

Assignment-1

Keshav Gambhir 2019249

Tanmay Rajore 2019118

Introduction:

In the assignment we have implemented the polyalphabetic substitution cipher on lower english letters. The substitution method used is caesar cipher method for each letter corresponding to each key letter. The code implementation also does a brute force attack on the list of recognizable plain texts to get the symmetric key used for encryption and decryption.

Explanation of polyalphabetic substitution cipher:

Polyalphabetic substitution is similar to monoalphabetic substitution just that the number of monoalphabetic substitution is increased for the same input symbol.

Let

$p = p_0 p_1 \dots p_n$ be the plain text and

$k = k_0 k_1 \dots k_m$ be a key that is used for encryption.

(Typically $m \ll n$)

For doing the polyalphabetic substitution we first make the length of the key equal to the length of the plain text. Then using each pair of alphabet in plain text and key we apply caesar cipher. Mathematically:

$$C = (p_0 + k_0)(p_1 + k_1) \dots (p_m + k_m)(p_{m+1} + k_0) \dots (p_{2m} + k_m) \dots$$

Where $C \rightarrow$ cipher text, $P_i \rightarrow$ alphabet from plain text and $k_i \rightarrow$ alphabet from the key

Here $+$ is modulo 26 addition that is add the number of plain text alphabet and key alphabet and then take mod 26 of the answer.

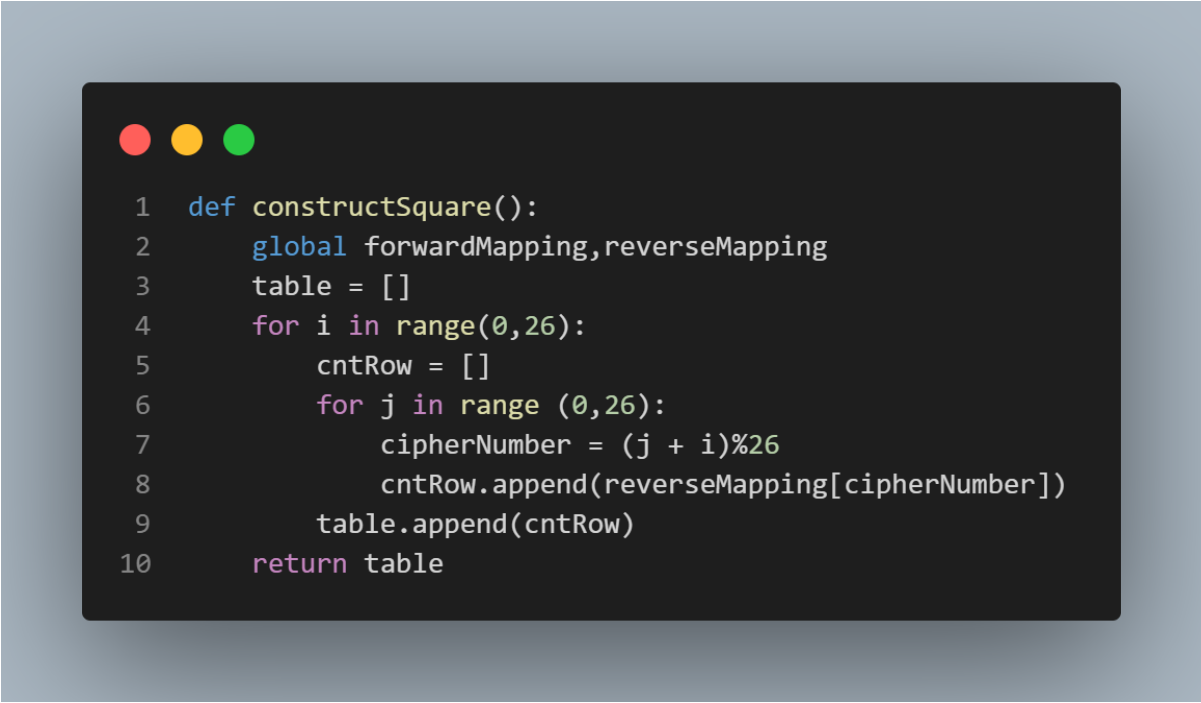
If we consider every possibility that is 26 alphabets for the key and 26 plain text characters we obtain the following table for encryption. This is known as Vigenere Cipher

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Implementation

We have implemented the whole code in 3 modules namely: encrypt.py, decrypt.py and bruteForce.py

To do the encryption and the decryption we have first generated the Vigenere Cipher using the following code:



```
1 def constructSquare():
2     global forwardMapping,reverseMapping
3     table = []
4     for i in range(0,26):
5         cntRow = []
6         for j in range (0,26):
7             cipherNumber = (j + i)%26
8             cntRow.append(reverseMapping[cipherNumber])
9         table.append(cntRow)
10    return table
```

Here forward mapping maps alphabets to their corresponding numbers and the reverse mapping maps numbers to corresponding alphabets. These both are a dictionary. We have done a nested loop from [0,25] and then computed the table containing the mapping for each alphabet corresponding to some other alphabet as shown in the Vigenere Cipher table.

The code for encryption is as follows:

```
1 def constructSameLengthKey(plainText,p_key):
2     q = int(len(plainText)/len(p_key))
3     r = int(len(plainText)%len(p_key))
4     key = p_key*q
5     key += p_key[:r]
6     return key
7
8 def encrypt(plainText,p_key):
9     encryptionKey = constructSameLengthKey(plainText,p_key)
10    table = constructSquare()
11    cipherText = ""
12    for char,keyChar in zip(plainText,encryptionKey):
13        cipherText += table[forwardMapping[keyChar]][forwardMapping[char]]
14    return cipherText
15
```

Here we have first made the key of the same length as that of the plain text. Then using the table generated above we have looked at the row element for key character and column element for plain text character. The corresponding value is the cipher text of the plain text.

The code for decryption is as follows:


```
1 def constructSameLengthKey(cipherText, p_key):
2     q = int(len(cipherText)/len(p_key))
3     r = int(len(cipherText) % len(p_key))
4     key = p_key*q
5     key += p_key[:r]
6     return key
7
8
9 def decrypt(cipherText, p_key):
10    key = constructSameLengthKey(cipherText, p_key)
11    table = constructSquare()
12    plainText = ""
13    for keyChar,char in zip(key,cipherText):
14        plainText += reverseMapping[table[forwardMapping[keyChar]].index(char)]
15    return plainText
```

In this code we have again made the length of key equal to the cipher text. Then using the table we have found the row corresponding to the key character. Then iterated on the row to get the


index of the cipher text. Finally we looked up the index in reverseMapping dictionary and appended that to plain text.

To make the plain text recognizable we have used the SHA256 hashing algorithm from the hashlib library present in the python. This hashed value is appended with the cipher text. The length of SHA256 hashing is equal to 64 characters. So if the key is correct then after decryption of cipher text and hashing it using the hash function we should get the same result as that of what we have appended with the cipher text string.


The brute force attack code is as follows: We have tested for each length that is length of key = 1, 2, 3 and 4



```
1 def bruteForceKeyLength1(cipherTextList):
2     for i in range(0, 26):
3         generatedKey = reverseMapping[i]
4         if (testBruteForcedKey(cipherTextList, generatedKey)):
5             return True, generatedKey
6     print("Key Length not equal to 1")
7     return False, None
```




```
1 def bruteForceKeyLength2(cipherTextList):
2     for i in range(0, 26):
3         for j in range(0, 26):
4             generatedKey = reverseMapping[i] + reverseMapping[j]
5             if (testBruteForcedKey(cipherTextList, generatedKey)):
6                 return True, generatedKey
7     print("Key Length not equal to 2")
8     return False, None
9
```



```

1  def bruteForceKeyLength3(cipherTextList):
2      for i in range(0, 26):
3          for j in range(0, 26):
4              for k in range(0, 26):
5                  generatedKey = reverseMapping[i] + \
6                      reverseMapping[j] + reverseMapping[k]
7                  if (testBruteForcedKey(cipherTextList, generatedKey)):
8                      return True, generatedKey
9      print("Key Length not equal to 3")
10     return False, None
11


```



```

1  def bruteForceKeyLength4(cipherTextList):
2      for i in range(0, 26):
3          for j in range(0, 26):
4              for k in range(0, 26):
5                  for l in range(0, 26):
6                      generatedKey = reverseMapping[i] + reverseMapping[j] + reverseMapping[k] + reverseMapping[l]
7                      if (testBruteForcedKey(cipherTextList, generatedKey)):
8                          return True, generatedKey
9      print("Key Length not equal to 4")
10     return False, None

```



```

1  def testBruteForcedKey(cipherTextList, generatedKey):
2      for cipherTextWithHash in cipherTextList:
3          cipherText, appendedHash = seperateHashAndCipherText(cipherTextWithHash)
4          decryptedPlainText = decrypt(cipherText, generatedKey)
5          if appendedHash != getHashedString(decryptedPlainText):
6              return False
7      return True

```