Symmertic Cipher

Team member

- 20171847118 金正旭
- 20171847121 李昊淼
- 20171847127 刘思颖

Lab Environment

- Python 3.7.0
- spyder3

Outline

1. Symmertic Cipher

Definition 1.1:

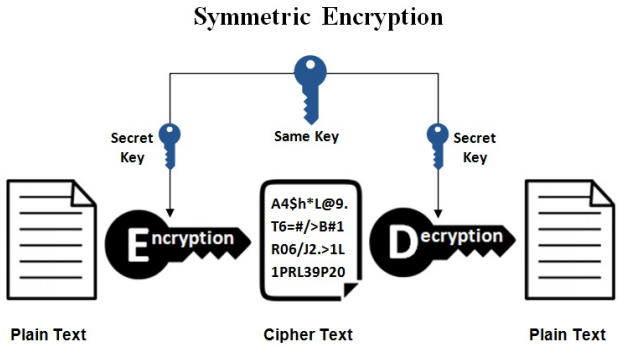
In order to definite a cryptosystem, we assume cryptosystem can be represent as a quintuple (E,D,M,K,C)

1. the M,K,C is finite space

- M is the plaintext space
- K is the ciphertext space
- C is the key space

2. $E={E_k|k \in K}$ is a family of functions $E_k:M \rightarrow C$ that are used for encryption, and $D={D_k|k \in D}$ is a family of functions $D_k:C \rightarrow M$ that are used for decryption.

Symmetric encryption is an encryption methodology that uses a single key to encrypt (encode) and decrypt (decode) data. It is the oldest and most well-known technique for encryption.So we can definite it by quintuple (E,D,M,K,C).



Definition 1.2:

Symmertric encryption is a cipher defined over (K, M, C) , is a pair of "efficient" algs (E,D) where

$$ \begin{align*} & E:K \times M \rightarrow C & D:K \times C \rightarrow M \end{align*} $$

$$s.t.\ \forall m \in M,\forall k \in K,\quad D(k,E(k,m))=m $$

2. Substitution Cipher

In cryptography, a substitution cipher is a method of encrypting by which units of plaintext are replaced with ciphertext, according to a fixed system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing the inverse substitution.

Definition 2.1: Chnage the characters in plaintext to produce the ciphertext.

2.1 Monoalphabetic Cipher

Caesar Cipher

2.2 Polyalphabetic Cipher

Vigenère cipher

3. Transposition Cipher

Definition 3.1: rearranges the characters in the plaintext to form the ciphertext, the letters are not changed.

3.1 Rail-Fence Cipher

In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.

When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.

After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

For instance: Given a string "HELLO WORLD" and set the row equal to 2, we will get a grid like this:

```
H*L*O*W*R*D*
*E*L*_*O*L**
```

So that we can get the cipher text `HLOWRDEL OL`, that encryption process can be implement with python. Our programmer contain two functions are encrpt and decrypt, the encrpt require two inputs: plaintext and key(the number of

row in grid) then output the ciphertext. decrypt function opposite.

```
def encrypt(plain,key):
    rail=[['\n' for i in range(len(plain))] for j in range(key)]
    dir_floor=False
    row,col=0,0
    for i in range(len(plain)):
        if row==0 or row==key-1:
            dir_floor = not dir_floor
        rail[row][col]=plain[i]
        col=col+1
        if dir_floor:
            row =row+1
        else:
            row =row-1

    cip=[]

    for i in range(key):
        for j in range(len(plain)):
            if rail[i][j]!='\n':
                cip.append(rail[i][j])

    print("".join(cip))

def decrypt(cipher,key):
    rail=[['\n' for i in range(len(cipher))]for j in range(key)]
    dir_down=None
    row,col=0,0
    for i in range(len(cipher)):
        if row==0:
            dir_down=True
        elif row==key-1:
            dir_down=False

        rail[row][col]='*'
        col+=1

        if dir_down:
            row+=1
        else:
            row-=1
    index=0
    for i in range(key):
        for j in range(len(cipher)):
            if(rail[i][j]=='*') and (index<len(cipher)):
                rail[i][j]=cipher[index]
                index+=1
    pla=[]
    row,col=0,0
    for i in range(len(cipher)):
        if row==0:
            dir_down=True
        elif row==key-1:
            dir_down=False

        pla.append(rail[row][col])
        col+=1
```

```
        if dir_down:
            row+=1
        else:
            row-=1
    print("".join(pla))

state=input("please chose the model of Rail Fence Cipher\nA. encryption\nB.
decryption\nEnter A or B to chose:")

if state=="A":
    plain=input("\nPlease input the plaintext:")
    key=input("\nPlease input the key:")
    key=int(key)
    encrypt(plain,key)
elif state=="B":
    cipher=input("\nPlease input the ciphertext:")
    key=input("\nPlease input the key:")
    key=int(key)
    decrypt(cipher,key)
else:
    print("You should select a right model")
```

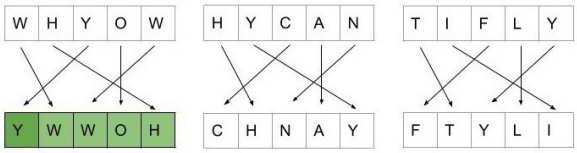
Run our code in bash:

```
$ py3 Rail_fence.py
please chose the model of Rail Fence Cipher
A. encryption
B. decryption
Enter A or B to chose:A

Please input the plaintext:HELLO WORLD

Please input the key:2
HLOWRDEL OL
$ py3 Rail_fence.py
please chose the model of Rail Fence Cipher
A. encryption
B. decryption
Enter A or B to chose:B

Please input the ciphertext:HLOWRDEL OL

Please input the key:2
HELLO WORLD
```

3.2 Transposition Cipher example #1 Permutation transposition

Permute each successive block of n letter in the massage according to position offset n-tuple.

For example: We have the a plaintext dived by 5 letter peer set and the quintuple <+1,+3,-2,0,-2> as key



That is a very simple method, and the represent with python we write is:

```
def encrypt(plain,key):
    grid=[['\n' for i in range(len(key))]for j in range(int(len(plain)/len(key))+1)]
    row,col=0,0
    for i in range(len(plain)):
        if col==len(key):
            row+=1
            col=0
        shift=key[col]
        grid[row][col+shift]=plain[i]
        col+=1
    cip=[]
    for i in range(int(len(plain)/len(key))+1):
        for j in range(len(key)):
            cip.append(grid[i][j])
    print("".join(cip))

def decrypt(cipher,key):
    grid=[['\n' for i in range(len(key))]for j in range(int(len(cipher)/len(key))+1)]
    row,col=0,0
    for i in range(len(key)):
        key[i]=-key[i]
    for i in range(len(cipher)):
        if col==len(key):
            row+=1
            col=0
        shift=key[col]
        grid[row][col-shift]=cipher[i]
        col+=1
    pla=[]
    for i in range(int(len(cipher)/len(key))+1):
        for j in range(len(key)):
            pla.append(grid[i][j])
    print("".join(pla))
state=input("please chose the model of Permutation Cipher\nA. encryption\nB.
decryption\nEnter A or B to chose:")

if state=="A":
    plain=input("\nPlease input the plaintext:")
    key=input("\nPlease input the tuple of key(list format):")
    key=eval(key)
    encrypt(plain,key)
elif state=="B":
    cipher=input("\nPlease input the ciphertext:")
    key=input("\nPlease input the tuple of key(list format):")
    key=eval(key)
    decrypt(cipher,key)
```

```
else:
    print("You should select a right model")
```

Run the code in bash:

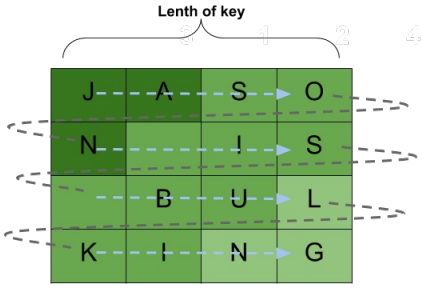
```
$ py3 Permutatuin.py
please chose the model of Rail Fence Cipher
A. encryption
B. decryption
Enter A or B to chose:A

Please input the plaintext:WHYOWHYCANTIFLY

Please input the tuple of key(list format):[1,3,-2,0,-2]
YWWOHCHNAYFTYLI
jason@21:42:$ py3 Permutatuin.py
please chose the model of Rail Fence Cipher
A. encryption
B. decryption
Enter A or B to chose:B

Please input the ciphertext:YWWOHCHNAYFTYLI

Please input the tuple of key(list format):[1,3,-2,0,-2]
WYHOWNCYAHYFILT
```

3.3 Transposition Cipher example #2 Columnar transposition

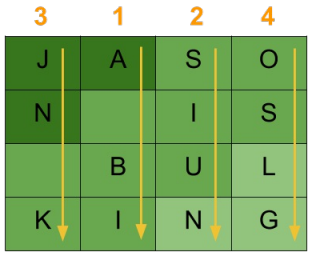
Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher. But Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.

We give a example about the encryption process:

- plaintext: JASON IS BULKING
- key:[3,1,2,4]
- write the plaintext in block like the Graph 3.3.1, and put the key on the top of block. The ciphertext is according to the order of the key to read the column of the block.



Graph 3.3.1



Graph 3.3.2

- ciphertext:A BISIUNIN KOSLG

This algorithm will be easy to implement with python matrix:

```
import math

def encrypt(plain,key):
```

```
    cip="" #establish a empty string to store ciphertext
    col=len(key) #calcuate the number of column in matrix
    right_k=sorted(key) #get the ordered key
    row=int(math.ceil(len(plain)/len(key))) # get the number of row in matrix
    fill_null=int((row*col)-len(plain)) #calcuate how many empty grid we need
    plain=list(plain) #converse plaintext to list
    plain.extend('_'*fill_null) #insert empty character in the tail of the plaintext
list
    matrix = [plain[i:i+col] for i in range(0,len(plain),col)] #fill matrix with
character by row-wise
    # read matrix by column-wise
    for i in range(col):
        curr_idx=key.index(right_k[i])
        cip += ''.join(row[curr_idx] for row in matrix)

    print("".join(cip))

def decrypt(cipher,key):
    pla=""
    index=0
    col=len(key)
    right_k=sorted(key)
    row=int(math.ceil(len(cipher)/col))
    cipher=list(cipher)
    matrix=[]
    for i in range(row):
        matrix += [[None]*col]
    for i in range(col):
        curr_idx=key.index(right_k[i])
        for j in range(row):
            matrix[j][curr_idx]=cipher[index]
            index+=1
    pla = ''.join(sum(matrix, []))

    print("".join(pla))

state=input("please chose the model of Cloumnar Transposition Cipher\nA.
encryption\nB. decryption\nEnter A or B to chose:")

if state=="A":
    plain=input("\nPlease input the plaintext:")
    key=input("\nPlease input the tuple of key(list format):")
    key=eval(key)
    encrypt(plain,key)
elif state=="B":
    cipher=input("\nPlease input the ciphertext:")
    key=input("\nPlease input the tuple of key(list format):")
    key=eval(key)
    decrypt(cipher,key)
else:
    print("You should select a right model")
```

Run the code in bash:

```
$ py3 Columar.py
please chose the model of Rail Fence Cipher
A. encryption
```

```
B. decryption
Enter A or B to chose:A

Please input the plaintext:JASON IS BULKING

Please input the tuple of key(list format):[3,1,2,4]
A BISIUNJN KOSLG
$ py3 Columar.py
please chose the model of Cloumnar Transposition Cipher
A. encryption
B. decryption
Enter A or B to chose:B

Please input the ciphertext:A BISIUNJN KOSLG

Please input the tuple of key(list format):[3,1,2,4]
JASON IS BULKING
```

4. One Time Pad Cipher

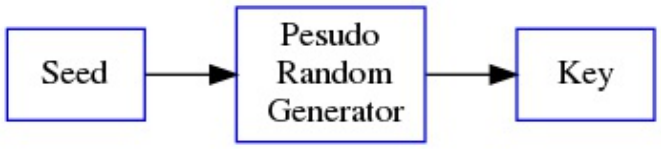
In order to increase the security of cipher, we should improve the randomness of key. So we got the OTP technique(also called Vernam-cipher or the perfect cipher), OTP is an encryption technique that cannot be cracked, but requires the use of a one-time pre-shared key the same size as, or longer than, the message being sent.

Some features about the OTP:

- The key is at least as long as the message or data that must be encrypted.
- The key is truly random (not generated by a simple computer function or such)
- Key and plaintext are calculated modulo 10 (digits), modulo 26 (letters) or modulo 2 (binary)
- Each key is used only once, and both sender and receiver must destroy their key after use.
- There should only be two copies of the key: one for the sender and one for the receiver (some exceptions exist for multiple receivers)

5. Stream Cipher

In order to make the OTP technique more practical, we can use the stream cipher. This method replaced random key by pseudorandom key. The pseudrandom key is produce by PRG(Pseudo Random Generator).



It also can be represent as those formula:

$$ \begin{align*} & C=E(K,M)=M \oplus G(k) & M=D(K,M)=C \oplus G(k) \end{align*} $$

5.1 RC4

RC4 is belong to the set of RC algorithm, this set is about symmertic-key encryption algorithm invented by Ron Rivest. Here is explain about those algorithms:

- RC1 was never published.
- RC2 was a 64-bit block cipher developed in 1987.
- RC3 was broken before ever being used.

- RC4 is the world's most widely used stream cipher.
- RC5 is a 32/64/128-bit block cipher developed in 1994.
- RC6, a 128-bit block cipher based heavily on RC5, was an AES finalist developed in 1997.

A RC4 key input is a pesudorandom bit generator that produces a stream 8-bits number that is unpredictable without the knowledge about the key input. The key stream is output of the bit generator, and it will combine plaintext and key stream one byte a time by XOR option.

So, we can discover that XOR option is easily to code and the main of algorithm will consist by two parts: KSA(Key-scheduling algorithm) and PRGA(Pseudo-random generation algorithm). To generator the key-stream, we need two things:

- A permutation of all 256 possible bytes (denoted "S" below).
- Two index-pointers (denoted "i" and "j").

KSA is used to initialize the permutation in the array "S".

6. Block Cipher

4. Supplement

- Transposition Cipher Wikipedia(https://en.wikipedia.org/wiki/Transposition_cipher)
- GeeksforGeeks(https://www.geeksforgeeks.org/columnar-transposition-cipher/)
- RC4 Wikipedia(https://en.wikipedia.org/wiki/RC4)
- OTP Wiki(https://en.wikipedia.org/wiki/One-time_pad)
- Johns Hopking Univercity(http://www.cs.jhu.edu/~scheideler/courses/600.471_S05/lecture_6.pdf)
- Boise State Univerctiy with Department of Mathematics(https://math.boisestate.edu/~liljanab/ISAS/course_materials/SymmetricKeyCryptography.PDF)
- .dot file for Graphviz

```
digraph g{
        node [
            shape = "record"
            color = "blue"
        ]
        rankdir="LR"
              Seed -> "Pesudo\n Random\n Generator";
        "Pesudo\n Random\n Generator"->Key;
}
```