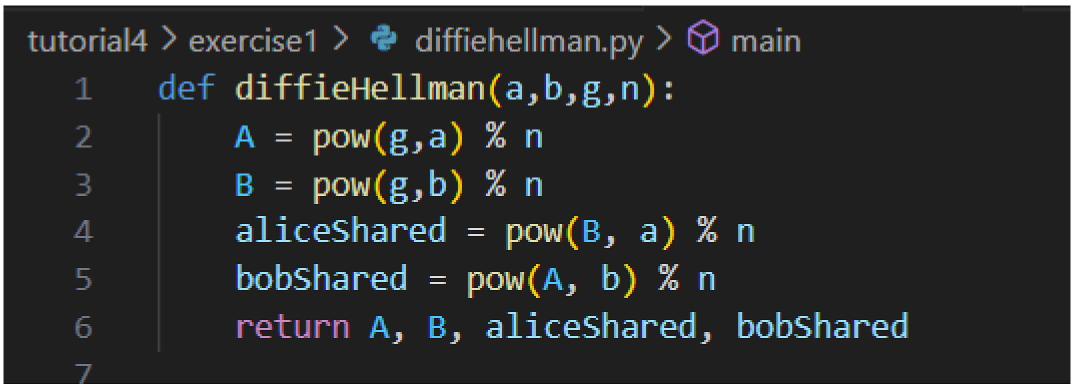
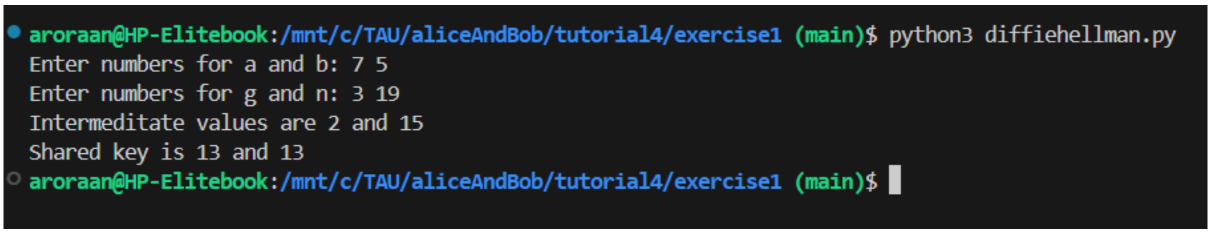
Exercise 1

- Task 1



```
tutorial4 > exercise1 > diffiehellman.py > main
def diffieHellman(a,b,g,n):
    A = pow(g,a) % n
    B = pow(g,b) % n
    aliceShared = pow(B, a) % n
    bobShared = pow(A, b) % n
    return A, B, aliceShared, bobShared
```

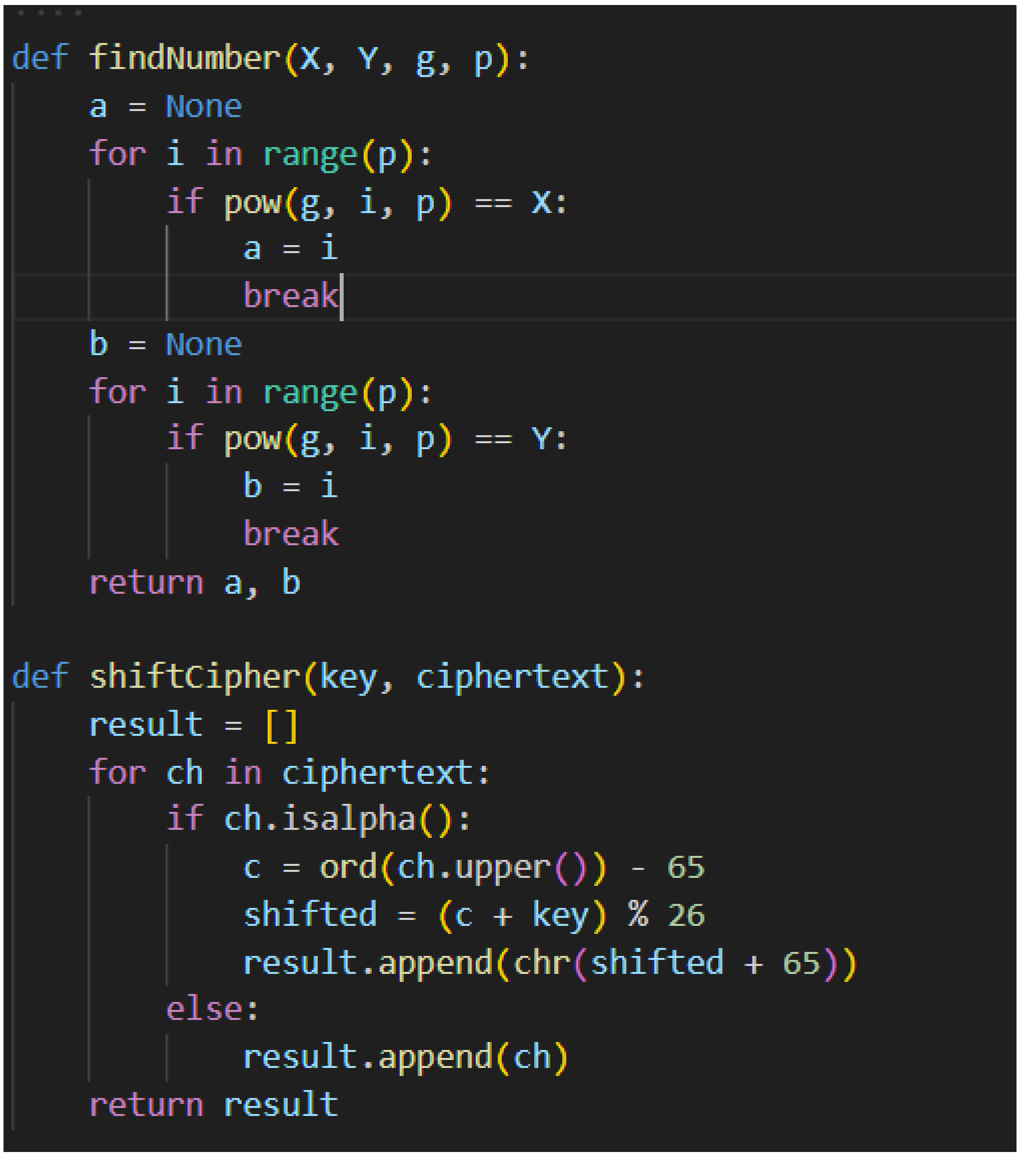
The intermediate values can be computed by making a python function.



```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial4/exercise1 (main)$ python3 diffiehellman.py
Enter numbers for a and b: 7 5
Enter numbers for g and n: 3 19
Intermeditate values are 2 and 15
Shared key is 13 and 13
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial4/exercise1 (main)$ 
```

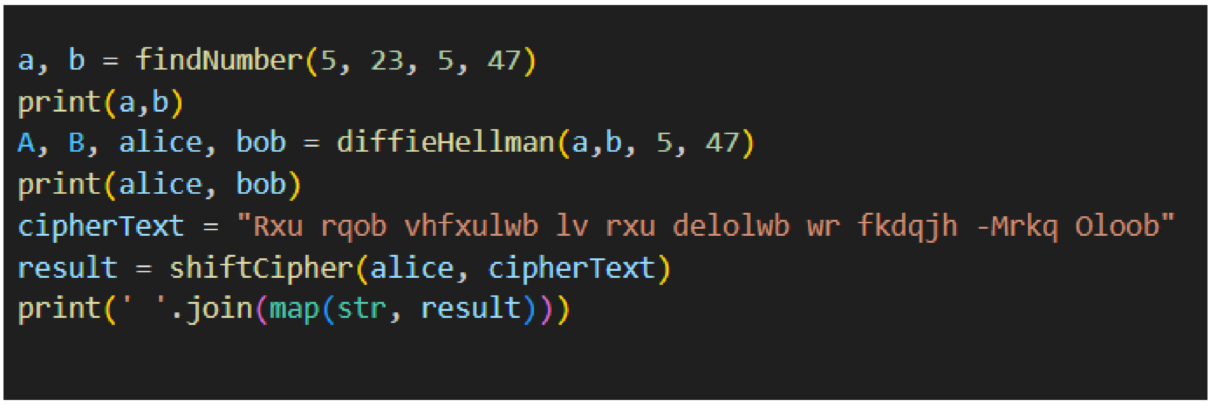
- Task 2

For task 2 we have g = 5, p = 47 X= 99 (not modulo) and Y=23 (has been moduloed)
99 mod 47 = 5 so use X as 5.

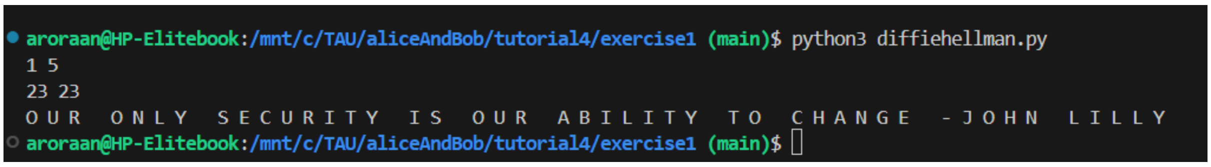


```python
def findNumber(X, Y, g, p):
    a = None
    for i in range(p):
        if pow(g, i, p) == X:
            a = i
            break
    b = None
    for i in range(p):
        if pow(g, i, p) == Y:
            b = i
            break
    return a, b

def shiftCipher(key, ciphertext):
    result = []
    for ch in ciphertext:
        if ch.isalpha():
            c = ord(ch.upper()) - 65
            shifted = (c + key) % 26
            result.append(chr(shifted + 65))
        else:
            result.append(ch)
    return result
```

- Task 3, using Key as 23.



```python
a, b = findNumber(5, 23, 5, 47)
print(a,b)
A, B, alice, bob = diffieHellman(a,b, 5, 47)
print(alice, bob)
cipherText = "Rxu rqob vhfxulwb lv rxu delolwb wr fkdqjh -Mrkq Oloob"
result = shiftCipher(alice, cipherText)
print(' '.join(map(str, result)))
```



```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial4/exercise1 (main)$ python3 diffiehellman.py
1 5
23 23
O U R   O N L Y   S E C U R I T Y   I S   O U R   A B I L I T Y   T O   C H A N G E   - J O H N   L I L L Y
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial4/exercise1 (main)$
```

Exercise 2

To design a protocol that allows Alice (A), Bob (B), and Charlie (C) to exchange a single symmetric key *K* using an extended Diffie-Hellman (DH) key exchange, we aim to meet the following conditions:

Key Features:

Only A, B, and C access the key.

Key integrity is verified by all.

Minimize the number of messages.

Proposed protocol:

Public variables – Just like DH we keep prime p and generator g. These values are public and can be reused.

Private variables – All 3, A, B and C select a random private a,b and c value. This value is not shared and kept only with them.

Intermediate values –

Alice computes – A = g^a mod p and sends it to Bob

Bob computes – B = g^b mod p and sends it to Charlie

Charlie computes – C = g^c mod p and sends it to Alice

Key Exchange –

Alice computes Kca = C^a mod p and sends it to Bob

Bob computes Kab = A^b mod p and sends it to Charlie

Charlie computes Kbc = B^c mod p and sends it to Alice

Again

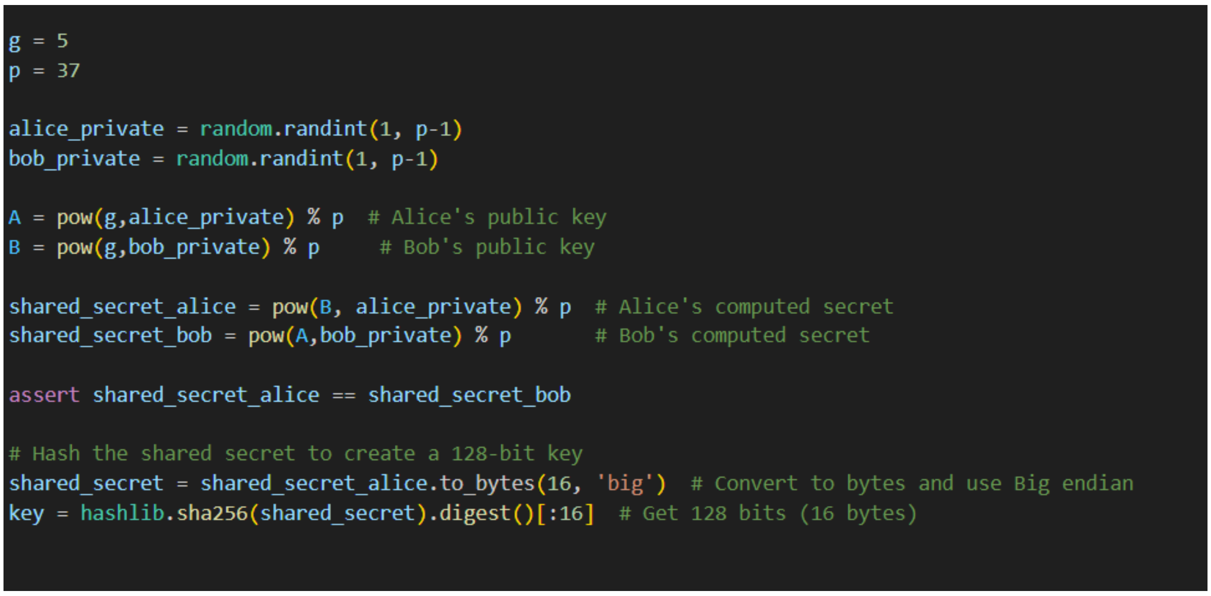
Alice computes Kcab = (C^a)^b mod p which is C^ab mod p which is g^cab mod p

Bob computes Kabc = g^abc mod p

Charlie computes Kbca = g^bca mod p

This way every one gets the same key computed. The total number of messages is 6 this can be reduced by involving a trusted party like 'Trent' who would create a common key and send it to everyone.

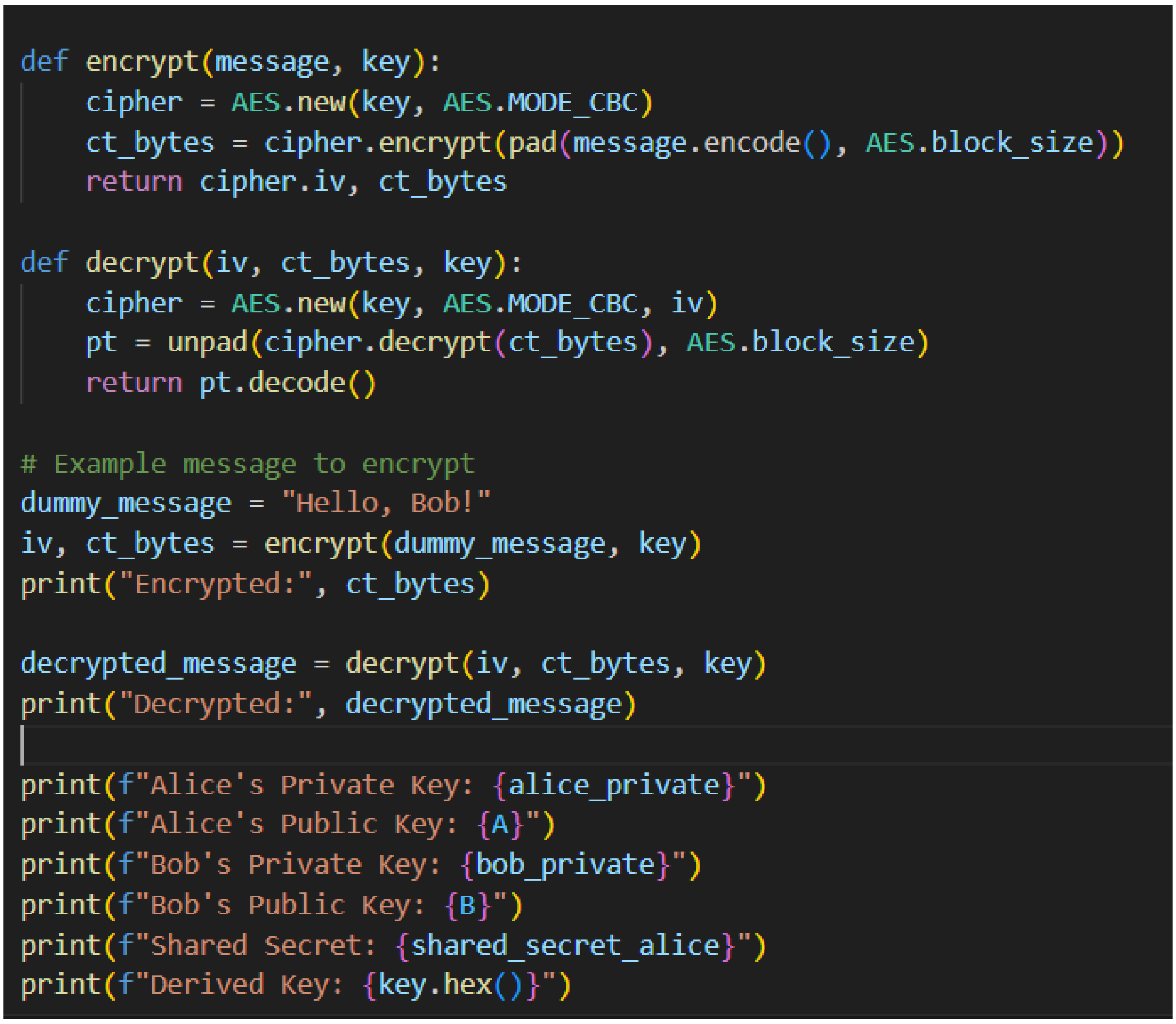
Exercise 3



```
g = 5
p = 37

alice_private = random.randint(1, p-1)
bob_private = random.randint(1, p-1)

A = pow(g,alice_private) % p  # Alice's public key
B = pow(g,bob_private) % p     # Bob's public key

shared_secret_alice = pow(B, alice_private) % p  # Alice's computed secret
shared_secret_bob = pow(A,bob_private) % p       # Bob's computed secret

assert shared_secret_alice == shared_secret_bob

# Hash the shared secret to create a 128-bit key
shared_secret = shared_secret_alice.to_bytes(16, 'big')  # Convert to bytes and use Big endian
key = hashlib.sha256(shared_secret).digest()[:16]  # Get 128 bits (16 bytes)
```

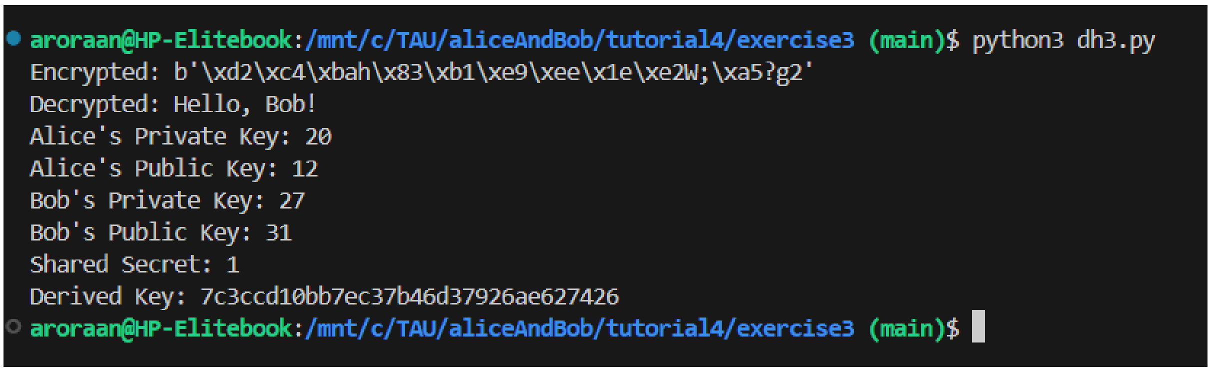
Taking values of g and p, using random to get values for a and b for alice and bob.

Using hashlib to compute sha256 hash and then take 128 bits (16bytes) from it.



```python
def encrypt(message, key):
    cipher = AES.new(key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(message.encode(), AES.block_size))
    return cipher.iv, ct_bytes

def decrypt(iv, ct_bytes, key):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct_bytes), AES.block_size)
    return pt.decode()

# Example message to encrypt
dummy_message = "Hello, Bob!"
iv, ct_bytes = encrypt(dummy_message, key)
print("Encrypted:", ct_bytes)

decrypted_message = decrypt(iv, ct_bytes, key)
print("Decrypted:", decrypted_message)

print(f"Alice's Private Key: {alice_private}")
print(f"Alice's Public Key: {A}")
print(f"Bob's Private Key: {bob_private}")
print(f"Bob's Public Key: {B}")
print(f"Shared Secret: {shared_secret_alice}")
print(f"Derived Key: {key.hex()}")
```

Encrypt and decrypt functions, dummy message. All computed values of a run.



```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial4/exercise3 (main)$ python3 dh3.py
Encrypted: b'\xd2\xc4\xbah\x83\xb1\xe9\xee\x1e\xe2W;\xa5?g2'
Decrypted: Hello, Bob!
Alice's Private Key: 20
Alice's Public Key: 12
Bob's Private Key: 27
Bob's Public Key: 31
Shared Secret: 1
Derived Key: 7c3ccd10bb7ec37b46d37926ae627426
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial4/exercise3 (main)$
```

The security of a D-H key exchange protocol is based on?

The security of the Diffie-Hellman key exchange is based on the difficulty of solving the discrete logarithm problem (DLP) in a finite cyclic group. This means that given g, p, and g^^a modp , it is computationally hard to determine a.

The D-H key exchange protocol is vulnerable to?

The Diffie-Hellman protocol is vulnerable to man-in-the-middle (MITM) attacks, where an attacker intercepts the public keys exchanged between Alice and Bob and replaces them with their own, thus establishing separate shared keys with both parties. To prevent this, additional mechanisms like digital signatures or certificates are used to verify the authenticity of the public keys.

What is the condition for selecting a good value of pp in a D-H key exchange protocol?

The prime number p should be a large prime (usually at least 2048 bits) to make the discrete logarithm problem hard enough. Additionally, p should be chosen such that $(p-1)/2$ is also prime (a safe prime) to improve security against certain attacks.