Experiment 1 & 2

Traditional Crypto Methods and Key Exchange

Name: Kashish Jain
UID: 2019130022
Class: TE Comps

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

Theory:

Cryptography

Encryption is the process of changing information (referred to as plaintext) using an algorithm (known to as a cypher) to make it unreadable to everyone save those with specific knowledge, generally referred to as a key, in cryptography. The procedure produces encrypted data therefore (in cryptography, referred to as cypher text). In many situations, the term encryption also implies the reverse process of decryption (e.g., “software for encryption” may generally also do decryption), which restores the readable state of the encrypted data (i.e. to make it unencrypted). Encryption is used to secure data in transit, such as data sent through networks (e.g., the Internet, e-commerce), mobile phones, wireless microphones, wireless intercom systems, Bluetooth devices, and bank ATMs. In recent years, there have been several cases of data in transit being intercepted. Because it is frequently difficult to physically safeguard all network access, encrypting data in transit also helps to secure it.

Substitution Technique

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

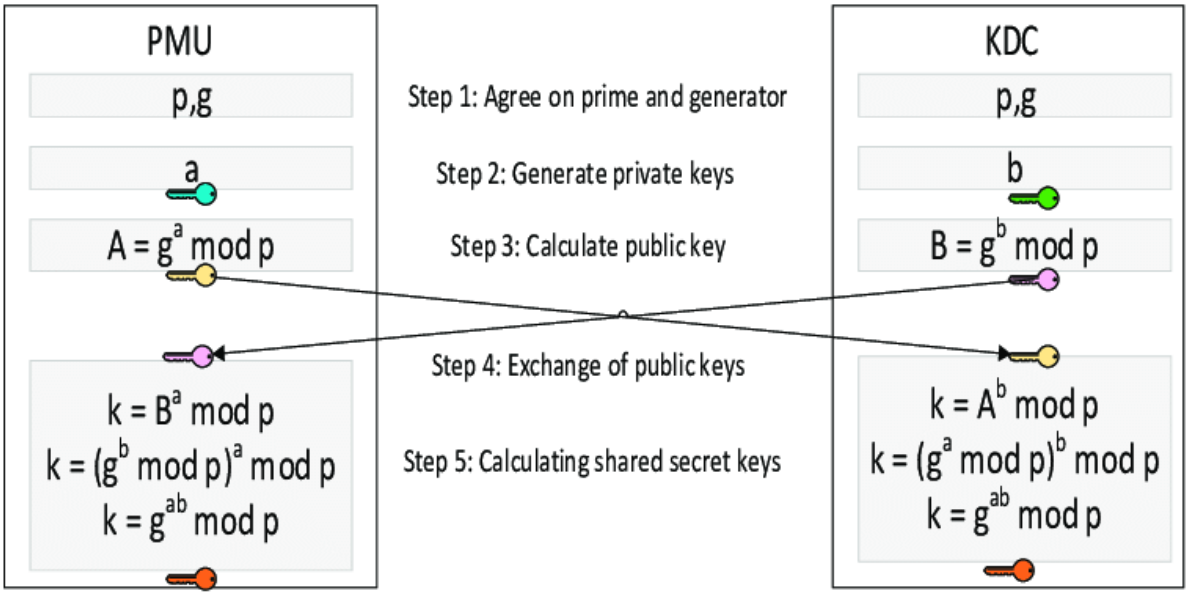
Any spare spaces in a normal columnar transposition cypher are filled with nulls, but the spaces in an irregular columnar transposition cypher are left blank. Finally, the message is read off in columns according to the keyword's sequence.

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 cypher is a symmetrical stream cypher 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 cypher is substituting pseudorandom data generated by a cryptographically safe pseudo-random number generator.



Code:

```
import math
import numpy as np

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

    # Encryption
    encryptedText = ''
    for char in plainText:
        if(char.isupper()):
            encryptedText += chr((ord(char) + shift-65) % 26 + 65)
        else:
            encryptedText += chr((ord(char) + shift-97) % 26 + 97)
    print('Encrypted Text:',encryptedText)

    # Decryption
    decryptedText = ''
    for char in encryptedText:
        if(char.isupper()):
            decryptedText += chr((ord(char) - shift-65) % 26 + 65)
        else:
            decryptedText += chr((ord(char) - shift-97) % 26 + 97)
    print('Decrypted Text:',decryptedText)

def rot13(plainText):
    # 13 is the shift

    # Encryption
    encryptedText = ''
```

```
    for char in plainText:
        if(char.isupper()):
            encryptedText += chr((ord(char) + 13 - 65) % 26 + 65)
        else:
            encryptedText += chr((ord(char) + 13 - 97) % 26 + 97)
    print('Encrypted Text:',encryptedText)

    # Decryption
    decryptedText = ''
    for char in encryptedText:
        if(char.isupper()):
            decryptedText += chr((ord(char) + 13 - 65) % 26 + 65)
        else:
            decryptedText += chr((ord(char) + 13 - 97) % 26 + 97)
    print('Decrypted Text:',decryptedText)

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:',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:',decryptedText)

def double_transposition(plainText):
    key1 = input('\nEnter the first key: ')
```

```
key2 = input('Enter the second key: ')
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 Encryption:",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 Encryption:',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)
middleText = middleText[:-null_values2]

pointer = 0
print('Single Decryption:',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
    decryptedText = decryptedText[:-decryptedText.count('_')]
    decryptedText = ''.join(''.join(col for col in row) for row in dec_matrix)
    decryptedText = decryptedText[:-decryptedText.count('_')]
    print('Double Decryption:',decryptedText)


def vernam_cipher(plainText):
    key = input('Enter the key(of same length as the message):')
    while(len(key)!=len(plainText)):
        print("Please enter the key of the same")
        key = input('Enter the key(of same length as the message):')

    # Encryption
    encryptedText = ''
    for i in range(len(plainText)):
        encryptedText += chr(((ord(plainText[i])-65)^(ord(key[i])-65))+65)
    print('Encrypted Text:',encryptedText)

    # Decryption
    decryptedText = ''
    for i in range(len(encryptedText)):
        decryptedText += chr(((ord(encryptedText[i]) - 65)^(ord(key[i]) - 65)) + 65)
    print('Decrypted Text:',decryptedText)

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

options = """1. Substitution
2. ROT 13
3. Transpose
4. Double Transposition
5. Vernam Cipher"""
print(options)
choice = int(input("Enter your choice: "))
plainText = input('\nEnter Plain text to be encrypted: ')
METHOD(choice,plainText)
```

Output:

Substitution

```
1. Substitution
2. ROT 13
3. Transpose
4. Double Transposition
5. Vernam Cipher
Enter your choice: 1

Enter Plain text to be encrypted: HELLO
Enter the no. of Position shift: 15
Encrypted Text: WTAAD
Decrypted Text: HELLO
```

ROT 13

```
1. Substitution
2. ROT 13
3. Transpose
4. Double Transposition
5. Vernam Cipher
Enter your choice: 2

Enter Plain text to be encrypted: HELLO
Encrypted Text: URYYB
Decrypted Text: HELLO
```

Transpose

```
1. Substitution
2. ROT 13
3. Transpose
4. Double Transposition
5. Vernam Cipher
Enter your choice: 3

Enter Plain text to be encrypted: HELLO
Enter the key:BLUE
Encrypted Text: HOL_E_L_
Decrypted Text: HELLO
```

Double Transposition

```
1. Substitution
2. ROT 13
3. Transpose
4. Double Transposition
5. Vernam Cipher
Enter your choice: 4

Enter Plain text to be encrypted: HELLO

Enter the first key: HELP
Enter the second key: RED
Single Encryption: E_HOL_L_
Double Encryption: H___L_EOL
Single Decryption: E_HOL_L_
Double Decryption: HELLO
```

Vernam Cipher

```
1. Substitution
2. ROT 13
3. Transpose
4. Double Transposition
5. Vernam Cipher
Enter your choice: 5

Enter Plain text to be encrypted: HELLO
Enter the key(of same length as the message):TABLE
Encrypted Text: UEKAK
Decrypted Text: HELLO
```

Conclusion:

I learned about different cryptographic algorithms and implemented five of them in my experiment. The following are my observation:

Substitution and ROT13

- I had to make sure that while shifting the values of the characters, the final ordinal value of each character did not exceed the total number of possible characters in the Substitution and ROT13 algorithms. If this was not handled, the software would encounter a runtime error when converting the original string's shifted ordinal values back. I solved this challenge by using the modulo method to encrypt and decrypt a specified string.

Transposition

- The transposition algorithm implemented by me is the columnar one in which the message is written out in rows of fixed length, and then read it out again column by column, and the column are chosen based on the key specified. This method is easy to implement and understand but difficult to be attacked by hackers. The intermediate person cannot break this unless he knows the method. Hence this method is certainly better than substitution and ROT13 but computationally heavy due the creation of matrices and sensitive to large number of characters in the plain text.

Double Transposition

- The user can provide 2 different keys for the transposition in the Double Transposition method, which gives it an extra layer of security. Because the first and second permutation keys can differ in size, I maintained the number of null values(‘_’ in my case) I added in each encryption and after the first decryption removed the number of null values added after the second encryption and removed the null values left after the second decryption.

Vernam Cipher

- The user has complete control over the Vernam Cipher Algorithm and can enter whatever string they choose. This contains strings with spaces in them. These must be dealt with by checking if the XOR value exceeds the permissible values. A drawback of this method is that the key needs to be of the same length of the plain text.

Diffie Hillman

- I observed that through Diffie hillman algorithm the users can communicate with each other through an insecure channel. I had implemented a function to check if number is a primitive root of the given prime number, but it was computationally heavy due to calculation of powers of large values. It makes it possible to silently agree and generate a symmetric key between 2 parties without sharing it over a public channel. It would be nearly impossible to crack the key for this algorithm if the value of the generator is large enough.

Github Links

Repository

https://github.com/kashishvjain/CSS-Lab