Experiment 1

Traditional Crypto Methods and Key Exchange

Name: Jash Jain
UID: 2019130021
Class: TE Comps

Objective:
To implement Substitution, ROT 13, Transposition, Double Transposition, and Vernam Cipher in Python.

Theory:
Cryptography
Cryptography is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents. The term is derived from the Greek word kryptos, which means hidden. It is closely associated with encryption, which is the act of scrambling ordinary text into what's known as ciphertext and then back again upon arrival. In addition, cryptography also covers the obfuscation of information in images using techniques such as microdots or merging. Ancient Egyptians were known to use these methods in complex hieroglyphics, and Roman Emperor Julius Caesar is credited with using one of the first modern ciphers.

Substitution Technique
A substitution cipher is a type of encryption in which plaintext units are replaced with ciphertext according to a set of rules; the "units" might be single letters, pairs of letters, triplets of letters, combinations of the aforementioned, and so on. The text is decoded by the receiver via inverse substitution.

Substitution ciphers come in a variety of shapes and sizes. A basic substitution cipher is one that acts on single letters; a polygraphic cipher is one that operates on bigger groupings of letters. A monoalphabetic cipher employs a single substitution throughout the message, but a polyalphabetic cipher uses several replacements throughout the message, where a unit from the plaintext is mapped to one of the numerous ciphertext options and vice versa.

ROT13
ROT13 ("rotate by 13 places", sometimes hyphenated ROT-13) is a simple letter substitution cipher that replaces a letter with the 13th letter after it in the alphabet. Because there are 26 letters (2×13) in the basic Latin alphabet, ROT13 is its own inverse; that is, to undo ROT13, the same algorithm is applied, so the same action can be used for encoding and decoding. The algorithm provides virtually no cryptographic security and is often cited as a canonical example of weak encryption.

Transposition Technique
A transposition cipher is an encryption method in which the locations of plaintext units (often letters or groups of characters) are moved according to a regular scheme, resulting in the ciphertext being a permutation of the plaintext. That is, the units' order is altered. To encrypt, a bijective function is applied to the locations of the characters, and to decrypt, an inverse function is used.

In a columnar transposition, the message is typed down in fixed-length rows, then read out column by column, with the columns picked in a random sequence. A keyword is generally used to describe the width of the rows and the permutation of the columns. The word ZEBRAS, for example, is six letters long (thus the rows are six letters long), and the permutation is determined by the alphabetical order of the letters in the keyword. The sequence would be "6 3 2 4 1 5" in this example.

Any spare spaces in a normal columnar transposition cipher are filled with nulls, but the spaces in an irregular columnar transposition cipher are left blank. Finally, the message is read off in columns according to the keyword's sequence.

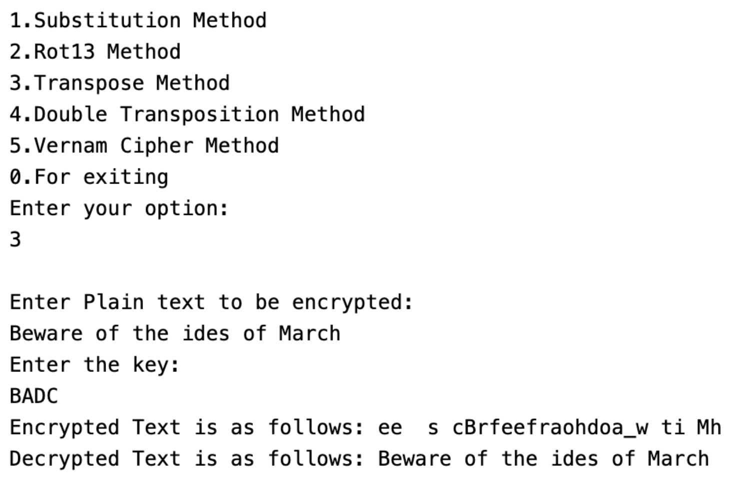
Double Transposition
A single columnar transposition may be attacked by guessing potential column lengths, putting the message down in columns (in the wrong order, because the key is unknown), and then hunting for anagrams. As a result, a double transposition was frequently employed to strengthen it. It's just a columnar transposition done twice. Both transpositions can be done with the same key, or with two distinct keys.

Vernam Cipher
A Vernam cipher is a symmetrical stream cipher that generates the ciphertext by XORing the plaintext with a random or pseudorandom stream of data (the "keystream") of the same length. This is basically a one-time pad if the keystream is genuinely random and used just once. A popular and successful construction for a stream cipher is substituting pseudorandom data generated by a cryptographically safe pseudo-random number generator.

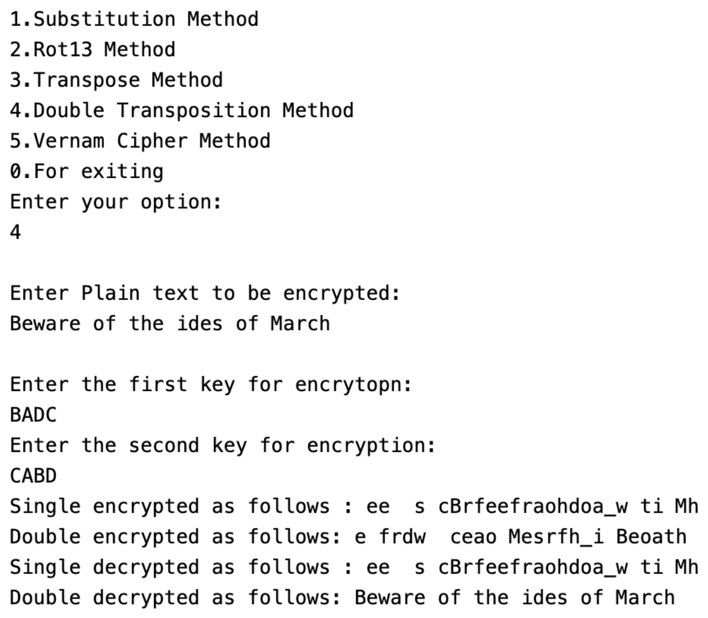
Code:
```
import math
import numpy as np

def chooseOption(i,Text):
    switcher = {
        1: substitution,
        2: rot13,
        3: transpose,
        4: double_transposition,
        5: vernam_cipher
    }
    switcher[i](Text)

def vernam_cipher(Text):

    key = input('Enter the key(NOTE: to be of the same length as the message):')
    while(len(key)!=len(Text)):
        print("Please enter the key of the same")
        key = input('Enter the key(NOTE: to be of the same length as the message):')


    encryptedText = ''
    for i in range(len(Text)):
        if Text[i] == ' ':
            encryptedText+=' '
        else:
            encryptedText += chr(((ord(Text[i])-65)^(ord(key[i])-65))+65)
    print('encrypted Text is as follows :',encryptedText)


    decryptedText = ''
    for i in range(len(encryptedText)):
        if Text[i]== ' ':
            decryptedText+=' '
        else:
            decryptedText += chr(((ord(encryptedText[i]) - 65)^(ord(key[i]) - 65)) + 65)
    print('decrypted Text is as follows :',decryptedText)
    return

def substitution(Text):
    posShift = int(input('Enter the no. of Position shift: '))

    encryptedText = ''
    for char in Text:
        #check if its capital or small
        if(char.isupper()):
            encryptedText += chr((ord(char) + posShift-65) % 26 + 65)
        else:
            if char == ' ':
                encryptedText += ' '
            else:
                encryptedText += chr((ord(char) + posShift-97) % 26 + 97)
    print('encrypted Text is as follows :',encryptedText)

    decryptedText = ''
    for char in encryptedText:
        #check if its capital or small
        if(char.isupper()):
            decryptedText += chr((ord(char) - posShift-65) % 26 + 65)
        else:
            if char == ' ':
                decryptedText += ' '
            else:
                decryptedText += chr((ord(char) - posShift-97) % 26 + 97)
    print('decrypted Text is as follows :',decryptedText)
    return

def transpose(plainText):
    key = input('Enter the key: ')
    key.upper()
    order = sorted(list(key))
    col = len(key)
    # Encryption
    msg_len = len(plainText)
    msg_list = list(plainText)
    row = int(math.ceil(msg_len/col))
    null_values = row*col - msg_len
    msg_list.extend('_'*null_values)
    matrix = np.array(msg_list).reshape(row,col)
    encryptedText = ''

    for i in range(col):
        index = key.index(order[i])
        encryptedText += ''.join([row[index] for row in matrix])
    print('Encrypted Text is as follows:',encryptedText)

    # Decryption
    encryptedText_lst = list(encryptedText)
    decryptedText = ''
    pointer = 0
    dec_matrix = np.array([None]*len(encryptedText)).reshape(row,col)
    for i in range(col):
        index = key.index(order[i])
        for j in range(row):
            dec_matrix[j,index] = encryptedText_lst[pointer]
            pointer += 1
    decryptedText = ''.join(''.join(col for col in row) for row in dec_matrix)
    decryptedText = decryptedText[:-decryptedText.count('_')]
    print('Decrypted Text is as follows:',decryptedText)
    return
def rot13(Text):
    # 13 is the shift (predefined)
    encryptedText = ''

    for char in Text:
        if(char.isupper()):
            encryptedText += chr((ord(char) + 13 - 65) % 26 + 65)
        else:
            if char == ' ':
                encryptedText+=' '
            else:
                encryptedText += chr((ord(char) + 13 - 97) % 26 + 97)
    print('encrypted Text is as follows :',encryptedText)


    decryptedText = ''

    for char in encryptedText:
        if(char.isupper()):
            decryptedText += chr((ord(char) + 13 - 65) % 26 + 65)
        else:
            if char == ' ':
                decryptedText += ' '
            else:
                decryptedText += chr((ord(char) + 13 - 97) % 26 + 97)
    print('decrypted Text is as follows :',decryptedText)
    return
def double_transposition(plainText):
    key1 = input('\nEnter the first key for encrytopn: ')
    key2 = input('Enter the second key for encryption: ')
    key1.upper()
    key2.upper()
    order1 = sorted(list(key1))
    order2 = sorted(list(key2))
    col1 = len(key1)
    col2 = len(key2)

    ## Encryption
    msg_len = len(plainText)
    msg_list = list(plainText)
    row1 = int(math.ceil(msg_len/col1))
    null_values1 = row1*col1 - msg_len
    msg_list.extend('_'*null_values1)
    matrix = np.array(msg_list).reshape(row1,col1)
    middleText,encryptedText = '',''

    for i in range(col1):
        index = key1.index(order1[i])
        middleText += ''.join([row1[index] for row1 in matrix])
    print("Single encrypted as follows :",middleText)

    middle_msg_len = len(middleText)
    middle_msg_list = list(middleText)
    row2 = int(math.ceil(middle_msg_len/col2))
    null_values2 = row2*col2 - middle_msg_len
    middle_msg_list.extend('_'*null_values2)
    matrix = np.array(middle_msg_list).reshape(row2,col2)

    for i in range(col2):
        index = key2.index(order2[i])
        encryptedText += ''.join([row2[index] for row2 in matrix])
    print('Double encrypted as follows:',encryptedText)

    ## Decryption
    encryptedText_lst = list(encryptedText)
    middleText,decryptedText = '',''
    pointer = 0
    dec_matrix = np.array([None]*len(encryptedText)).reshape(row2,col2)
    for i in range(col2):
        index = key2.index(order2[i])
        for j in range(row2):
            dec_matrix[j,index] = encryptedText_lst[pointer]
            pointer += 1

    middleText = ''.join(''.join(col for col in row) for row in dec_matrix)
    if null_values2 > 0:
        middleText = middleText[:-null_values2]
    pointer = 0
    print('Single decrypted as follows :',middleText)

    middletxt_lst = list(middleText)
    dec_matrix = np.array([None]*len(middleText)).reshape(row1,col1)
    for i in range(col1):
        index = key1.index(order1[i])
        for j in range(row1):
            dec_matrix[j,index] = middletxt_lst[pointer]
            pointer += 1
    if null_values1 > 0:
        decryptedText = decryptedText[:-decryptedText.count('_')]
    decryptedText = ''.join(''.join(col for col in row) for row in dec_matrix)
    decryptedText = decryptedText[:-decryptedText.count('_')]
    print('Double decrypted as follows:',decryptedText)
    return
while(True):
    print("1.Substitution Method\n2.Rot13 Method\n3.Transpose Method\n4.Double Transposition
Method\n5.Vernam Cipher Method\n0.For exiting")
    choice = int(input("Enter your option: "))
    if choice ==0:
        break
    Text = input('\nEnter Plain text to be encrypted: ')
    chooseOption(choice,Text)
print("Thanks for encrypting decrypting")
```

Screenshots:

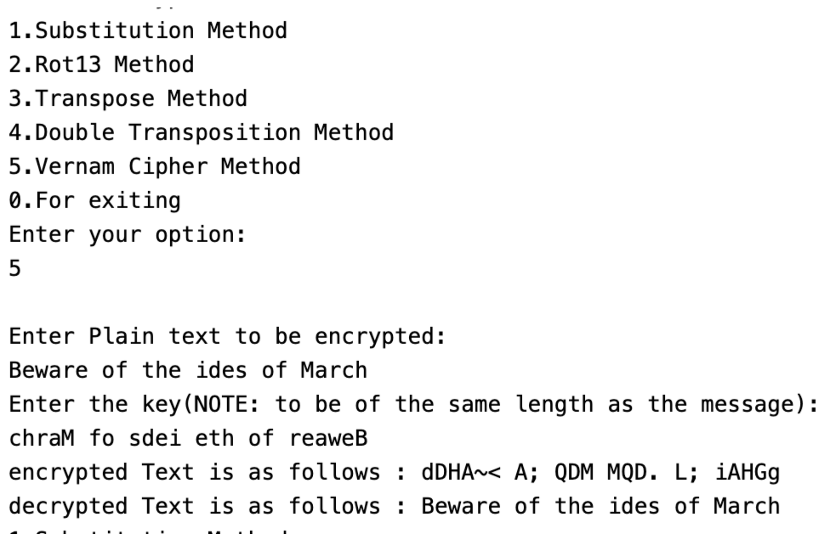
```
1.Substitution Method
2.Rot13 Method
3.Transpose Method
4.Double Transposition Method
5.Vernam Cipher Method
0.For exiting
Enter your option:
1

Enter Plain text to be encrypted:
Beware of the ides of March
Enter the no. of Position shift:
5
encrypted Text is as follows : Gjbfwj tk ymj nijx tk Rfwhm
decrypted Text is as follows : Beware of the ides of March
1.Substitution Method
2.Rot13 Method
3.Transpose Method
4.Double Transposition Method
5.Vernam Cipher Method
0.For exiting
Enter your option:
2

Enter Plain text to be encrypted:
Beware of the ides of March
encrypted Text is as follows : Orjner bs gur vqrf bs Znepu
decrypted Text is as follows : Beware of the ides of March
```



```
1.Substitution Method
2.Rot13 Method
3.Transpose Method
4.Double Transposition Method
5.Vernam Cipher Method
0.For exiting
Enter your option:
3

Enter Plain text to be encrypted:
Beware of the ides of March
Enter the key:
BADC
Encrypted Text is as follows: ee  s cBrfeefraohdoa_w ti Mh
Decrypted Text is as follows: Beware of the ides of March
```



```
1.Substitution Method
2.Rot13 Method
3.Transpose Method
4.Double Transposition Method
5.Vernam Cipher Method
0.For exiting
Enter your option:
4

Enter Plain text to be encrypted:
Beware of the ides of March

Enter the first key for encrytopn:
BADC
Enter the second key for encryption:
CABD
Single encrypted as follows : ee  s cBrfeefraohdoa_w ti Mh
Double encrypted as follows: e frdw  ceao Mesrfh_i Beoath
Single decrypted as follows : ee  s cBrfeefraohdoa_w ti Mh
Double decrypted as follows: Beware of the ides of March
```



```
1.Substitution Method
2.Rot13 Method
3.Transpose Method
4.Double Transposition Method
5.Vernam Cipher Method
0.For exiting
Enter your option:
5

Enter Plain text to be encrypted:
Beware of the ides of March
Enter the key(NOTE: to be of the same length as the message):
chraM fo sdei eth of reaweB
encrypted Text is as follows : dDHA~< A; QDM MQD. L; iAHGg
decrypted Text is as follows : Beware of the ides of March
```

Conclusion:

1. In this experiment, I understood the need to test for long sentences with upper as well as lower cases else the program crashes or gives abnormal output if not tested and

corrected(the reason being the code was written previously only for ASCII values of lowercase which was later corrected).

2. Another such thing happened in terms of usage of space bar which on decrypting was giving errored outputs which were then corrected(The space bar was getting converted to ‘q’ in rot13 and on decrypting al other q’s were also getting converted to spaces which was then later corrected).
3. Lastly while storing the values in the matrix I had not padded the remaining spaces with ‘ ‘ and hence the program wasn’t working which was modified later.