CENG418 HW1

Furkan Şahin

250201042

The processes is going on in this order :

1 - Get a valid P (prime) value from the user.

2 - Find a valid G (generator) value automatically.

3 - Get the private keys from users (a and b)

4 - Calculate the shared key according to the given private keys

5 - Get a IV (vector) value from user

6.1 - Get the message input from User 1

6.2 - Encrypt the message that User 1 send and print as encrypted

6.3 - Decrypt the message that User 2 delivers and print as decrypted

6.4 - Get the clear message from decrypted message and print

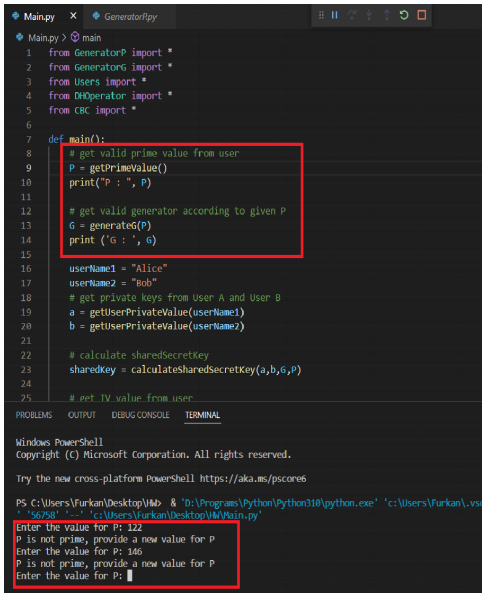
7.1 - Get the message input from User 2

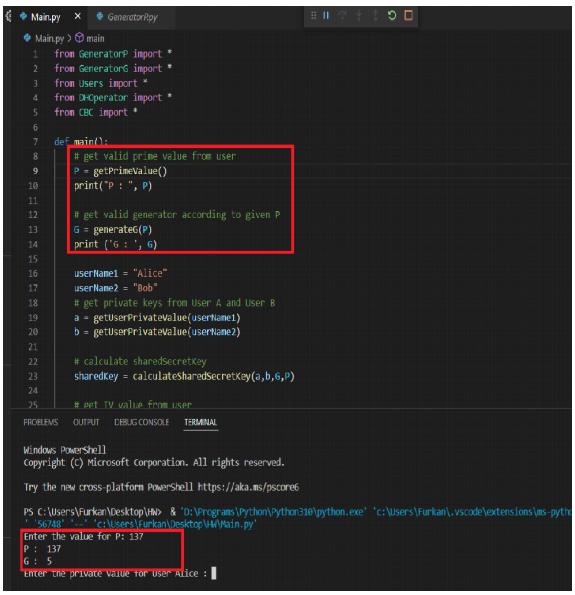
7.2 - Encrypt the message that User 2 send and print as encrypted

7.3 - Decrypt the message that User 1 delivers and print as decrypted

7.4 - Get the clear message from decrypted message and print

For the 1. part : Until the user writes a valid Prime number, get the input.





While we looking at the details of getPrimeValue function :

```python
# the function gets the prime value from user until it is valid.
def getPrimeValue():
    # A prime number P is taken randomly
    P = int(input("Enter the value for P: "))
    K = P-1
    # start test for gotten P
    result = primalityTestIteration(K,P)

    # If test failed, get new value until it is valid
    if result == False:
        print('P is not prime, provide a new value for P')
        return getPrimeValue()
    else:
        return P
```

Until user text a valid Prime value, get the value from user and do primalityTestIteration

```python
# the function provides certain control to be sure gotten P is really valid for selected random a value
def primalityTestIteration(K, P):
    for i in range(0,10):
        # Get random value from a
        a = random.randint(0,P-1)
        # do test for gotten P with random selected a
        primeControl = primalityTest(K,P,a)
        if primeControl != "PRIME":
            return False

    return True
```

for 10 iteration, each round random "a" value is selected and do primalityTest. If one of round doesn't get "PRIME" result, it means value is not prime.

```python
# the function calculates the result of primary test
def primalityTest(K, P, a):
    # calculates the result of primary test
    result_of_calculation_control = calculateValueIsPrime(K,P, a)
    # If the gotten result is one of "PRIME" | "COMPOSITE" return result,
    # else do the test with K/2
    while result_of_calculation_control == "CONTINUE" :
        if(K>2) :
            K = int(ceil(K/2))
            result_of_calculation_control = calculateValueIsPrime(K,P,a)
        else:
            result_of_calculation_control = "FAIL"

    return result_of_calculation_control
```

In the primalityTest, first I invoke calculateValueIsPrime function that controls calculation is PRIME, COMPOSITE or none of them. If none of them, until failing or getting COMPOSITE or PRIME result, do the same process

```python
# the function calculates a^K mod P is prime or not and returns the calculation string result according to the situation
def calculateValueIsPrime(K, P, a):
    # Calculate a^K mod P
    calculation = pow(a, K) % P
    if calculation == P-1:
        return "PRIME"
    elif calculation != 1 and (calculation != -1 and K< (P-1)) :
        return "COMPOSITE"
    elif calculation == -1 or (calculation == 1 and K%2 == 1):
        return "PRIME"
    else:
        return "CONTINUE"
```

In the calculateValueIsPrime function, I do calcuation for a^K mod P and according to the conditions I return a string result.

For the 2. part :

While we looking at the details of generateG function :

First get the prime factors of P-1, then invoke generatorTest function to be sure that there is no result as 1. Until it finds a valid G, it repeats.

```python
def generateG(P):
    # A generator number G is taken randomly
    G = random.randint(2,P-1)

    # find the prime factors to ensure that G garanties the generation.
    primeFactors = findPrimeFactors(P-1)
    generatorResult = generatorTest(P,G, primeFactors)

    # until find a valid G, do the process
    while generatorResult == False:
        G = random.randint(2,P-1)
        generatorResult = generatorTest(P,G, primeFactors)

    return G
```

In the generateG function : until I find a valid G, I get a random number and test it to be sure it can be a generator.

```python
def repeatedSquareMethod(P,G,expo):
    moduloResult = 1
    for i in range (0,expo):
        moduloResult = (G * moduloResult) % P
    return moduloResult

# do the generator test in the Example 2 of homework.pdf
def generatorTest(P, G, primeFactors):
    for primeFactor in primeFactors:
        expo = int((P-1)/primeFactor)
        calculation = repeatedSquareMethod(P,G, expo)
        if calculation == 1:
            return False


    return True
```

In the controlling of tested generator number, I do the process in the right picture. According to the prime factors, I calculate all the results and control that there exists any result == 1 situation or not.

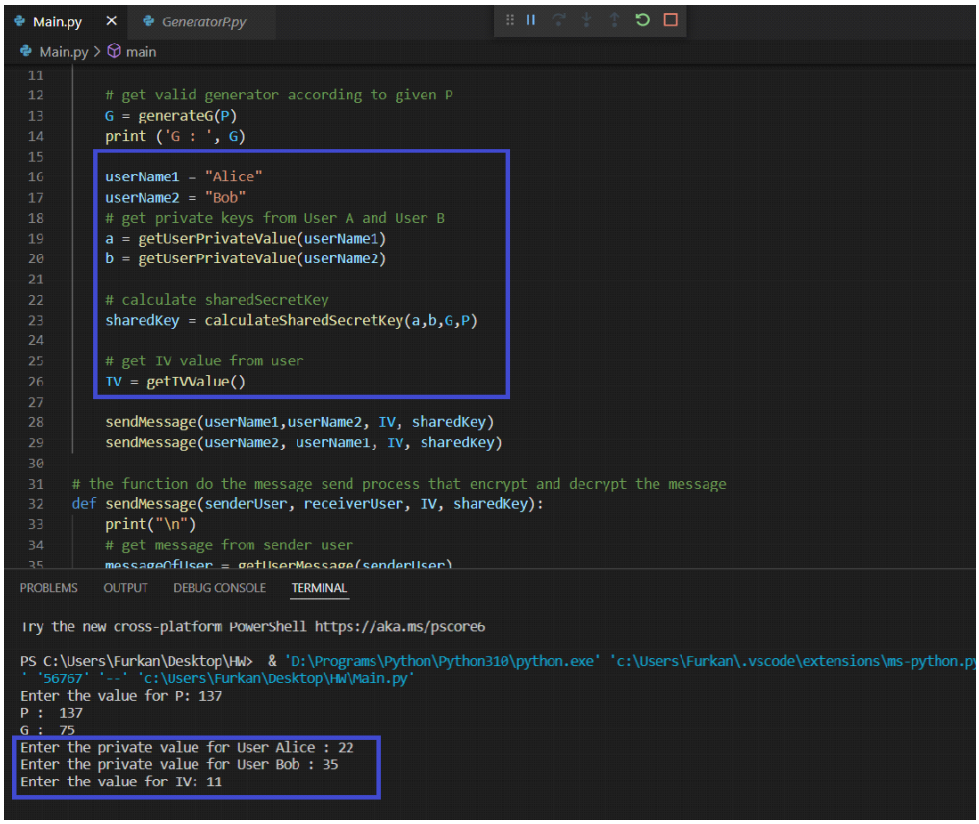
Example 2:

- Let p=11, prime factors of (p-1)=10 are 2 and 5.

Testing 2 whether it is a generator:	Testing 3 whether it is a generator:
$2^{(11-1)/2} \pmod{11} = 10$ $2^{(11-1)/5} \pmod{11} = 4$	$3^{(11-1)/2} \pmod{11} = 1$ $3^{(11-1)/5} \pmod{11} = 9$
Neither result is 1, so 2 is a generator.	One result is 1, so 3 is NOT a generator.

I have used the repeatedSquareMethod for the calculation because it causes Overflow 34 error when using pow(G,expo) % P, because G^expo is too big to fit in an integer value.

For the 3. part : I get the private values from User1(Alice) and User2(Bob)



For the 4. part :

After that I calculate the shared secret key and get the Vector value from the user.

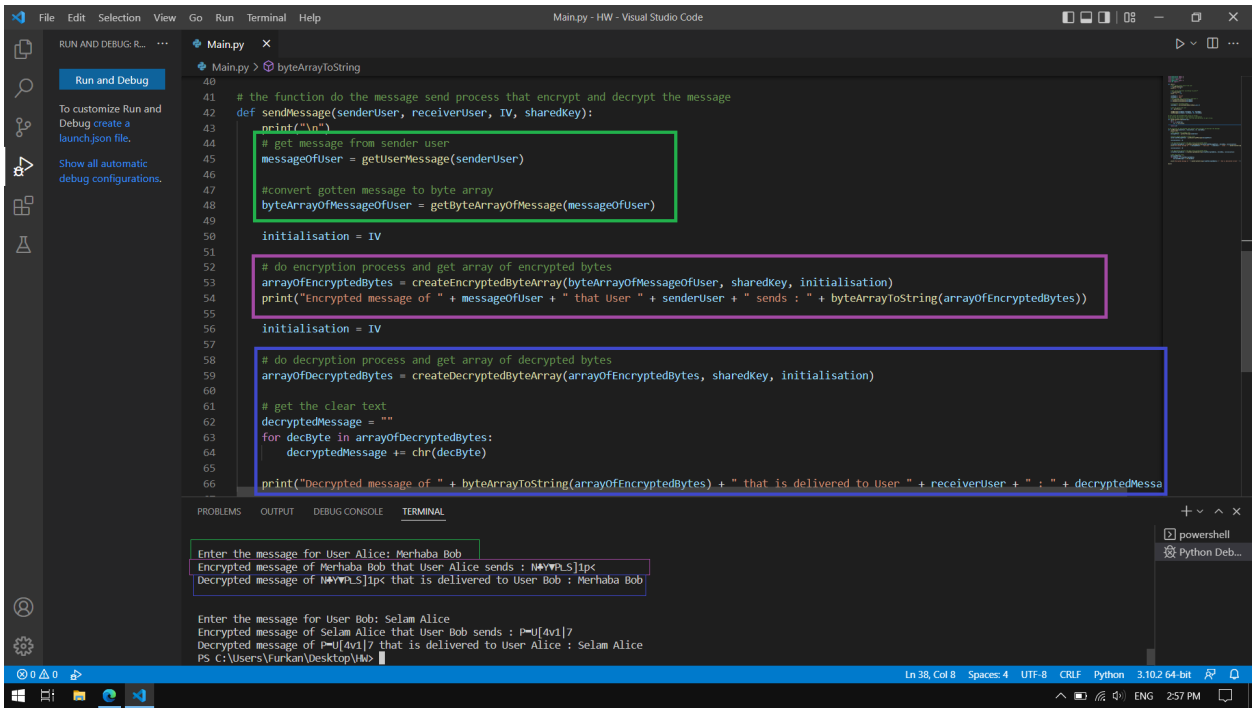
```
# the function calculates the shared key according to the given a,b,G,P values.
def calculateSharedSecretKey(a, b, G, P):
    sharedKey = int(pow(pow(G, a),b) % P)
    return sharedKey
```

After getting a and b private values, I calculate the sharedKey value

For the 5. part :

I get the IV value from user

For the 6. part : And the important part has started.



6.1 :

Firstly I get a message from User1 (Alice) using getUserMessage() and convert the string message to byte array using getByteArrayOfMessage()

6.2 & 6.3 :

I invoke the createEncryptedByteArray function to encrypt a byteArray of message. After that I print the encrypted value to the console. I invoke the createDecryptedByteArray function to decrypt encryptedByteArray. After that I print the decrypted value to the console. I do the encryption part and decryption part as you can see in the picture below

```
# the function gets the byte array of message, shared key and initialisation value (IV)
# and do xOR operation for each byte to encrypt each byte.
def createEncryptedByteArray(byteArrayOfMessageOfUser, sharedKey,initialisation):
    arrayOfEncryptedBytes = []
    for byte in byteArrayOfMessageOfUser :
        result = doXOR(initialisation, byte, sharedKey)
        initialisation = result
        arrayOfEncryptedBytes.append(result)
    return arrayOfEncryptedBytes
```

In the encryption part, I get the byte array of the message, sharedKey and initialisation value (it is IV for the first round and Ci(result of xOR process) after each round). And after each round, I append the encrypted byte to an array.

```
# the function gets the encrypted bytes array, shared key and initialisation value (IV)
# and do xOR operation for each byte to decrypt each byte.
def createDecryptedByteArray(arrayOfEncryptedBytes,sharedKey, initialisation ):
    arrayOfDecryptedBytes = []
    for i in range(0,len(arrayOfEncryptedBytes)):
        result = doXOR(initialisation, arrayOfEncryptedBytes[i], sharedKey)
        initialisation = arrayOfEncryptedBytes[i]
        arrayOfDecryptedBytes.append(result)

    return arrayOfDecryptedBytes
```

In the decryption part, I do the same thing in the encyrption. I get array of encrypted bytes, sharedKey and initialisation value (for the first round IV, after each round Mi(result of xOR process). And after each round, I append the decrypted byte to an array.

For the 7. part :

It is the same process of 6. part.

You can see the result in the console.