Batch: AC-1 Roll No: 16010121045

Experiment No. 2

Title:	**Title: Implementation of Diffie Hellman Key exchange Algorithm**

Objective:

The objective of this experiment is to write a program for the Diffie Hellman Key exchange algorithm and verify if the secret key computed by Alice and Bob are the same or not.

Expected Outcome of Experiment:

CO	Outcome
3	Comprehend cryptographic hash functions, Message Authentication Codes and Digital Signatures for Authentication

Books/ Journals/ Websites referred:

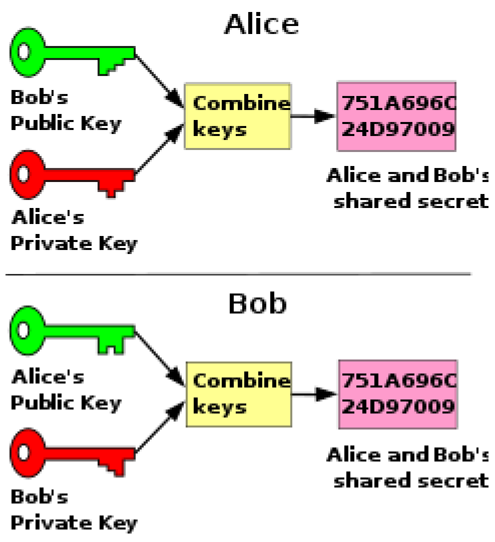
Abstract:-

The Diffie–Hellman (DH) Algorithm is a key-exchange protocol that enables two parties communicating over public channel to establish a mutual secret without it being transmitted over the Internet. DH enables the two to use a public key to encrypt and decrypt their conversation or data using symmetric cryptography.

The sender and receiver don't need any prior knowledge of each other. Once the keys are exchanged, the communication of data can be done through an insecure channel. The sharing of the secret key is safe.

Related Theory: -

Diffie–Hellman key exchange is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as conceived by Ralph Merkle and named after Whitfield Diffie and Martin Hellman. DH is one of the earliest practical examples of public key exchange implemented within the field of cryptography. Published in 1976 by Diffie and Hellman, this is the earliest publicly known work that proposed the idea of a private key and a corresponding public key.



Diffie–Hellman key exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network. An analogy illustrates the concept of public key exchange by using colours instead of very large numbers.

The process begins by having the two parties, Alice and Bob, publicly agree on an arbitrary starting colour that does not need to be kept secret (but should be different every time). In this example, the colour is yellow. Each person also selects a secret colour that they keep to themselves – in this case, red and blue-green. The crucial part of the process is that Alice and Bob each mix their own secret colour together with their mutually shared colour, resulting in orange-tan and light-blue mixtures respectively, and then publicly exchange the two mixed colours. Finally, each of them mixes the colour they received from the partner with their own private colour. The result is a final colour mixture (yellow-brown in this case) that is identical to the partner's final colour mixture.

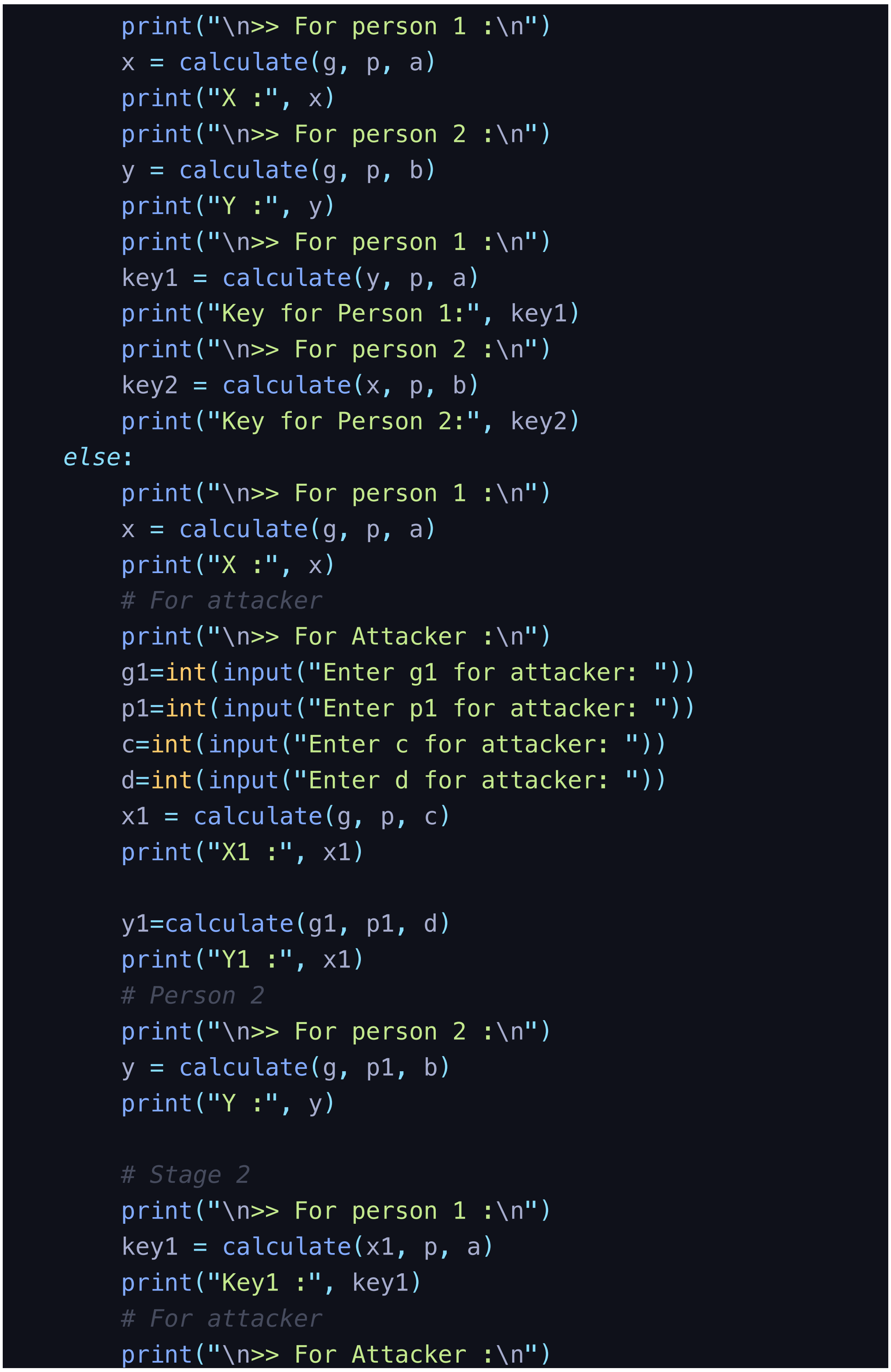
If a third party listened to the exchange, it would only know the common colour (yellow) and the first mixed colours (orange-tan and light-blue), but it would be difficult for this party to determine the final secret colour (yellow-brown). Bringing the analogy back to a real-life exchange using large numbers rather than colours, this determination is computationally expensive. It is impossible to compute in a practical amount of time even for modern supercomputers.

Diffie -Hellman algorithm:

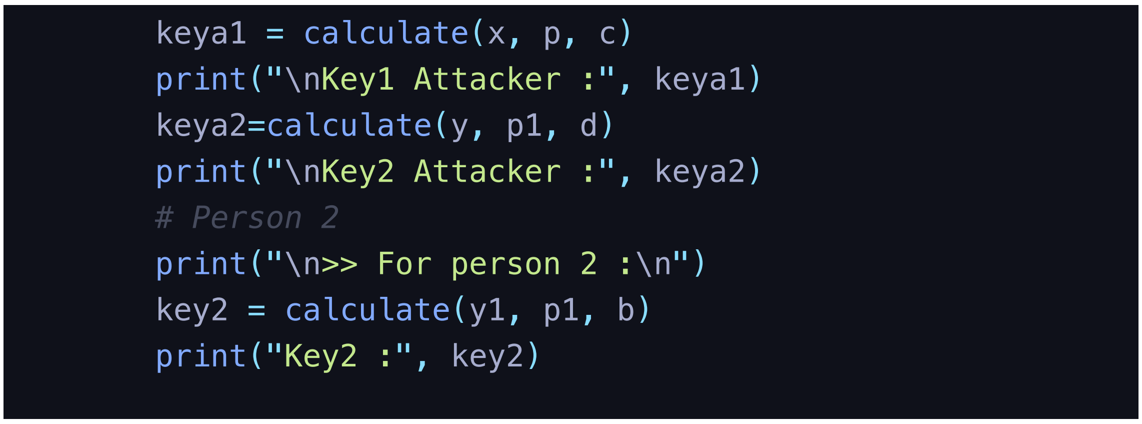
Alice	**Bob**
Public Keys available = P, G	**Public Keys available = P, G**
Private Key Selected = a	**Private Key Selected = b**
Key generated = $x = G^a mod P$	**Key generated =** $y = G^b mod P$
Exchange of generated keys takes place	
Key received = y	**key received = x**
Generated Secret Key = $k_a = y^a mod P$	**Generated Secret Key =** $k_b = x^b mod P$
Algebraically, it can be shown that $k_a = k_b$	
Users now have a symmetric secret key to encrypt	

Program:

```python
from random import random


def primeRootModulo(g, p):
    li = []
    for i in range(1, p):
        li.append(pow(g, i) % p)
    li.sort()
    print(li)
    for i in range(1, p):
        if not (li[i-1] == i):
            return False
    return True


def isprime(p):
    for i in range(2, (p//2)+1):
        if (p % i == 0):
            return False
    return True


def calculate(g, p, a):
    print("g ^ a mod p :", end=" ")
    print(g, "^", a, "mod", p, "")
    return pow(g, a) % p


g = int(input(">> Enter g : "))
p = int(input(">> Enter p a prime number : "))
a = int(input(">> Enter a for Person 1 : "))
b = int(input(">> Enter b for Person 2 :"))
mim=input(">> Do you want to display Man In the Middle? (y/n)
: ")
if not (isprime(p) and a < p and b < p):
    print(">> The numbers selected are not applicable")
else:
    if(mim.lower()=='n'):
```

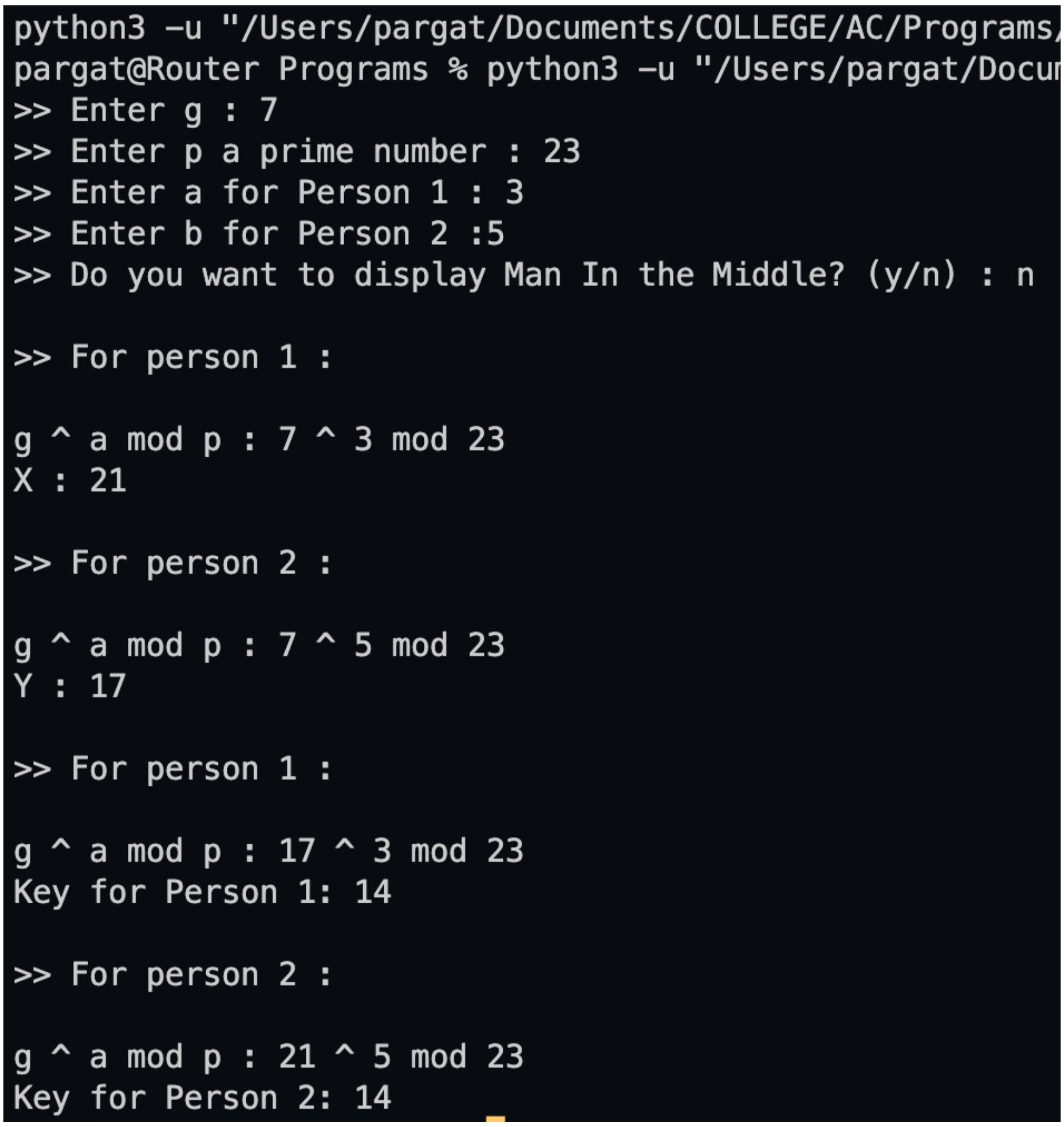


```
        print("\n>> For person 1 :\n")
        x = calculate(g, p, a)
        print("X :", x)
        print("\n>> For person 2 :\n")
        y = calculate(g, p, b)
        print("Y :", y)
        print("\n>> For person 1 :\n")
        key1 = calculate(y, p, a)
        print("Key for Person 1:", key1)
        print("\n>> For person 2 :\n")
        key2 = calculate(x, p, b)
        print("Key for Person 2:", key2)
    else:
        print("\n>> For person 1 :\n")
        x = calculate(g, p, a)
        print("X :", x)
        # For attacker
        print("\n>> For Attacker :\n")
        g1=int(input("Enter g1 for attacker: "))
        p1=int(input("Enter p1 for attacker: "))
        c=int(input("Enter c for attacker: "))
        d=int(input("Enter d for attacker: "))
        x1 = calculate(g, p, c)
        print("X1 :", x1)

        y1=calculate(g1, p1, d)
        print("Y1 :", x1)
        # Person 2
        print("\n>> For person 2 :\n")
        y = calculate(g, p1, b)
        print("Y :", y)

        # Stage 2
        print("\n>> For person 1 :\n")
        key1 = calculate(x1, p, a)
        print("Key1 :", key1)
        # For attacker
        print("\n>> For Attacker :\n")
```

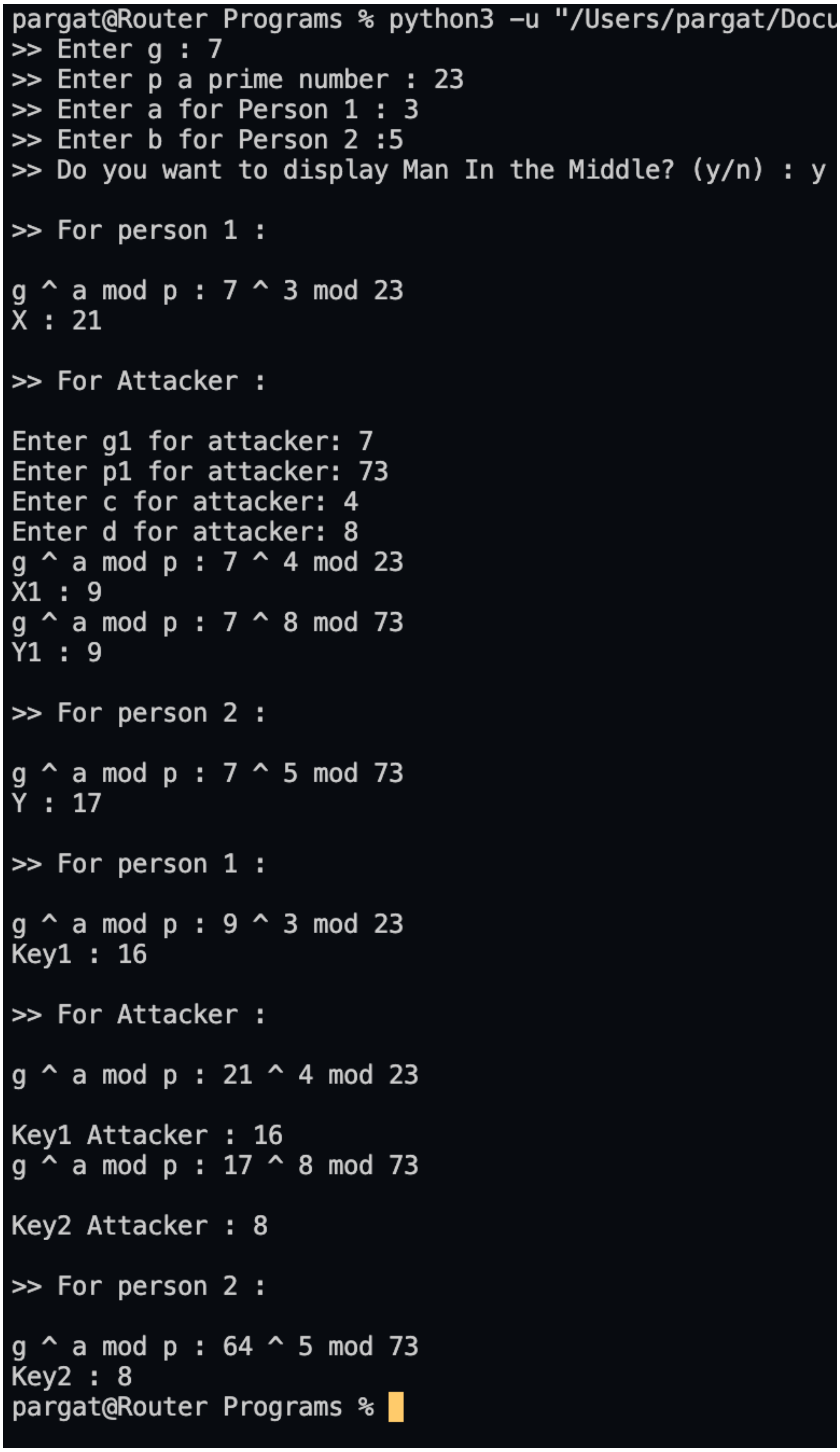


```
        keya1 = calculate(x, p, c)
        print("\nKey1 Attacker :", keya1)
        keya2=calculate(y, p1, d)
        print("\nKey2 Attacker :", keya2)
        # Person 2
        print("\n>> For person 2 :\n")
        key2 = calculate(y1, p1, b)
        print("Key2 :", key2)
```

Output Screenshots:



```
python3 -u "/Users/pargat/Documents/COLLEGE/AC/Programs
pargat@Router Programs % python3 -u "/Users/pargat/Docu
>> Enter g : 7
>> Enter p a prime number : 23
>> Enter a for Person 1 : 3
>> Enter b for Person 2 :5
>> Do you want to display Man In the Middle? (y/n) : n

>> For person 1 :

g ^ a mod p : 7 ^ 3 mod 23
X : 21

>> For person 2 :

g ^ a mod p : 7 ^ 5 mod 23
Y : 17

>> For person 1 :

g ^ a mod p : 17 ^ 3 mod 23
Key for Person 1: 14

>> For person 2 :

g ^ a mod p : 21 ^ 5 mod 23
Key for Person 2: 14
```



```
pargat@Router Programs % python3 -u "/Users/pargat/Docu
>> Enter g : 7
>> Enter p a prime number : 23
>> Enter a for Person 1 : 3
>> Enter b for Person 2 :5
>> Do you want to display Man In the Middle? (y/n) : y

>> For person 1 :

g ^ a mod p : 7 ^ 3 mod 23
X : 21

>> For Attacker :

Enter g1 for attacker: 7
Enter p1 for attacker: 73
Enter c for attacker: 4
Enter d for attacker: 8
g ^ a mod p : 7 ^ 4 mod 23
X1 : 9
g ^ a mod p : 7 ^ 8 mod 73
Y1 : 9

>> For person 2 :

g ^ a mod p : 7 ^ 5 mod 73
Y : 17

>> For person 1 :

g ^ a mod p : 9 ^ 3 mod 23
Key1 : 16

>> For Attacker :

g ^ a mod p : 21 ^ 4 mod 23

Key1 Attacker : 16
g ^ a mod p : 17 ^ 8 mod 73

Key2 Attacker : 8

>> For person 2 :

g ^ a mod p : 64 ^ 5 mod 73
Key2 : 8
pargat@Router Programs %
```

Conclusion:-

Comprehended cryptographic hash functions, Message Authentication Codes and Digital Signatures for Authentication and successfully executed the given task.

Postlab:

1. **Comment on weakness(s) of Diffie-Hellman scheme**

   One weakness in Diffie-Hellman scheme is that this implementation is susceptible to Man-in-the-middle attacks (MITM). This is because of lack of authentication in the implementation of this algorithm.

   Encryption of information cannot be performed with the help of this algorithm.

   Digital signature cannot be signed using this algorithm.

2. **Suggest at least two methods to eliminate the weakness(s) of D-H scheme.**

   We can use some additional layer of security to check for data integrity and to check whether the give values have been modified or not this helps us to identify whether there has been a breach in the transmitted data.

   An authentication layer can be implemented to check and authenticate the other receiver.