Assignment1

(0100+0082)mod 3=2

Project 2:

1. Encryption
2. Decryption
3. Brute Force attack

On Transposition Cipher

Aditi Sharma
MT20100

Anjali
MT20082

Meta Data of Assignment

- Language used- Python
- Number of code files -2
  EncryptDecrypt.py
  BruteForceDecrypt.py

- Input data to Encrypt function:
  Key=45132
  Plaintext- Wearediscoveredsaveyourself

- Output of Encrypt function(which is input to Decrypt function):
  encrypted message or ciphertext-
  asrvr8047724eodyefb6a95*rcees1ff7f1aWdvsol7d9ed3eieauf7378d1

- Output of Decrypt function:
  Wearediscoveredsaveyourself

Screenshots

Output of Encrypt and Decrypt

```
The plain text message to be encrypted is:  Wearediscoveredsaveyourself
The hash of the plain text message is:  81f770fbd34f69777ae87f9dd215314a
The message concatenated with hash which is to be encrypted is:   Wearediscoveredsaveyourself81f770fbd34f69777ae87f9dd215314a
The Encrypted Message is:   asrvr8047724eodyefb6a95*rcees1ff7f1aWdvsol7d9ed3eieauf7378d1
The plain text after decryption is:  Wearediscoveredsaveyourself
The hash received in the concatenated string after decryption is:  81f770fbd34f69777ae87f9dd215314a
The hash of the decrypted plain text is:  81f770fbd34f69777ae87f9dd215314a
The hash of plain text after decryption matches to the hash given in the decrypted text. Hence the correct plain text is received
```

Output of BruteForceDecrypt

```
The hash of the decrypted plain text is:  595ffd2af0c75a704835839ea1e7e7a5
The hash of plain text after decryption matches to the hash given in the decrypted text. Hence the correct plain text is received
Iamfinehowareyoucd1f849e2d1b2ed16f3c84427f0ef3da
The plain text after decryption is:  Iamfinehowareyou
The hash received in the concatenated string after decryption is:  cd1f849e2d1b2ed16f3c84427f0ef3da
The hash of the decrypted plain text is:  cd1f849e2d1b2ed16f3c84427f0ef3da
The hash of plain text after decryption matches to the hash given in the decrypted text. Hence the correct plain text is received
Letsgoforawalk0c5ee5bc3d7d9f42034e404ef037f10c
The plain text after decryption is:  Letsgoforawalk
The hash received in the concatenated string after decryption is:  0c5ee5bc3d7d9f42034e404ef037f10c
The hash of the decrypted plain text is:  0c5ee5bc3d7d9f42034e404ef037f10c
The hash of plain text after decryption matches to the hash given in the decrypted text. Hence the correct plain text is received
Whereisthepark6db6756620267a6680274ece0c821223
The plain text after decryption is:  Whereisthepark
The hash received in the concatenated string after decryption is:  6db6756620267a6680274ece0c821223
The hash of the decrypted plain text is:  6db6756620267a6680274ece0c821223
The hash of plain text after decryption matches to the hash given in the decrypted text. Hence the correct plain text is received
Alotofworkispendingc6a07a582f4e9db027d3ae9de6226ab9
The plain text after decryption is:  Alotofworkispending
The hash received in the concatenated string after decryption is:  c6a07a582f4e9db027d3ae9de6226ab9
The hash of the decrypted plain text is:  c6a07a582f4e9db027d3ae9de6226ab9
The hash of plain text after decryption matches to the hash given in the decrypted text. Hence the correct plain text is received


Key Found :  45132
```

We build 3 functionalities:

1. Encryption using *encryptMessage*
2. Decryption using *decryptMessage*
3. Brute Force Attack using *BruteForceDecrypt*

- The user sends a plain text message *T*
- The *key* assumed to be composed of digits with allowable digits given as [1 … Length(*key*)]
- The *key* is known to both functionalities - *encryptMessage* and *decryptMessage*
- Let the *key* be 2134 for illustration purposes.

Encryption using *encryptMessage*

- A plain text **T** is received.
- The MD5 hash, say **H(T)**, of the plain text T is calculated using a pre-built python library function.
- The hash H(T) is concatenated with the plain text T such that the new text ,

  T´ = T || H(T)
- The T´ is then encrypted using transposition cipher method as follows:
  1. A matrix is formed in which the rows and columns are filled corresponding to the order of elements of key(using two nested for loops in the code):

2	1	3	4
C 1	C 2	C 3	C 4
…	…	…	…

1	2	3	4
C 2	C 1	C 3	C 4
…	…	…	…

The **no of rows of matrix** = $\left\lceil \frac{\text{Length}(T')}{\text{Length}(\text{key})} \right\rceil$ **Cols** = Length(*key*)

Here, the … represents more rows as given by the formula
The C 1, C 2 … represents the characters of the plain text

1	2	3	4
C 2	C 1	C 3	C 4
…	…	…	…

The matrix is prefilled by * character , in case the **Length(T´) < Rows * Cols**

- We get the cipher text after the matrix is read row by row, column by column(using two nested for loops in the code) i.e.
  first row first column, first row second column ...
  second row first column, second row second column ...
  and so on
- This encrypted text **E** is sent to *decryptMessage* to perform decryption.

Decryption using *decryptMessage*

- We receive the cipher text **E**
- We make a matrix and fill it with characters from **E** row by row, column by column

1	2	3	4
E 1	E 2	E 3	E 4
…	…	…	…

The **number of rows of matrix** = $\left\lceil \frac{\text{Length(E)}}{\text{Length(key)}} \right\rceil$ **Cols = Length(key)**

Here, since **No of rows = Rows * Cols** , so no need for padding
The matrix also contains the special character ***** we used for padding during encryption

- We read the matrix in the order given by elements of *key* i.e.
  key = **2134** (Assumed initially), then, first the column **2** of matrix is read for each row, then column **1** is read for each row and so on
- We read all characters excluding *****
- The resultant text **T"** is the text **T´ = T || H(T)**. We get the plain text **T** from this using the length of original text **T** sent by the user.

We verify that the plain text **T”** we got matches the plain text **T** sent by the user using method, say **MV,** as follows:

- We find the MD5 hash of **T”** i.e. **H(T”)** and compare it with the **H(T)** we received after the decryption.

If **H(T”) = H(T) ,** decryption yielded the correct original text **T**

Brute Force Attack using *BruteForceDecrypt*

We do the brute force attack as follows:

We take some ***n*** number of <plain text T, cipher text E> pairs

- We generate all possible permutations of *key* with length ***i*** where initially
  ***i* = 2, *key* = 12**

- For each generated key **K,** we decrypt the cipher text.

- If any of the cipher text decryption results in incorrect match with the plain text, as given by the method **MV** we used previously, then the key is discarded and next generated key is checked.

- The above procedure on completion, gives the key that decrypts all cipher texts correctly.