

## 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.

```
Main.py x Generator.py
Main.py > main
1 from GeneratorP import *
2 from GeneratorG import *
3 from Users import *
4 from DHOperator import *
5 from CBC import *
6
7 def main():
8     # get valid prime value from user
9     P = getPrimeValue()
10    print('P : ', P)
11
12    # get valid generator according to given P
13    G = generated(P)
14    print('G : ', G)
15
16    userName1 = "Alice"
17    userName2 = "Bob"
18    # get private keys from User A and User B
19    a = getUserPrivateValue(userName1)
20    b = getUserPrivateValue(userName2)
21
22    # calculate sharedSecretKey
23    sharedKey = calculateSharedSecretKey(a,b,G,P)
24
25    # get IV value from user
26
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\Furkan\Desktop\440 > & "D:\Programs\Python\Python310\python.exe" "c:\Users\Furkan\.vscode\extensions\ms-python-56288\...\.c:\Users\Furkan\Desktop\440\Main.py"
Enter the value for P: 122
P is not prime, provide a new value for P
Enter the value for P: 146
P is not prime, provide a new value for P
Enter the value for P: 173
P: 173
G: 5
Enter the private value for user Alice :
```

```
Main.py x Generator.py
Main.py > main
1 from GeneratorP import *
2 from GeneratorG import *
3 from Users import *
4 from DHOperator import *
5 from CBC import *
6
7 def main():
8     # get valid prime value from user
9     P = getPrimeValue()
10    print('P : ', P)
11
12    # get valid generator according to given P
13    G = generated(P)
14    print('G : ', G)
15
16    userName1 = "Alice"
17    userName2 = "Bob"
18    # get private keys from User A and User B
19    a = getUserPrivateValue(userName1)
20    b = getUserPrivateValue(userName2)
21
22    # calculate sharedSecretKey
23    sharedKey = calculateSharedSecretKey(a,b,G,P)
24
25    # get IV value from user
26
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\Furkan\Desktop\440 > & "D:\Programs\Python\Python310\python.exe" "c:\Users\Furkan\.vscode\extensions\ms-python-56288\...\.c:\Users\Furkan\Desktop\440\Main.py"
Enter the value for P: 173
P: 173
G: 5
Enter the private value for user Alice :
```

While we looking at the details of getPrimeValue function :

```
# 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

```
# 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.

```
# 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,
    # also 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

```
# 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%2 == 1):
        return "COMPOSITE"
    elif calculation == -1 or (calculation == 1 and K%2 == 1):
        return "PRIME"
    else:
        return "CONTINUE"
```

In the calculateValueIsPrime function, I do calculation for  $a^K \bmod P$  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.

```
def generateG(P):  
    # A generator number G is taken randomly  
    G = random.randint(2,P-1)  
  
    # find the prime factors to ensure that G guarantees 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.

```
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
```

### Example 2:

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

Testing 2 whether  
it is a generator:

$$2^{(11-1)/2} \pmod{11} = 10$$
$$2^{(11-1)/5} \pmod{11} = 4$$

Neither result is 1,  
so 2 is a generator.

Testing 3 whether  
it is a generator:

$$3^{(11-1)/2} \pmod{11} = 1$$
$$3^{(11-1)/5} \pmod{11} = 9$$

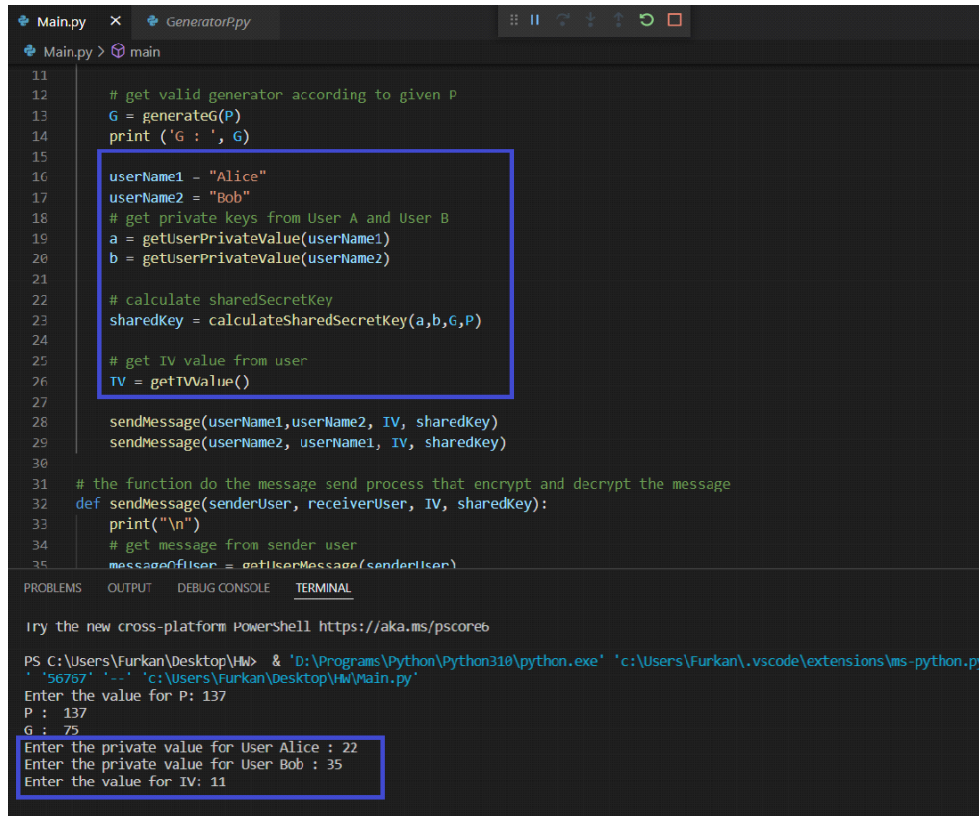
One result is 1,  
so 3 is NOT a generator.

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.

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

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



The screenshot shows a VS Code editor with a file named 'Main.py'. The code in the editor is as follows:

```
11
12 # get valid generator according to given P
13 G = generateG(P)
14 print('G : ', G)
15
16 userName1 = "Alice"
17 userName2 = "Bob"
18 # get private keys from User A and User B
19 a = getUserPrivateValue(userName1)
20 b = getUserPrivateValue(userName2)
21
22 # calculate sharedSecretKey
23 sharedKey = calculateSharedSecretKey(a,b,G,P)
24
25 # get IV value from user
26 IV = getIVValue()
27
28 sendMessage(userName1,userName2, IV, sharedKey)
29 sendMessage(userName2, userName1, IV, sharedKey)
30
31 # the function do the message send process that encrypt and decrypt the message
32 def sendMessage(senderUser, receiverUser, IV, sharedKey):
33     print("\n")
34     # get message from sender user
35     messageOfUser = getUserMessage(senderUser)
```

The terminal output at the bottom shows the execution of the script:

```
try the new cross-platform Powershell https://aka.ms/pscore6
PS C:\Users\Furkan\Desktop\HW> & 'D:\Programs\Python\Python310\python.exe' 'c:\Users\Furkan\.vscode\extensions\ms-python.p
' '56757' '--' 'c:\Users\Furkan\Desktop\HW\Main.py'
Enter the value for P: 137
P : 137
G : 75
Enter the private value for User Alice : 22
Enter the private value for User Bob : 35
Enter the value for IV: 11
```

**For the 4. part :**

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

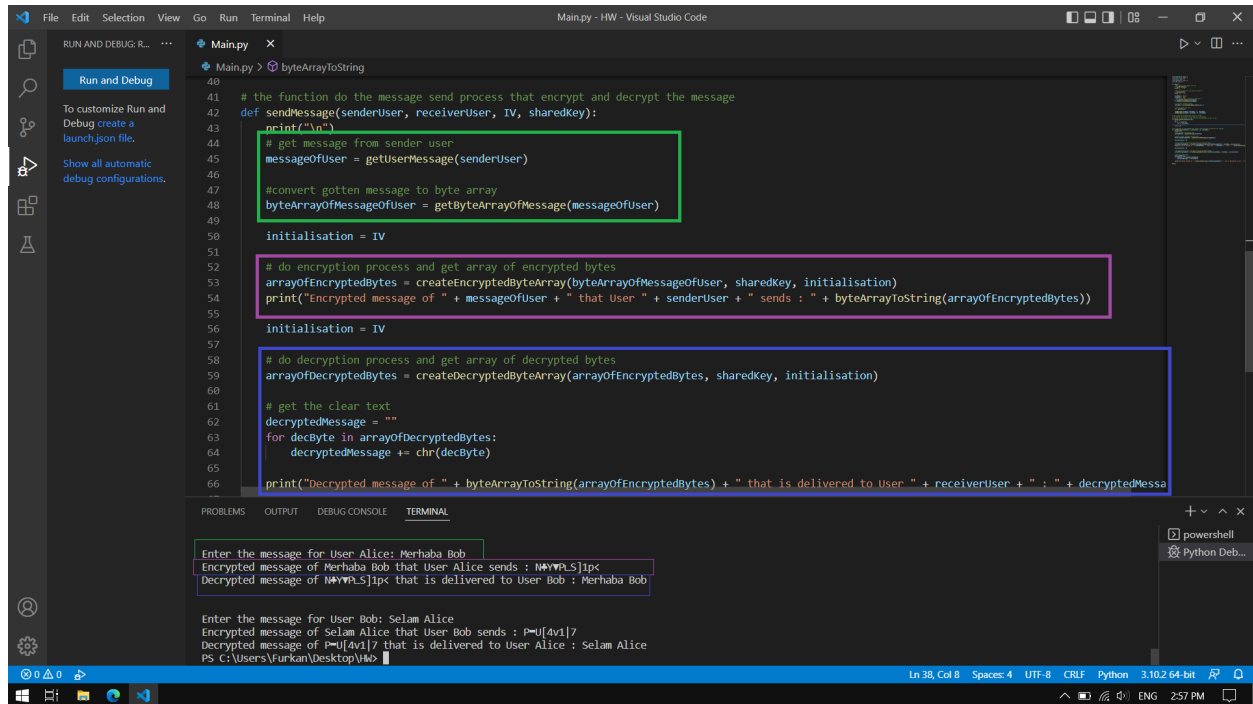
```
# the function calculates the shared key according to the given a,b,G,P values.
def calculateSharedSecretKey(a, b, G, P):
    sharedKey = int(pow(pow(G, a), b) % P)
    return sharedKey
```

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.



```
40
41 # the function do the message send process that encrypt and decrypt the message
42 def sendMessage(senderUser, receiverUser, IV, sharedKey):
43     print("\n")
44     # get message from sender user
45     messageOfUser = getUserMessage(senderUser)
46
47     #convert gotten message to byte array
48     byteArrayOfMessageOfUser = getByteArrayOfMessage(messageOfUser)
49
50     initialisation = IV
51
52     # do encryption process and get array of encrypted bytes
53     arrayOfEncryptedBytes = createEncryptedByteArray(byteArrayOfMessageOfUser, sharedKey, initialisation)
54     print("Encrypted message of " + messageOfUser + " that User " + senderUser + " sends : " + byteArrayToString(arrayOfEncryptedBytes))
55
56     initialisation = IV
57
58     # do decryption process and get array of decrypted bytes
59     arrayOfDecryptedBytes = createDecryptedByteArray(arrayOfEncryptedBytes, sharedKey, initialisation)
60
61     # get the clear text
62     decryptedMessage = ""
63     for decByte in arrayOfDecryptedBytes:
64         decryptedMessage += chr(decByte)
65
66     print("Decrypted message of " + byteArrayToString(arrayOfEncryptedBytes) + " that is delivered to User " + receiverUser + " : " + decryptedMessage)
```

Enter the message for User Alice: Merhaba Bob  
Encrypted message of Merhaba Bob that User Alice sends : H4VWLSJlpc  
Decrypted message of H4VWLSJlpc that is delivered to User Bob : Merhaba Bob

Enter the message for User Bob: Selam Alice  
Encrypted message of Selam Alice that User Bob sends : P=U4vI7  
Decrypted message of P=U4vI7 that is delivered to User Alice : Selam Alice

## 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.

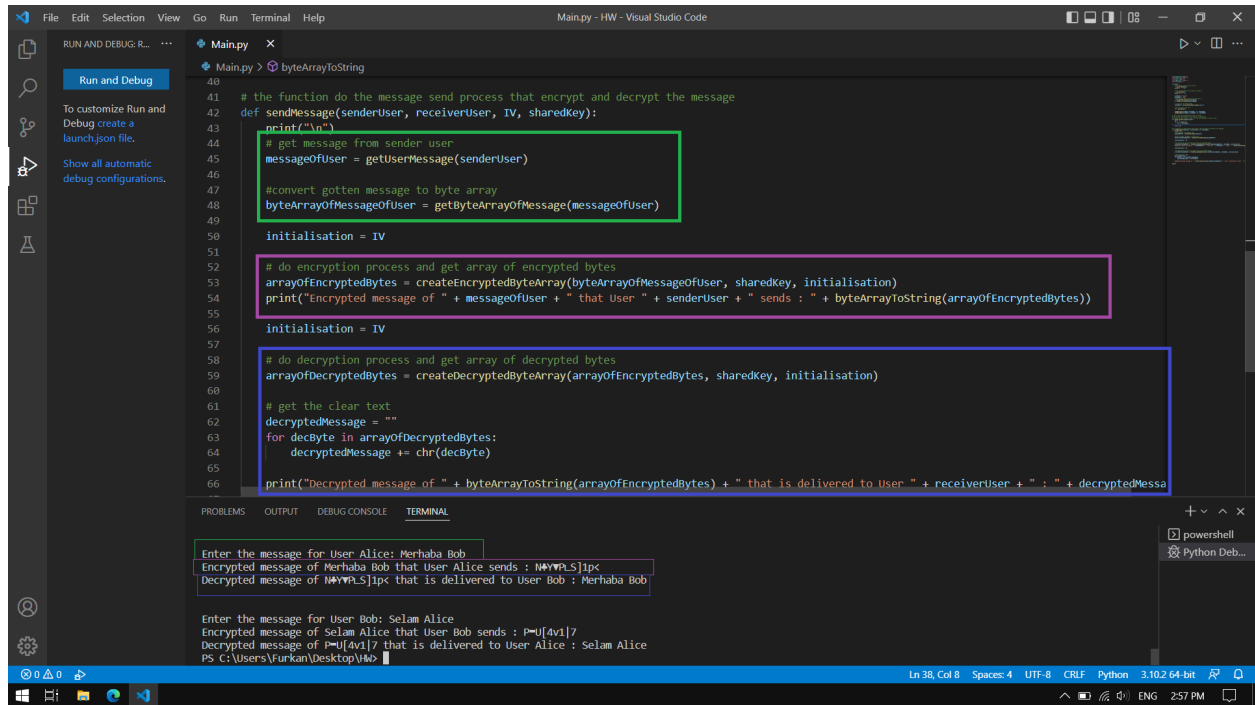
```
# the function gets the encrypted bytes array, shared key and initialisation value (IV)
# and do XOR operation for each byte to decrypt each byte.
def createDecryptedByteArray(arrayOfEncryptedBytes,sharedKey, initialisation ):
    arrayOfDecryptedBytes = []
    for i in range(0,len(arrayOfEncryptedBytes)):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



```
40
41 # the function do the message send process that encrypt and decrypt the message
42 def sendMessage(senderUser, receiverUser, IV, sharedKey):
43     print("\n")
44     # get message from sender user
45     messageOfUser = getUserMessage(senderUser)
46
47     # convert gotten message to byte array
48     byteArrayOfMessageOfUser = getByteArrayOfMessage(messageOfUser)
49
50     initialisation = IV
51
52     # do encryption process and get array of encrypted bytes
53     arrayOfEncryptedBytes = createEncryptedByteArray(byteArrayOfMessageOfUser, sharedKey, initialisation)
54     print("Encrypted message of " + messageOfUser + " that User " + senderUser + " sends : " + byteArrayToString(arrayOfEncryptedBytes))
55
56     initialisation = IV
57
58     # do decryption process and get array of decrypted bytes
59     arrayOfDecryptedBytes = createDecryptedByteArray(arrayOfEncryptedBytes, sharedKey, initialisation)
60
61     # get the clear text
62     decryptedMessage = ""
63     for decByte in arrayOfDecryptedBytes:
64         decryptedMessage += chr(decByte)
65
66     print("Decrypted message of " + byteArrayToString(arrayOfEncryptedBytes) + " that is delivered to User " + receiverUser + " : " + decryptedMessage)
```

Enter the message for User Alice: Merhaba Bob  
Encrypted message of Merhaba Bob that User Alice sends : H4VWLSjipx  
Decrypted message of H4VWLSjipx that is delivered to User Bob : Merhaba Bob

Enter the message for User Bob: Selam Alice  
Encrypted message of Selam Alice that User Bob sends : P=U[4vI]7  
Decrypted message of P=U[4vI]7 that is delivered to User Alice : Selam Alice  
PS C:\Users\Furkan\Desktop\440