Report on

CE708-7-AU– Computer Security

Lab Assignment 2

Student Name : Atiwat Onsuwan (ao18409)

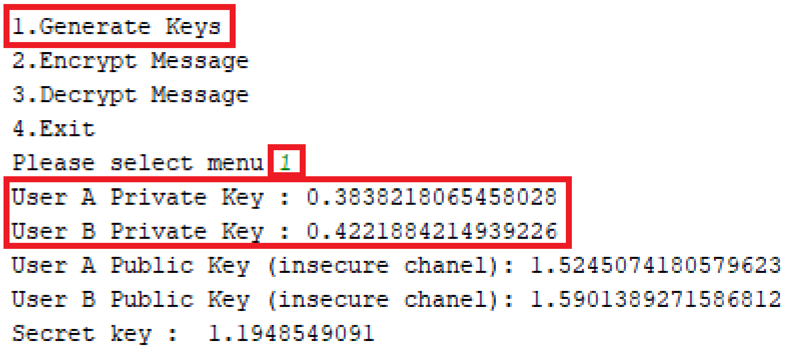
Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection. The two users, each is going to generates a one-time private key and calculates a public key. These public values, together with global public values for q and α, are stored in some central directory. Write a program in python and address the following requirements.

a. [3%] Generate two random numbers as the one-time private keys for users A and B using combined linear congruential generator considering m1 = 2,147,483,642, a1 = 450, m2 = 2,147,483,423, a2 = 234. For Y0,1, generate a random number between [1, 2,147,483,641] and for Y0,2, generate a random number between [1, 2,147,483,422]. Use modulo operation to generate the random numbers in the range of 0 and 500. (For a new connection, new private keys should be generated)

Code

```
import random

# this funtion is used to read files
def read_text(file_path):
    with open(file_path, "r") as file:
        return file.read()


private_keys = []

# this function is used to generate private keys and calculate public keys to get
secret key
def generate_keys():
    if len(private_keys) == 2:
        del private_keys[:]
    y1 = random.randint(1, 2147483423) # random y01
    y2 = random.randint(1, 2147483422) # random y01
    m1 = 2147483642 #given m1
    m2 = 2147483423 #given m2
    a1 = 450 #given a1
    a2 = 234 #given a2
    alpha = 3 #given alpha
    prime = 353 #given prime
    for i in range(2):
        # Evaluate 2 LCGs
        y1 = (y1 * a1) % m1
        y2 = (y2 * a2) % m2
        # CLCG equation
        x = (y1 - y2) % m1
        # calculate private key using clcgs formula and cope range not to higer than
500 by modding 500
        if x > 0:
            private_keys.append((x / m1) % 500)
        elif x < 0:
            private_keys.append((x / (m1 + 1)) % 500)
        elif x == 0:
            private_keys.append(((m1 - 1) / m1) % 500)
```

Output



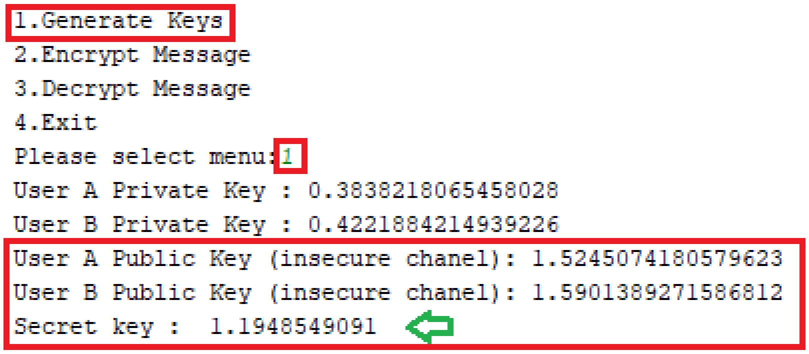
```
1.Generate Keys
2.Encrypt Message
3.Decrypt Message
4.Exit
Please select menu 1
User A Private Key : 0.3838218065458028
User B Private Key : 0.4221884214939226
User A Public Key (insecure chanel): 1.5245074180579623
User B Public Key (insecure chanel): 1.5901389271586812
Secret key :  1.1948549091
```

b. [3%] Using the Diffie-Hellman algorithm and the private keys generated in (a), generate the secret key for users A and B. consider the prime number q = 353 and a primitive root of 353, in this case is α= 3.

Code

```
insec_secretA = (alpha ** private_keys[0]) % prime  # calculate insecure channel of user A
insec_secretB = (alpha ** private_keys[1]) % prime  # calculate insecure channel of user B
# calculate shared key for user A based on insecure channel of user B and private key of user A
a_secret = ((insec_secretB ** private_keys[0]) % prime)
# calculate shared key for user B based on insecure channel of user A and private key of user B
b_secret = ((insec_secretA ** private_keys[1]) % prime)
print("User A Private Key :", private_keys[0])
print("User B Private Key :", private_keys[1])
print("User A Public Key (insecure chanel):", insec_secretA)
print("User B Public Key (insecure chanel):", insec_secretB)
print("Secret key : ", (round(a_secret, 10) + round(b_secret, 10)) / 2)
```

Output



```
1.Generate Keys
2.Encrypt Message
3.Decrypt Message
4.Exit
Please select menu:1
User A Private Key : 0.3838218065458028
User B Private Key : 0.4221884214939226
User A Public Key (insecure chanel): 1.5245074180579623
User B Public Key (insecure chanel): 1.5901389271586812
Secret key :  1.1948549091
```

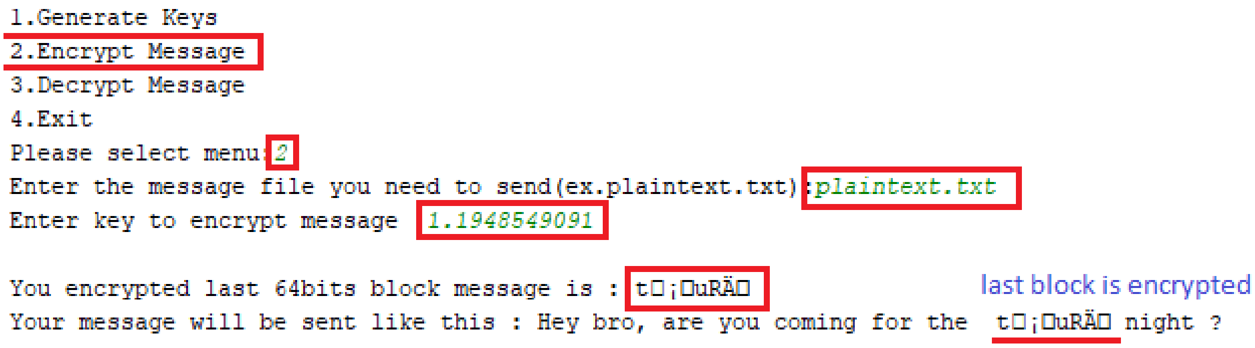
c. [3%] Suppose that user A wishes to send a text file to user B. Break the plaintext into 64 bits blocks and encrypt the last 64 bits block of the plaintext based on RC4 algorithm using the secret key generated in (b) (read an existing text file using 'open' function)

Code

```
# this function is to encrypt and decrypt message
def crypt(key, message):
    S = list(range(256)) # create list of 256 indexes
    j = 0

    for i in list(range(256)): # loop to swap between key and store posible ascii code
        j = (j + S[i] + ord(key[i % len(key)])) % 256
        S[i], S[j] = S[j], S[i]

    j, y = 0, 0
    return_output = []

    for char in message: # loop again to swap each message index and key table above
        j = (j + 1) % 256
        y = (y + S[j]) % 256
        S[j] = S[y]
        S[y] = S[j]
        return_output.append(chr(ord(char) ^ S[(S[j] + S[y]) % 256]))

    return "".join(return_output)

# this function is to match the function read text and crypt to encrypt the message
(calculate last 64 bits block)
def send():
    plain = read_text(input(
        "Enter the message file you need to send(ex.plaintext.txt):"))  # read the
text file and store it in to 'plain'
    plain = list(plain)  # turn every letter into array
    message = list(plain)
    len_message = len(plain)  # count the length of the array plain
    cal_bytes = len_message  # get the last byte of last block
    cal_last_block = (cal_bytes % 8)  # get the first byte of last block
    message = len_message - cal_last_block - 8
    message_last = len_message - cal_last_block
    list_plaintext = plain[
                     (len_message - cal_last_block) - 8:len_message - cal_last_block]
# store every bytes of last block
    list_message = plain[0:message]
    arr_mes_last = plain[len_message - cal_last_block:len_message]

    plain = ""  # 'plain' is = to new 'plain' which is string
    message = ""
    message_last = ""
    for i in list_plaintext:  # loop and add every byte into string 'plain'
        plain += i

    for i in list_message:
        message += i

    for i in arr_mes_last:
        message_last += i

    key = input("Enter key to encrypt message :")  # now get the secret key of user A
```

```
and store it in 'key'
    encrypted = crypt(key,
                      plain)  # call 'crypt' function along with the 'key' and last
64bits block of the read plaintext from text file
    print("\nYou encrypted last 64bits block message is :", encrypted)
    print("Your message will be sent like this :", message, encrypted, message_last)
```

Output



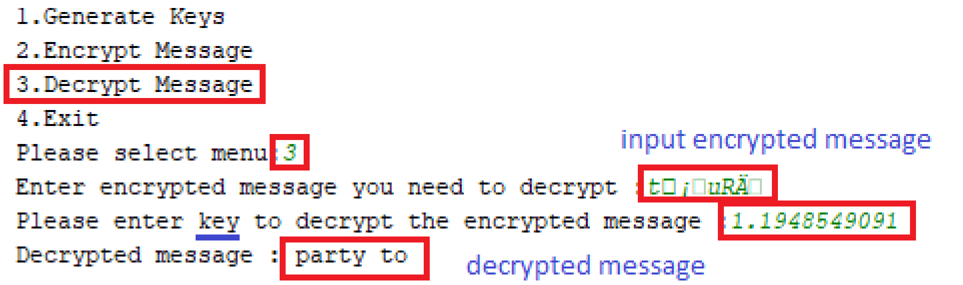
```
1.Generate Keys
2.Encrypt Message
3.Decrypt Message
4.Exit
Please select menu:2
Enter the message file you need to send(ex.plaintext.txt):plaintext.txt
Enter key to encrypt message  1.1948549091

You encrypted last 64bits block message is : t□¡□uRÄ□
Your message will be sent like this : Hey bro, are you coming for the  t□¡□uRÄ□ night ?
```

d. [1%] Decrypt the encrypted message generated in (c) for user B using the secret key.

Code

```
# this function is to encrypt and decrypt message
def crypt(key, message):
  S = list(range(256)) # create list of 256 indexes
  j = 0

  for i in list(range(256)): # loop to swap between key and store posible ascii code
    j = (j + S[i] + ord(key[i % len(key)])) % 256
    S[i], S[j] = S[j], S[i]

  j, y = 0, 0
  return_output = []

  for char in message: # loop again to swap each message index and key table above
    j = (j + 1) % 256
    y = (y + S[j]) % 256
    S[j] = S[y]
    S[y] = S[j]
    return_output.append(chr(ord(char) ^ S[(S[j] + S[y]) % 256]))

  return "".join(return_output)
```

Output



```
1.Generate Keys
2.Encrypt Message
3.Decrypt Message
4.Exit
Please select menu:3
Enter encrypted message you need to decrypt :t□¡□uRÄ□
Please enter key to decrypt the encrypted message :1.1948549091
Decrypted message : party to
```

input encrypted message

decrypted message