Tamper Resistant Devices." *Security Protocols*, Springer, 1996, pp. 125–136.

Gonzalez, Rafael C., and Richard E. Woods. *Digital Image Processing*. 4th ed., Pearson, 2017.

Jassim, Fadhil, and Abbas Ali. "Chaos-based Image Encryption Using Matrix Operations." *International Journal of Computer Applications*, vol. 69, no. 25, 2013, pp. 21-25.

5. Appendix

1. **User manual**

To operate this program input an image, the message you want to encrypt, the cipher you want to use and the key for the encryption. You then input the output image and the image you want to compare with. You will get shown the image with the encoded values and an image comparing the two images. It will also get saved to the file given

2. **Project management plan** *(*Team members' contribution*)*

In general we divided work in class. We each chose one of the tasks to complete. Then we completed the work and sent it to Ankur. Ankur then compiled the code and tested it and talked to any member with broken code to find a solution. For example, for task 3. Arjun completed the caesar cipher, Ramon completed the encoding image function, Grant completed the vigerene cipher, and Ankur completed the image comparison file.

3. **Discussion of design process** *(Approach to the design process)*

We followed the design process to create a Python program that encodes text into an image. After defining the requirements, we chose Least Significant Bit (LSB) encoding with the Pillow library for its simplicity and effectiveness. We developed and refined the prototype through testing, ensuring the program met all requirements, including maintaining image quality and reliable decoding..

4. **Code**

Tp1_team_1_11.py

```
import matplotlib.pyplot as plt
import numpy as np

def load_image(path):
    img = plt.imread(path) #read the image

    if int(np.max(img)) <= 1.0: #if the maximum is less than 1, it must be in
the range 0-1 not 0-255.
        img =img * 255 # multiply the float by 255
        img = img.astype(np.uint8) #convert back to integer

    return img

def main():
    path = input("Enter the path of the image you want to load:")
    img = load_image(path)

    plt.imshow(img, cmap= 'gray', vmin= 0, vmax= 255)
    plt.show()


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

Tp1_team_2_11.py

```
import matplotlib.pyplot as plt
import numpy as np
from tp1_team_1_11 import load_image

#converts string of text to binary
def to_binary(string):
```

```
    output=""
    for char in string:
        byte = bin(ord(char))[2:]
        output += "0"*(8-len(byte))+byte
    return output

# finds the LSD of each value in an image
def img_to_bin(img):
    output = ""
    isRGB = len(img.shape)==3 #if there are 3 dimensions its rgb not
grayscale
    for row in img:
        for item in row:
            values = item if isRGB else [item] #make a list with each value,
if rgb the list is already made, else make one item list
            for i in values: # for each value add to the output
                output += str(i%2) #add 0 if even (LSD is 0) or 1 if odd (LSD
is 1)
    return output

def findMessage(start, end, bitfield):
    sIndex = bitfield.find(start)
    eIndex = bitfield.find(end)
    if(sIndex==-1 or eIndex==-1):
        return None
    sIndex += len(start)
    return bitfield[sIndex:eIndex]

def processImage(path, start, end):
    img = load_image(path)
    output = img_to_bin(img)
    message = findMessage(start, end, output)
    if(message==None):
        return "Start or end sequence not found in the image."
    else:
        return f"Extracted Message: {message}"

def main():
    path = input("Enter the path of the image you want to load: ")
```

```
    start = to_binary(input("Enter the start sequence: "))
    end = to_binary(input("Enter the end sequence: "))
    print(processImage(path, start, end))

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

Tp1_team_3_11.py

```
def binToText(binary):
    message = ""
    for i in range(0, len(binary), 8): #take the binary message 8 digits at a
time
        group =  binary[i:i+8]
        decoeded = chr(int(group,2)) #convert to an integer in base 10 and
use ASCII table to convert to letter
        message += decoeded
    return message

def main():
    x = input("Enter the binary message: ")
    print(binToText(x))

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

Tp2_team_1_11.py

```
from matplotlib import pyplot as plt
import numpy as np

from tp1_team_1_11 import load_image
#copied from tp1_team_2_11.py
# finds the LSD of each value in an image
def img_to_bin(img):
```

```
    output = ""
    isRGB = len(img.shape)==3 #if there are 3 dimensions its rgb not
grayscale

    if(isRGB and img.shape[2]==4):
        img = np.delete(img, 3, 2)
    for row in img:
        for item in row:
            values = item if isRGB else [item] #make a list with each value,
if rgb the list is already made, else make one item list
            for i in values: # for each value add to the output
                output += str(i%2) #add 0 if even (LSD is 0) or 1 if odd (LSD
is 1)
    return output


def compare_images(img1, img2, resultPath):
    different = False
    imgO = []
    img1_bin = img_to_bin(img1)#converts both images to binary
    img2_bin = img_to_bin(img2)
    diff_bin = ""
    for i in range(len(img1_bin)):
        diff_bin += str(int(img1_bin[i]!=img2_bin[i])) #returns 1 if
different
    isRGB = img1.ndim==3

    counter = 0
    for i in range(img1.shape[0]):
        row = []
        for j in range(img1.shape[1]): #iterate through image
            pixel = []
            if(isRGB): #if its an rgb image
                for k in range(3):
                    different = different if diff_bin[counter]=="0" else True
#if this pixel is 0, keep it the same, otherwise change to True. This will
change it to be True if any pixel is different
                    pixel.append(int(diff_bin[counter])*255)#if the pixel is
different, add 255 otherwise add 0
```

```
                    counter+=1#increment counter to keep the place
                row.append(pixel)
            else:
                row.append(int(diff_bin[counter])*255)#if the pixel is
different, add 255 otherwise add 0
                different = different if diff_bin[counter]=="0" else True
                counter +=1

        imgO.append(row)

    plt.imshow(imgO, cmap='gray', vmin= 0, vmax= 255)#show the difference map
    if(resultPath!=None):
        plt.savefig(resultPath)#save the image if requested
    plt.show()
    return different

def main():
    path1 = "output.png"
    path2 = "bear_col_020_11_v.png"
    pathDiff = "diff.png"
    # path1 = input("Enter the path of your first image: ")
    # path2 = input("Enter the path of your second image: ")
    # pathDiff = input("Enter the path for the output image: ")
    img1 = load_image(path1)
    img2 = load_image(path2)
    if img1.ndim != img2.ndim or img1.shape[0] != img2.shape[0] or
img1.shape[1] != img2.shape[1]:
        print("Cannot compare images in different modes (RGBA and L) or of
different sizes.")
        print(img1.shape, img2.shape)
        return
    if compare_images(img1, img2, pathDiff):
        print("The images are different.")
    else:
        print("The images are the same.")


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

Tp2_team_2_11.py

```
def v_encrypt(text, u_key):
    key_value = []
    u_key = u_key.upper()
    #reading the text and making it the number

    a = len(text)

    #making the shift vaules into list, then using while to go through the
values
    for c in u_key:
            key_ord = ord(c)
            shift = (key_ord - 65)
            key_value.append(shift)
    # print(key_value)
    #main for loop that goes through all of the characters in the string
    new_message = ""
    for i in range(a):
        l = text[i]
        u = i % len(key_value)
        text_ord = ord(l) #cagegorizes what section it should go into
        if l.isupper():
            new_text = (text_ord - 65)
            new_value = new_text + key_value[i % len(key_value)]
            new_value = new_value % 26
            new_message += chr(new_value + 65)
        elif l.islower():
            new_text = (text_ord - 97)
            new_value = new_text + key_value[i % len(key_value)]
            new_value = new_value % 26
            new_message += chr(new_value + 97)
        elif l.isdigit():
            new_text = int(l)
            new_value = new_text + key_value[i % len(key_value)]
```

```
            new_value = new_value % 10
            new_message += str(new_value)
        else:
            new_message += l
        # print(i, l, new_text, u_key[u:u+1], key_value[i % len(key_value)],
chr(new_value + 97))
    return new_message



#converts string of text to binary
#From Checkpoint 1 task 2
def to_binary(string):
    output=""
    for char in string: #for each character
        byte = bin(ord(char))[2:] #convert the character to the ASCII value
and then to binary
        # when converting to binary it gives you exactly the number of digits
with "0b" before
        output += "0"*(8-len(byte))+byte #add the missing zeros
        output += " " # add a space after each byte for readability
    return output




def main():
    # text = input("Enter the plaintext you want to encrypt: ")
    # key = input("Enter the key for Vigenere cipher: ")
    # start_seq = input("Enter the start sequence: ")
    # end_seq = input("Enter the end sequence: ")
    text = "Team 11 is praiseworthy!!!"
    key = "RItwKtCnrV"
    start_seq = "117"
    end_seq = "711"

    new_message = v_encrypt(text, key)
    print(f"Encrypted Message using Vigenere Cipher: {new_message}")
    binary = to_binary(start_seq + new_message + end_seq)
```

```
    print(f"Binary output message: {binary}")

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

Tp2_team_3_11.py

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

from tp2_team_1_11 import load_image
from tp2_team_2_11 import v_encrypt, to_binary

#see tp3_team_1_11.py for full comments
def encodeImage(binary_message,image_path,output_path):
    binary_message = binary_message.replace(" ","") #remove spaces from
binary message
    image= load_image(image_path)
    if image.ndim==2: #if grayscale
        x= image.shape[0]*image.shape[1] #total number of encodable bits is
equal to width*height
        if len(binary_message)> x:#if its too long return error
            print("Given message is too long to be encoded in the image")
            return
        pos=0 #keep track of which position you are at
        for i in range(len(image)):
            if(pos>=len(binary_message)):
                break
            for j in range(len(image[i])):
                if(pos>=len(binary_message)):
                    break
                value=image[i][j]
                y=value % 2
                bin_value=binary_message[pos:pos+1]
                pos+=1
                if bin_value==y:
```

```
                continue
            if y==0:
                image[i,j]+=1
            else:
                image[i,j]-=1
    pil_image = Image.fromarray(image, mode="L")
    pil_image.save(output_path)
else:
    x= image.shape[0]*image.shape[1]*image.shape[2]
    if len(binary_message)> x:
        print("Given message is too long to be encoded in the image")
        return
    pos=0
    for i in range(len(image)):
        if(pos>=len(binary_message)):
            break
        for j in range(len(image[i])):
            if(pos>=len(binary_message)):
                break
            for k in  range(len(image[i,j])):
                if(pos>=len(binary_message)):
                    break
                value =image[i][j][k]
                y=value % 2
                bin_value=binary_message[pos:pos+1]
                # print(bin_value, y, image[i,j,k])
                pos+=1
                if int(bin_value)==y:
                    continue
                if y==0:
                    image[i,j,k]+=1
                else:
                    image[i,j,k]-=1
                print(bin_value, image[i,j,k])

    pil_image = Image.fromarray(image)
    pil_image.save(output_path)
print(f"Message successfully encoded and saved to: {output_path}")
```

```
def main():
    message = input("Enter the plaintext you want to encrypt: ")
    key = input("Enter the key for Vigenere cipher: ")
    start_seq = input("Enter the start sequence: ")
    end_seq = input("Enter the end sequence: ")
    imagePath = input("Enter the path of the image: ")
    outputPath = input("Enter the path for the encoded image: ")



    encrypted = v_encrypt(message, key)
    print(f"Encrypted Message using Vigenere Cipher: {encrypted}")
    binary = to_binary(start_seq + encrypted + end_seq)
    print(f"Binary output message: {binary}")

    encodeImage(binary, imagePath, outputPath)

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

Tp3_team_1_11.py

```
from tp1_team_3_11 import binToText
from tp2_team_1_11 import compare_images, img_to_bin, load_image
from tp2_team_2_11 import to_binary, v_encrypt
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

# @author Arjun
# encrypt with the caesar cipher
def c_encrypt(message, shift):
    shift = int(shift) #make sure the shift is a number
    encrypted_text = ""

    for char in message:
        if char.isalpha(): #if its a letter
```

```
            shift_base = ord('A') if char.isupper() else ord('a') #subtract
the ord value for Capital A or lowercase a if it is upper or lowercase
respectivly
            encrypted_text += chr((ord(char) - shift_base + shift) % 26 +
shift_base) # add the shift and mod 26 to ensure it is still a letter, and
convert back to letter
        elif char.isdigit(): #if its a number
            encrypted_text += str((int(char)+shift)%10) #convert to a number
and add the shift. Then mod 10 to ensure it is still one digit
        else:
            encrypted_text += char  # Keep non-alphabetical characters
unchanged (punctuation, spaces)

    return encrypted_text

def xor(a, b):
    return str(int(a != b)) #if they are the same, return 0, else return 1

#from tp2_team_3_11.py but added offset
def encodeImage(binary_message,image_path,output_path, offset):
    binary_message = "0"*offset+ binary_message.replace(" ","")
    image= load_image(image_path)
    length = len(binary_message)
    if image.ndim==2:
        x= image.shape[0]*image.shape[1] #number of bits (width*height)
        if length> x: #if its too long return error
            print("Given message is too long to be encoded in the image")
            return
        pos=0 #keep track of which position you are at
        for i in range(len(image)):
            if(pos>=length):
                break
            for j in range(len(image[i])):
                if(pos>=length):
                    break
                if(pos<offset): # if the current position is less than the
offset
                    pos+=1
                    continue
```

```
                value=image[i][j] #current pixel value
                y=value % 2 #LSB of current pixel
                bin_value=int(binary_message[pos:pos+1]) #find the value of
the current position
                pos+=1 #increment position
                if bin_value==y: #if it already matches move on
                    continue
                if y==0: #if the current value is even, it could be 0.
                    image[i,j]+=1 #you must add one to make it odd, so it
doesn't become negative
                else: #if the current value is odd, it could be 255.
                    image[i,j]-=1 # you must subtract one to make it even so
it doesn't overflow
        pil_image = Image.fromarray(image, mode="L") #save image
        pil_image.save(output_path)
    else:
        x= image.shape[0]*image.shape[1]*3 # number of bits (width*height*3)
        if length> x:
            print("Given message is too long to be encoded in the image")
            return
        pos=0
        for i in range(len(image)):
            if(pos>=length): #if message is done, exit
                break
            for j in range(len(image[i])):
                if(pos>=length):
                    break
                for k in  range(3):
                    if(pos<offset): #if the offset isnt over, pass
                        pos+=1
                        continue
                    if(pos>=length):
                        break
                    if(pos<offset): # if the current position is less than
the offset
                        pos+=1
                        continue
                    value=image[i][j][k] #current pixel value
                    y=value % 2 #LSB of current pixel
```

```
                    bin_value=int(binary_message[pos:pos+1]) #find the value
of the current position
                    pos+=1 #increment position
                    if bin_value==y: #if it already matches move on
                        continue
                    if y==0: #if the current value is even, it could be 0.
                        image[i,j,k]+=1 #you must add one to make it odd, so
it doesn't become negative
                    else: #if the current value is odd, it could be 255.
                        image[i,j,k]-=1 # you must subtract one to make it
even so it doesn't overflow

        pil_image = Image.fromarray(image)
        pil_image.save(output_path)
    print(f"Message successfully encoded and saved to: {output_path}")

def xor_cypher(message, key):
    m_bin = to_binary(message).replace(" ", "")
    k_bin = to_binary(key).replace(" ", "") #convert to binary and remove
spaces
    output = ""
    counter = 0
    for i in range(len(m_bin)):
        output += xor(m_bin[i], k_bin[i%len(k_bin)]) # perform xor operation
        if counter == 7:# add a space every 8 characters
            counter = 0
            output+= " "
        else:
            counter+=1
    return output


def main():
    cipher = input("Enter the cipher you want to use for encryption: ")
    message = input("Enter the plaintext you want to encrypt: ")
    key = input("Enter the key for the cipher: ")
    start_seq = to_binary(input("Enter the start sequence: "))
    end_seq = to_binary(input("Enter the end sequence: "))
```

```
    offset = int(input("Enter the bit offset before you want to start
encoding: "))
    input_path = input("Enter the path of the input image: ")
    output_path = input("Enter the path for your encoded image: ")
    comp_path = input("Enter the path of the image you want to compare: ")

    encrypted = None
    match cipher:
        case "vigenere":
            encrypted = v_encrypt(message, key) #encrypt with vigenere
            binary = start_seq + to_binary(encrypted) + end_seq #convert to
binary and add the start and end sequences
            print(f"Encrypted Message using Vigenere Cipher: {encrypted}")
        case "caesar":
            encrypted = c_encrypt(message, key) #encrypt with caesar
            binary = start_seq + to_binary(encrypted) + end_seq #convert to
binary and add the start and end sequences
            print(f"Encrypted Message using Caesar Cipher: {encrypted}")
        case "xor":
            binary = xor_cypher(message, key) #encrypt with xor (returns
binary)
            encrypted = binToText(binary.replace(" ","")) #convert back to
text
            binary = start_seq + binary + end_seq #add start and end
sequences
            print(f"Encrypted Message using XOR Cipher: {encrypted}")
        case _:
            print("error")
            return

    print(f"Binary output message: {binary}")
    encodeImage(binary, input_path, output_path, offset) #encode the binary
into the image

    #compare images
    img1 = load_image(output_path)
    plt.imshow(img1)
    plt.show()
    img2 = load_image(comp_path)
```

```
    if img1.ndim != img2.ndim or img1.shape[0] != img2.shape[0] or
img1.shape[1] != img2.shape[1]: #if they are different sizes
        print("Cannot compare images in different modes (RGBA and L) or of
different sizes.")
        print(img1.shape, img2.shape)
        return
    if compare_images(img1, img2, None): #compare them, if it returns true
they are different
        print("The images are different.")
    else:
        print("The images are the same.")
```

Tp3_team_2_11.py

```
from tp1_team_2_11 import processImage
from tp1_team_3_11 import binToText
from tp2_team_1_11 import compare_images, load_image
from tp2_team_2_11 import to_binary, v_encrypt
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

from tp3_team_1_11 import xor, xor_cypher



# encrypt with the caesar cipher
def c_decrypt(message, shift):
    shift = int(shift) #make sure the shift is a number
    encrypted_text = ""

    for char in message:
        if char.isalpha(): #if its a letter
            shift_base = ord('A') if char.isupper() else ord('a') #subtract
the ord value for Capital A or lowercase a if it is upper or lowercase
respectivly
```

```
            encrypted_text += chr((ord(char) - shift_base - shift) % 26 +
shift_base) # add the shift and mod 26 to ensure it is still a letter, and
convert back to letter
        elif char.isdigit(): #if its a number
            encrypted_text += str((int(char)-shift)%10) #convert to a number
and add the shift. Then mod 10 to ensure it is still one digit
        else:
            encrypted_text += char  # Keep non-alphabetical characters
unchanged (punctuation, spaces)

    return encrypted_text

def v_decrypt(text, u_key):
    key_value = []
    u_key = u_key.upper() #should be all uppercase

    #making the shift vaules into list, then using while to go through the
values
    for c in u_key:
            key_ord = ord(c)
            shift = (key_ord - 65)
            key_value.append(shift)
    # print(key_value)
    #main for loop that goes through all of the characters in the string
    new_message = ""
    for i in range(len(text)):
        l = text[i]
        u = i % len(key_value)
        text_ord = ord(l) #cagegorizes what section it should go into
        if l.isupper():
            new_text = (text_ord - 65) #uppercase A in ASCII is 65
            new_value = new_text - key_value[u] #shift by the key value
            new_value = new_value % 26 # mod 26 to keep it a letter
            new_message += chr(new_value + 65) # convert back to a character
        elif l.islower():
            new_text = (text_ord - 97) #lowercase a in ASCII is 97
            new_value = new_text - key_value[u] #shift by the key value
            new_value = new_value % 26 # mod 26 to keep it a letter
            new_message += chr(new_value + 97) # convert back to a character
```

```
        elif l.isdigit():
            new_text = int(l) #convert to a number
            new_value = new_text - key_value[u] #shift by the key value
            new_value = new_value % 10 # mod 10 to keep it a single digit
            new_message += str(new_value) #convert back to string
        else:
            new_message += l #if its not a letter/number don't change it
    return new_message


def main():
    cipher = input("Enter the cipher you want to use for encryption: ")
    key = input("Enter the key for the cipher: ")
    start = to_binary(input("Enter the start sequence: ")).replace(" ","")
    end = to_binary(input("Enter the end sequence: ")).replace(" ","")
    path = input("Enter the path of the input image: ")

    #converts to binary
    output = processImage(path, start, end) #cconverts image to binary,
removing the start and end sequence
    if "0" in output or "1" in output: #if there is a 0 or 1 it must contain
binary
        binary = output[19:]
    else:
        return
    print(f"Extracted Binary Message: {binary}")


    encrypted = binToText(binary) #convert back to text
    print(f"Converted Binary Text: {encrypted}")
    match cipher:
        case "vigenere":
            message = v_decrypt(encrypted, key) # decrypt text using key
        case "caesar":
            message = c_decrypt(encrypted, key) # decrypt text using key
        case "xor":
            message = binToText(xor_cypher(encrypted, key).replace(" ",""))
#decrypt, returns binary, so convert back
        case _:
```

```
            print("error")
            return

    print(f"Converted text: {message}")

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