Deep Nayak: 2019130045
Batch - C TE Comps
09/10/2021

# Experiment 1: Traditional Cryptography methods

**Aim:** To understand the concept of cryptography methods and implement the Substitution, ROT13, Transposition, Double Transposition and Vernam Cipher algorithms in python.

## Code:

```
import random, math


def substitution(inputText, shift):
   encrypted = []
   decrypted = []
   for character in inputText:
       if character == ' ':
           encrypted.append('*')
       else:
            encrypted.append(chr(((ord(character) - ord('a') + shift)%26 +
ord('a'))))

   for character in encrypted:
       if character == '*':
           decrypted.append(' ')
       else:
            decrypted.append(chr(((ord(character) - ord('a') - shift)%26 +
ord('a'))))

   return ("".join(encrypted), "".join(decrypted))

def rot13(inputText):
   encrypted = []
   decrypted = []
   for character in inputText:
```

```
        if character == ' ':
            encrypted.append('*')
        else:
                encrypted.append(chr(((ord(character) - ord('a') + 13)%26 +
ord('a'))))

    for character in encrypted:
        if character == '*':
            decrypted.append(' ')
        else:
                decrypted.append(chr(((ord(character) - ord('a') - 13)%26 +
ord('a'))))

    return ("".join(encrypted), "".join(decrypted))

def vernam(inputText, key):
    ansInput = inputText.split(' ')
    superEnc = []
    superDec = []
    for x in ansInput:
        buffer = []
        decrypted = []
        encrypted = []
        for character in x:
            buffer.append(ord(character) - ord('a'))

        for index, character in zip(range(len(key)), key):
            buffer[index] += (ord(character) - ord('a'))
        encrypted = [chr(element%26 + ord('a')) for element in buffer]

        for keychar, character in zip(key, encrypted):
                  decrypted.append(chr((ord(character) - ord(keychar))%26 +
ord('a')))

        superEnc.append("".join(encrypted))
        superDec.append("".join(decrypted))

    return (" ".join(superEnc), " ".join(superDec))

def transpose(inputText, permLength, permOg = []):
```

```
matrix = []
perm = permOg
if len(permOg) == 0:
    perm = [i for i in range(permLength)]
    random.shuffle(perm)
# key1 = " ".join(perm)
print("The key is", perm)
encrypted = []
decrypted = []

buffer = []
cnt = 0
for x in inputText:
    if cnt < permLength:
        buffer.append(x)
    else:
        matrix.append(buffer)
        cnt = 0
        buffer = []
        buffer.append(x)
    cnt += 1
while len(buffer) != permLength:
    buffer.append('X')
matrix.append(buffer)

print(matrix)

for i in perm:
    for j in range(len(matrix)):
        encrypted.append(matrix[j][i])

for i in matrix:
    for j in i:
        if j == 'X':
            break
        decrypted.append(j)

return ("".join(encrypted), "".join(decrypted))
```

```
def doubleTranspose(inputText, permLength1, permLength2, permOg1 = [],
permOg2 = []):
    matrix = []
    perm = permOg1
    if len(permOg1) == 0:
        perm = [i for i in range(permLength1)]
        random.shuffle(perm)
    print("The first key is", perm)
    encrypted1 = []
    encrypted2 = []
    decrypted = []

    buffer = []
    cnt = 0
    for x in inputText:
        if cnt < permLength1:
            buffer.append(x)
        else:
            matrix.append(buffer)
            cnt = 0
            buffer = []
            buffer.append(x)
        cnt += 1
    while len(buffer) != permLength1:
        buffer.append('X')
    matrix.append(buffer)

    print(matrix)

    for i in perm:
        for j in range(len(matrix)):
            encrypted1.append(matrix[j][i])
    matrix1 = matrix
    matrix = []
    buffer = []
    cnt = 0
    perm = permOg2
    if len(permOg2) == 0:
        perm = [i for i in range(permLength2)]
        random.shuffle(perm)
```

```
    # key2 = " ".join(perm)
    print("The second key is", perm)
    for x in "".join(encrypted1):
        if cnt < permLength2:
            buffer.append(x)
        else:
            matrix.append(buffer)
            cnt = 0
            buffer = []
            buffer.append(x)
        cnt += 1
    while len(buffer) != permLength2:
        buffer.append('X')
    matrix.append(buffer)

    print(matrix)

    for i in perm:
        for j in range(len(matrix)):
            encrypted2.append(matrix[j][i])

    for i in matrix1:
        for j in i:
            if j == 'X':
                break
            decrypted.append(j)

    return ("".join(encrypted1), "".join(encrypted2), "".join(decrypted))
```

```
print("~~~~~~~~~~~~~~~~~Experiment 1 - Deep Nayak~~~~~~~~~~~~~~~~~")
print("Choose the cryptography method:")
print("1. Substitution")
print("2. ROT13")
print("3. Transpose")
print("4. Double Transpose")
print("5. Verman Cipher")
```

```
choice = int(input())

if choice == 1:
   print("Enter plain text to be encrypted: ")
   inputText = input()
   print("Enter number of position shift: ")
   shift = int(input())

   (encrypted, decrypted) = substitution(inputText, shift)

   print("The encrypted message is", encrypted)
   print("The decrypted message is", decrypted)

elif choice == 2:
   print("Enter plain text to be encrypted: ")
   inputText = input()

   (encrypted, decrypted) = rot13(inputText)

   print("The encrypted message is", encrypted)
   print("The decrypted message is", decrypted)

elif choice == 3:
   print("Enter plain text to be encrypted: ")
   inputText = input()
   print("Enter the length of the key")
   perm = int(input())

            (encrypted,    decrypted)    =    transpose(inputText=inputText,
permLength=perm)

   print("The encrypted message is", encrypted)
   print("The decrypted message is", decrypted)


elif choice == 4:
   print("Enter plain text to be encrypted: ")
   inputText = input()
   print("Enter the length of the first key")
```

```
    perm1 = int(input())

    print("Enter the length of the second key")
    perm2 = int(input())

                        (encrypted1,       encrypted2,       decrypted)       =
doubleTranspose(inputText=inputText, permLength1=perm1, permLength2=perm2)

    print("The first encrypted message is", encrypted1)
    print("The second encrypted message is", encrypted2)
    print("The decrypted message is", decrypted)

elif choice == 5:
    print("Enter plain text to be encrypted: ")
    inputText = input()
    print("Enter the key of same length as plain text:")
    key = input()
    while len(inputText) != len(key):
        print("Length of key is not correct\nPlease try again:")
        key = input()

    (encrypted, decrypted) = vernam(inputText, key)

    print("The encrypted message is", encrypted)
    print("The decrypted message is", decrypted)
```

# Output:

## Substitution Algorithm

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css$ /bin/python3 /home/deep/spit/css/exp1.py
~~~~~~~~~~~~~~~~Experiment 1 - Deep Nayak~~~~~~~~~~~~~~~~
Choose the cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transpose
5. Verman Cipher
1
Enter plain text to be encrypted:
all that glitters is not gold
Enter number of position shift:
7
The encrypted message is hss*aoha*nspaalyz*pz*uva*nvsk
The decrypted message is all that glitters is not gold
```

## ROT13 Algorithm

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css$ /bin/python3 /home/deep/spit/css/exp1.py
~~~~~~~~~~~~~~~~Experiment 1 - Deep Nayak~~~~~~~~~~~~~~~~
Choose the cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transpose
5. Verman Cipher
2
Enter plain text to be encrypted:
hell is empty and all the devils are here
The encrypted message is uryy*vf*rzcgl*naq*nyy*gur*qrivyf*ner*urer
The decrypted message is hell is empty and all the devils are here
```

## Transposition Algorithm

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css$ /bin/python3 /home/deep/spit/css/exp1.py
~~~~~~~~~~~~~~~~Experiment 1 - Deep Nayak~~~~~~~~~~~~~~~~
Choose the cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transpose
5. Verman Cipher
3
Enter plain text to be encrypted:
uneasy lies the head that wears a crown
Enter the length of the key
5
The key is [1, 2, 3, 0, 4]
The matrix is:
u n e a s
y   l i e
s   t h e
  h e a d
  t h a t
  w e a r
s   a   c
r o w n X
The encrypted message is aihaaa nuys   srn  htw oelteheawseedtrcX
The decrypted message is uneasy lies the head that wears a crown
```

## Double Transposition Algorithm

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css$ /bin/python3 /home/deep/spit/css/exp1.py
~~~~~~~~~~~~~~~~Experiment 1 - Deep Nayak~~~~~~~~~~~~~~~~
Choose the cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transpose
5. Verman Cipher
4
Enter plain text to be encrypted:
these violent delights have violent ends
Enter the length of the first key
5
Enter the length of the second key
7
The first key is [3, 0, 2, 1, 4]
The matrix is:
t h e s e
  v i o l
e n t   d
e l i g h
t s   h a
v e   v i
o l e n t
  e n d s
The second key is [5, 3, 1, 4, 0, 6, 2]
The matrix is:
h v n l s e l
e s o   g h v
n d e i t i
  e n t   e e
t v o   e l d
h a i t s X X
The first encrypted message is hvnlseleso ghvndeiti  ent eetvo eldhaits
The second encrypted message is sgt esnoenoilv edXvsdeval it then thehielX
The decrypted message is these violent delights have violent ends
```

## Vernam Cipher Algorithm

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css$ /bin/python3 /home/deep/spit/css/exp1.py
~~~~~~~~~~~~~~~~Experiment 1 - Deep Nayak~~~~~~~~~~~~~~~~
Choose the cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transpose
5. Verman Cipher
5
Enter plain text to be encrypted:
brevity is the soul of wit
Enter the key of same length as plain text:
asedfdwdfasetpoajsepresase
The encrypted message is bjiynwu ns xat sxmp fj wax
The decrypted message is brevity is the soul of wit
```

## Conclusion:

Through this experiment I learnt about different algorithms used for cryptography and I implemented 5 of them. My findings about the same are as follows:

1. In the Substitution algorithm and ROT13 algorithm, I had to make sure that while shifting the ASCII values of the characters, the final ordinal value of every character does not exceed the total number of available characters. In case this was not handled, the program would run into a runtime error while converting back the shifted ordinal values of the original string. I solved this problem by utilizing the modulo operation and applied it while encrypting as well as decrypting a given string.
2. In the Transposition algorithm, I gave the users two options. Either the user can enter the length of the permutation key that is required while performing columnar transposition, or the user can define the permutation key itself and the program will use that to decode and encode a given string. With this the entire program becomes dynamic and adapts to every different scenario.
3. In the double transposition algorithm, similar to the transposition algorithm, the user can either enter the length of the permutation or enter the exact permutation that needs to be used in both steps of encoding and decoding. Since the size of the first and second permutation keys can be different I first appended all the characters of the string in a 2 dimensional list and in order to make the matrix symmetric, I appended ‘X’ wherever a character was missing.
4. In the Vernam Cipher Algorithm, the user is allowed to enter any string that they can enter. This also includes strings containing spaces. In order to handle this scenario, I performed the vernam cipher algorithm individually on different words present in the string. I had to keep a track of the offset and position of the spaces in the original string so that the same can be mapped in the key provided and the final encrypted string can be generated without any issues.

Github Link: https://github.com/deepnayak/CSS-Lab-Deep-Nayak