Aditya Motwani
2019130040
TE Comps

Experiment 1: Traditional Cryptography Algorithms

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

Github Link:

https://github.com/adityamotwani/CSS-Lab-2019130040/blob/main/exp1.py

Code:

```
import math

def sub(rot):
    print("Enter Plain Text:", end=' ')
    s=input()

    shift=13
    if not rot:
        print("Enter Shift:", end=' ')
        shift=int(input())
    encrypted=""
    for i in s:
        if i!=' ':
            encrypted+=chr(((ord(i)-65+shift)%26)+65)
        else:
            encrypted+=' '
    print("Encrypted String:",encrypted)
    decrypted=""
    for i in encrypted:
        if i!=' ':
            decrypted+=chr(((ord(i)-shift-65)%26)+65)
        else:
            decrypted+=' '
    print("Decrypted String:",decrypted)
    print()
```

```
def vernam():
    print("Enter Plain Text:", end=' ')
    s=input()
    print("Enter Key:", end=' ')
    key=input()
    if len(s)==len(key):
        encrypted=""
        for i in range(len(s)):
            if s[i]!=' ':
                encrypted+=chr((ord(s[i])+ord(key[i])-130)%26 +65)
            else:
                encrypted+=' '
        print("Encrypted String:",encrypted)
        decrypted=""
        for i in range(len(s)):
            if encrypted[i]!=' ':
                decrypted+=chr((ord(encrypted[i])-ord(key[i])-130)%26 +65)
            else:
                decrypted+=' '
        print("Decrypted String:",decrypted)
        print()

    else:
        print("Plain Text Length and Key Length do not match")
        print()




def double_transposition():
    print("Enter Plain Text:", end=' ')
    s=input()
    print("Enter 1st Key:", end=' ')
    key=input()
    print("Enter 2nd Key:", end=' ')
    key2=input()

    if len(key2)>math.ceil(len(s)/len(key)):
        print("Dimensions of the keys and the plain text do not match.
Please try again")
```

```
    print()
    return None

table=[]
for i in range(len(key)):
    temp=list(s[i:len(s):len(key)])
    # print(temp)
    table.append(temp)
if len(table[-1])!=len(key):
    t1=len(table[-1])
    for _ in range(len(key)-t1):
        table[-1].append('x')
order=[]
for i in range(len(key)):
    order.append([key[i],i])
order=sorted(order, key = lambda x: x[0])
for i in range(len(order)):
    order[i].append(i)



# print(table)


order2=[]
for i in range(len(key2)):
    order2.append([key2[i],i])
order2=sorted(order2, key = lambda x: x[0])
for i in range(len(order2)):
    order2[i].append(i)
# print(order2)
#Shuffle Rows
order2=sorted(order2, key = lambda x: x[1])
for i in range(len(key)):
    temp=[]
    for j in range(len(key2)):
        temp.append([ table[i][j],order2[j][2] ])
    temp=sorted(temp, key = lambda x: x[1])
    table[i]=[]
    for j in temp:
```

```
        table[i].append(j[0])
order2=sorted(order2, key = lambda x: x[0])

# print(table)

#Shuffle Columns
final=[]
for i in order:
    final.append(table[i[1]])

# print(order)
# print(final)


encrypted=""
for i in range(len(key2)):
    for j in final:
        encrypted+=j[i]
print("Encrypted String:",encrypted)


decrypt_table=[]
for i in range(len(key)):
    decrypt_table.append([])
    for j in range(i,len(encrypted),len(key)):
        decrypt_table[-1].append(encrypted[j])
# print(decrypt_table)

#Unshuffle column
decrypt_final=[]
order=sorted(order, key = lambda x: x[1])
for i in order:
    decrypt_final.append(decrypt_table[i[2]])

#Unshuffle Rows
for i in range(len(key)):
    temp=[]
    for j in range(len(key2)):
        temp.append([ decrypt_final[i][j],order2[j][1] ])
    temp=sorted(temp, key = lambda x: x[1])
```

```
            decrypt_final[i]=[]
            for j in temp:
                decrypt_final[i].append(j[0])

        # print(decrypt_final)

        decrypted=""
        for i in range(len(key2)):
            for j in decrypt_final:
                decrypted+=j[i]
        print("Decrypted String:",decrypted)
        print()



def transposition():
    print("Enter Plain Text:", end=' ')
    s=input()
    print("Enter Key:", end=' ')
    key=input()

    if len(key)>len(s):
        print("Dimensions of the key and the plain text do not match.
Please try again")
        print()
        return None

    table=[]
    for i in range(len(key)):
        temp=list(s[i:len(s):len(key)])
        # print(temp)
        table.append(temp)
    order=[]
    for i in range(len(key)):
        order.append([key[i],i])
    order=sorted(order, key = lambda x: x[0])
    # print(table)
    encrypted=""
    for i in order:
        for j in range(len(key)):
```

```
            if j<len(table[i[1]]):
                encrypted+=table[i[1]][j]
            else:
                encrypted+='x'
    print("Encrypted String:",encrypted)

    decrypt_table=[]
    k=0
    for i in range(len(key)):
        decrypt_table.append([])
        for j in range(i,len(encrypted),len(key)):
            decrypt_table[-1].append(encrypted[j])
    print(decrypt_table)
    for i in range(len(order)):
        order[i].append(i)
    order=sorted(order, key = lambda x: x[1])
    decrypted=""
    for i in decrypt_table:
        for j in order:
            decrypted+=i[j[2]]
    print("Decrypted String:",decrypted)
    print()


choice=1
while choice!=0:
    print("Please Enter your choice for an Encryption algorithm\n1:
Substitution Cipher\n2: ROT 13\n3: Vernam Cipher\n4: Transposition\n5:
Double Transposition\n0: To Exit")
    choice=int(input())
    if choice==1:
        sub(False)
    if choice==2:
        sub(True)
    if choice==3:
        vernam()
    if choice==4:
        transposition()
    if choice==5:
        double_transposition()
```

Output:

1) Substitution Cipher

```
Please Enter your choice for an Encryption algorithm
1: Substitution Cipher
2: ROT 13
3: Vernam Cipher
4: Transposition
5: Double Transposition
0: To Exit
1
Enter Plain Text: DEEP IS A GOD BOY
Enter Shift: 6
Encrypted String: JKKV OY G MUJ HUE
Decrypted String: DEEP IS A GOD BOY
```

2) ROT 13 Cipher

```
Please Enter your choice for an Encryption algorithm
1: Substitution Cipher
2: ROT 13
3: Vernam Cipher
4: Transposition
5: Double Transposition
0: To Exit
2
Enter Plain Text: DEEP IS A GOD BOY
Encrypted String: QRRC VF N TBQ OBL
Decrypted String: DEEP IS A GOD BOY
```

3) Vernam Cipher

```
Please Enter your choice for an Encryption algorithm
1: Substitution Cipher
2: ROT 13
3: Vernam Cipher
4: Transposition
5: Double Transposition
0: To Exit
3
Enter Plain Text: HACK ON
Enter Key: DAMN IT
Encrypted String: KAOX WG
Decrypted String: HACK ON
```

4) Transposition Cipher

```
Please Enter your choice for an Encryption algorithm
1: Substitution Cipher
2: ROT 13
3: Vernam Cipher
4: Transposition
5: Double Transposition
0: To Exit
4
Enter Plain Text: ATTACK AT BERLIN
Enter Key: ENIGMA
Encrypted String: KExA RA NTTICBxTAL
Decrypted String: ATTACK AT BERLINxx
```

5) Substitution Cipher

```
Please Enter your choice for an Encryption algorithm
1: Substitution Cipher
2: ROT 13
3: Vernam Cipher
4: Transposition
5: Double Transposition
0: To Exit
5
Enter Plain Text: ATTACK AT BERLIN
Enter 1st Key: ENIGMA
Enter 2nd Key: GER
Encrypted String: E  TBAKAATCTxRNIxL
Decrypted String: ATTACK AT BERLINxx
```

Conclusion:

In this experiment we have implemented various cryptography algorithms and learnt their theoretical and implementation details, with the following limitations:

1) The algorithm is only for capital english alphabets
2) The algorithm does not support special characters and number, however it does support spaces
3) Extra spaces have been appended in Transposition and Double Transposition algorithms to satisfy dimensions of the matrix. These spaces are represented by ‘x’

I have further summarized my implementational findings below:

1) Substitution Algorithm:

   This algorithm shifts all alphabets by a certain number, hence to break this algorithm, one has to only try out only 26 values for shift, although shift can take infinite values, but the mod of these values belongs to only one of 26 values. Even with all ASCII characters taken in consideration, yet only a finite number of values need to be tried to crack this algorithm like 128, 256, or 1024. Hence, this algorithm has not been of much use in modern cryptography but it lays down a base ro build more complex cryptography algorithms.

2) ROT-13

   This algorithm always rotates or shifts a character by 13 places, hence this algorithm is more hard-coded version of Substitution method, hence all observations of the latter follow for the former

3) Vernam Algorithm

   This algorithm is polyalphabetic algorithm and encrypts each character with its corresponding character in the key, and hence to break this algorithm one not needs to have the key, or has to try out various combination which would take exponential time complexity and hence would require strong computational power and a lot of time as well. The complexity of this algorithm can be further increased by mapping characters of plain text to non-corresponding characters of the key.

4) Transposition and Double Transposition Algorithms

   In these algorithms the input text is transformed to a matrix and their columns and/or rows are permuted to shuffle the characters. This method is very effective for long strings and a lot of text, but short text can be manually decrypted, since

the characters are only shuffled. An important implementation detail is that len(key2)=ceil(len(input_text)/len(key1)). The complexity of this algorithm can be further increased by using a 3d array instead of a matrix(2d array).