

Table of Contents

1.Introduction 3

2.Answer A..... 3

 Task 1 3

 Task 2 3

 Task 3 4

 Output 1 4

3. Answer B 5

 Output 2 6

4. Answer C 7

 Task 1 7

 Task 2 8

 Task 3 8

 Output 3 9

5. Answer D 10

 Output 4 10

6. References 11

7. Appendix 12

1.Introduction

The purpose of this lab report is to answer the Lab Assignment 2 questions given by Dr Hossein Anisi on MOODLE [1]. Python 3.5.1 is used as the programming language.

2.Answer A

As a solution to question A, using two multiplicative generators, combined linear congruential generator will be created [2].

Task 1

```
#####COMBINED LINEAR CONGRUENTIAL GENERATOR #####
import random
y_1 = (random.randint(1, m_1 - 1))
y_2 = (random.randint(1, m_1 - 1))
m_1 = 2147483563
m_2 = 2147483399
a_1 = 40014
a_2 = 20692
one_time_private_keys=[]
```

In task 1, firstly, random module is imported. For creating “seeds”, the computer generated initial values (y₁ & y₂) are used. The other values (m₁, m₂, a₁, a₂) are predefined by Lab Assignment 2 [1]. Finally, an array (one_time_private_keys) is created for holding private keys.

Task 2

```
for i in range (0,2):
    y_1 = a_1 * y_1 % m_1
    y_2 = a_2 * y_2 % m_2
    decision = (y_1-y_2)%(m_1-1)
    if decision > 0:
        one_time_private_keys += [int((decision / m_1) * 500)]
    elif decision < 0:
        one_time_private_keys += [int((decision / m_1 + 1) * 500)]
    else:
        one_time_private_keys += [int(((m_1 - 1)/m_1) * 500)]
```

In task 2, a loop is created, (in this case it is between 0-2, because only two key is needed)

$$X_{1,j+1}=(a_1 * X_1) \text{ Mod } m_1$$

Using the formula above, a number between 0 and 1 is produced. Then, this number is multiplied by 500 to be converted to a number between 0-500. Thanks to the for loop, this process is repeated twice. The results are assigned into the array (one_time_private_keys).

Task 3

```
print("COMBINED LINEAR CONGRUENTIAL GENERATOR")
for i in one_time_private_keys:
    print("User %s Private Key= " %(chr(65+one_time_private_keys.index(i))) ,i)
```

In this step, with a for loop, the private keys of the users are printed on the screen.

Output 1

As can be seen from the picture, Figure 1 , the program can produce private key for User A and User B between 0-500. In addition, Program produces new keys for each new connection.



```
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 339
User B Private Key= 185
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 466
User B Private Key= 48
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 490
User B Private Key= 189
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 234
User B Private Key= 445
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 79
User B Private Key= 328
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 458
User B Private Key= 305
>>> |
```

Ln: 39 Col: 4

Figure 1 output of Combined Linear Congruential Generator

3. Answer B

As a solution to question B, using Diffie-Hellman algorithm and private keys from Answer A, will be created the secret key for User A and User B [1].

```
##### DIFFIE-HELLMAN ALGORITHM #####
x,y=[],[]
q,alpha=353,3
for i in one_time_private_keys:
    x+= [i % q]
    y+= [(alpha ** i) % q]

secretkey=(y[0] ** x[1]) % q

print(" Xa= %s\n Xb= %s\n Ya= %s\n Yb= %s\n" %(x[0],x[1],y[0],y[1]))
print("Diffie-Hellman algorithm Secret Key = %s" %(secretkey))
```

In this step, two arrays (x and y) are used for holding calculation results. Alpha and q values (alpha=353 & q=3) are predefined by Lab Assignment 2 [1].

The private keys produced in the previous step were between 0-500. In the for loop, firstly, with Mod operation,

$$i \% \text{mod } q$$

the private keys are set to the desired value range (0-353). In this way, another condition [3] is provided ($X_A < q$ and $X_B < q$). After that, Y_A and Y_B values are calculated according to these formulas [3]:

$$Y_A = \alpha^{X_A} \text{ mod } q$$

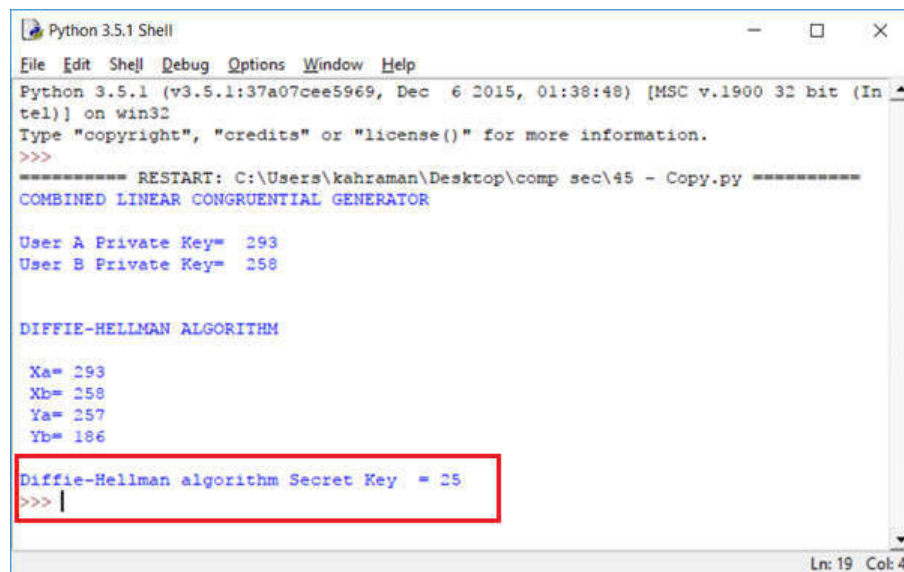
$$Y_B = \alpha^{X_B} \text{ mod } q$$

After exiting the for loop, the secret key is calculated according to the formula below and printed on the screen.

$$\text{Key} = (Y_A)^{X_B} \text{ mod } q \text{ or } \text{Key} = (Y_B)^{X_A} \text{ mod } q$$

Output 2

As can be seen from the picture, Figure 2, the program produces Diffie-Hellman algorithm secret key



```
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 293
User B Private Key= 258

DIFFIE-HELLMAN ALGORITHM

Xa= 293
Xb= 258
Ya= 257
Yb= 186

Diffie-Hellman algorithm Secret Key = 25
>>> |
```

Figure 2 output of Diffie-Hellman algorithm

4. Answer C

As a solution to question C, using RC4 algorithm[4] and the secret key from Answer B, will be encrypted the last 128-bit part of text from book.txt file[1].

Task 1

```
bookfile = open("book.txt")
book=bookfile.read()
bookfile.close()
block_128_bits=[]

for i in range(0,len(book),16):
    block_128_bits += [book[ i : i + 16 ]]

book=block_128_bits[len(block_128_bits)-1]

keystream, s, t = [],[],[]
key =str(secretkey)
```

In task 1, Firstly, book.txt file is opened and assigned all text in a variable(book). After that, in the for loop, the text is spitted into 128bits part (16 letters), and hold in an array. After exiting the for loop, the last chunk of text is assigned into a variable (book). This variable will be encrypted.

Finally, the necessary arrays are created. (key, keystream for Stream Generation, s & t for Initialization and Initial Permutation)

Task 2

```
# Initialization
for i in range (0 , 256 ) :
    s+=i
    t+= [ key[i % len(key ) ] ]

# Initial Permutation of S
j = 0
for i in range (0,256 ) :
    j = (j + s[i] + ord(t[i] ) ) % 256
    s[i] ,s[j] = s[j] ,s[i]

# Stream Generation
j = 0
for i in range (1,len(book ) + 1 ) :
    j = (j + s[i] ) % 256
    s[i] , s[j] = s[j] , s[i]
    counter = (s[i] + s[j] ) % 256
    keystream+=s[counter] ]
```

In task 2, three basic operations are performed: Initialization, Initial Permutation of S and Stream Generation.

In the Initialization step, the numbers between 0-256 are assigned to the s variable and , the key variable is being extended to make the same length with the text.

In the Initial Permutation of S step, the value of the variable s is changed, this operation is repeated 256 times.

In the Stream Generation step, random numbers are generated as many as the length of the text and stored in the keystream array.

Task 3

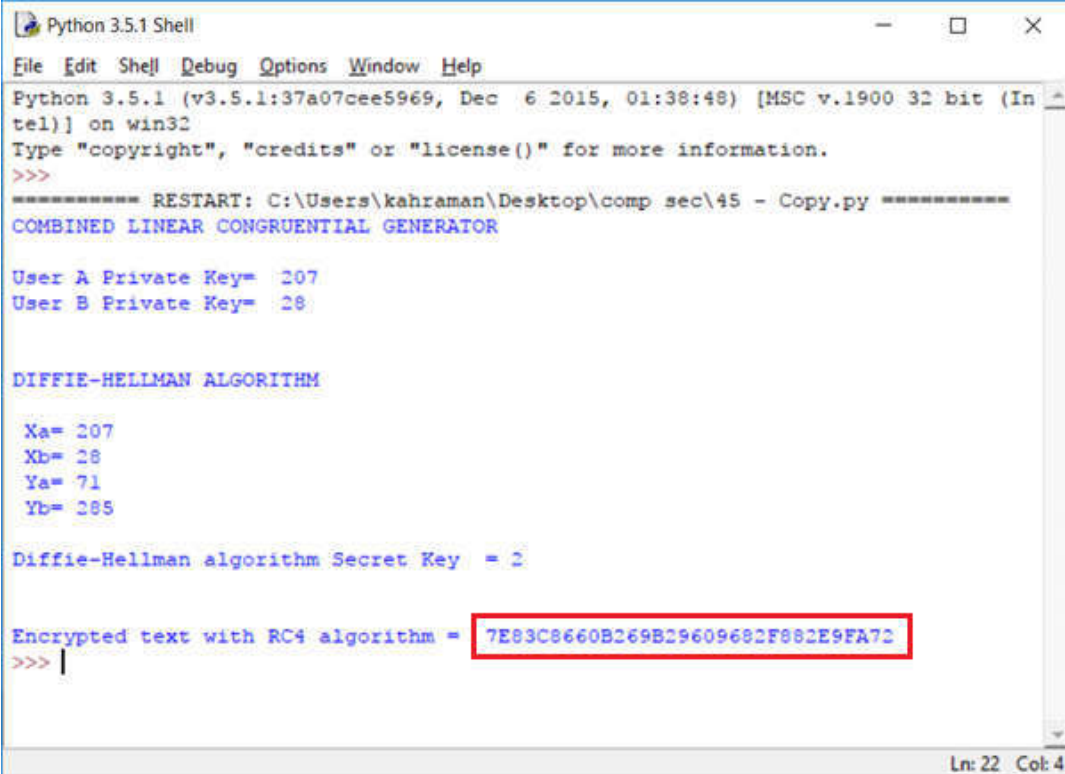
```
encrypted_msg = ""
counter = 0

for i in book:
    temporary = ("%02X" % (ord( i ) ^ (keystream[counter] ) ) )
    encrypted_msg = encrypted_msg + str(temporary)
    counter += 1
print("Encrypted text with RC4 algorithm = %s" %(encrypted_msg))
```

In task 3, a for loop is created in text and in this loop, the XOR operation is performed with the keystream for each letter in the text. The result of this operation is stored in a string variable (encrypted_msg). After exiting the for loop, encrypted text is printed on the screen.

Output 3

As can be seen from the picture, Figure 3, the program produces an encrypted text made by RC4 algorithm.



```
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 207
User B Private Key= 28

DIFFIE-HELLMAN ALGORITHM

Xa= 207
Xb= 28
Ya= 71
Yb= 285

Diffie-Hellman algorithm Secret Key = 2

Encrypted text with RC4 algorithm = 7E83C8660B269B29609682F882E9FA72
>>> |
```

Figure 3 output of RC4 Algorithm

5. Answer D

As a solution to question D, using RC4 algorithm [4] the encrypted text taken from answer C will be decrypted.

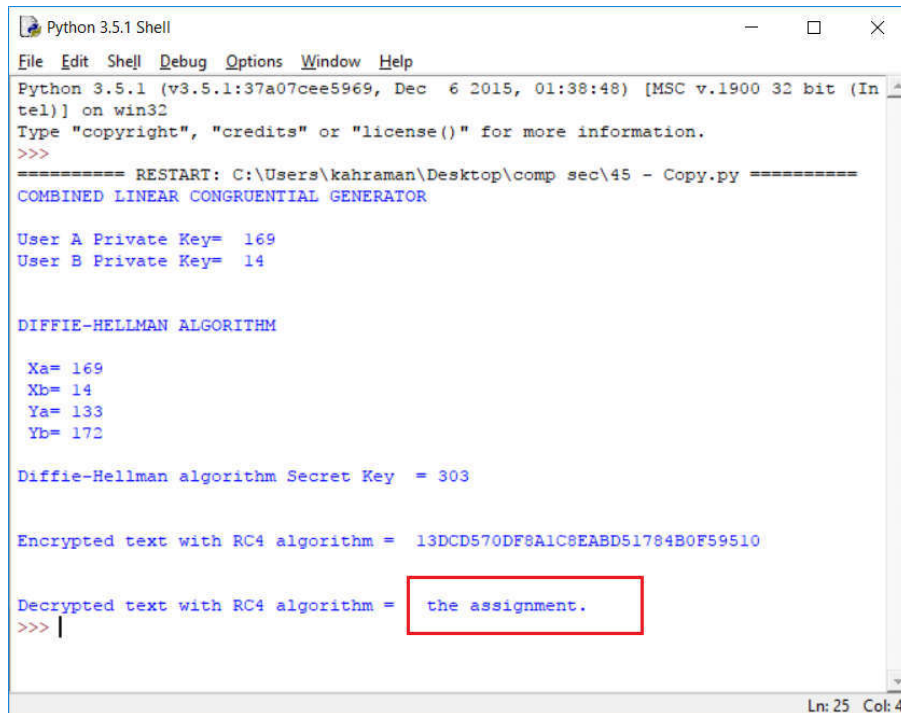
```
##### RC4 ALGORITHM DECRYPTION #####
counter = 0
decrypted_msg=""
for i in range(0,len(encrypted_msg)-1,2):
    temporary = (((int(encrypted_msg[i:i+2],16)) ^ (keystream[counter] ) ) )
    decrypted_msg = decrypted_msg + chr(temporary)
    counter += 1
print("Decrypted text with RC4 algorithm = %s" %(decrypted_msg) )
```

In task 3, a for loop is created in the encrypted text. The encrypted text will be divided into 2 letter groups and the operation will be done with these groups. Because two hexadecimal characters hold an ascii character so “2” is selected as the for-loop step.

In this loop, the XOR operation is performed with the keystream for each two letter in the encrypted text. The result of this operation is stored in a string variable (decrypted_msg). After exiting the for loop, decrypted text is printed on the screen.

Output 4

As can be seen from the picture, Figure 4, the program produces a decrypted text made by RC4 algorithm.



```
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\kahraman\Desktop\comp sec\45 - Copy.py =====
COMBINED LINEAR CONGRUENTIAL GENERATOR

User A Private Key= 169
User B Private Key= 14

DIFFIE-HELLMAN ALGORITHM

Xa= 169
Xb= 14
Ya= 133
Yb= 172

Diffie-Hellman algorithm Secret Key = 303

Encrypted text with RC4 algorithm = 13DCD570DF8A1C8EABD51784B0F59510

Decrypted text with RC4 algorithm = the assignment.
>>> |
```

Figure 4 output of RC4 Algorithm