EXPERIMENT 03

Aim: Implementation of Diffie Hellman Key exchange algorithm
Theory:

- **Diffie-Hellman key exchange algorithm:**

The Diffie-Hellman key exchange algorithm is a method for two parties to establish a shared secret over an insecure channel, which can then be used for secure communication. It allows two parties, typically named Alice and Bob, to jointly compute a shared secret without ever exchanging it directly. The key idea behind Diffie-Hellman is the use of modular exponentiation in a finite field.
Here's a brief description of how the algorithm works:
1. Both Alice and Bob agree on a large prime number `p` and a primitive root `g` modulo `p`. These parameters `p` and `g` are public and can be shared over the insecure channel.
2. Alice chooses a private key `a`, computes `A = g^a mod p`, and sends `A` to Bob.
3. Bob chooses a private key `b`, computes `B = g^b mod p`, and sends `B` to Alice.
4. Now, both Alice and Bob can compute the shared secret:
   - Alice computes `shared_secret = B^a mod p`.
   - Bob computes `shared_secret = A^b mod p`.

   Since `B^a mod p = (g^b)^a mod p = g^(ba) mod p` and `A^b mod p = (g^a)^b mod p = g^(ab) mod p`, both parties end up with the same shared secret `g^(ab) mod p`.
5. The shared secret obtained by both Alice and Bob can be used as a symmetric key for secure communication, allowing them to encrypt and decrypt messages exchanged between them.

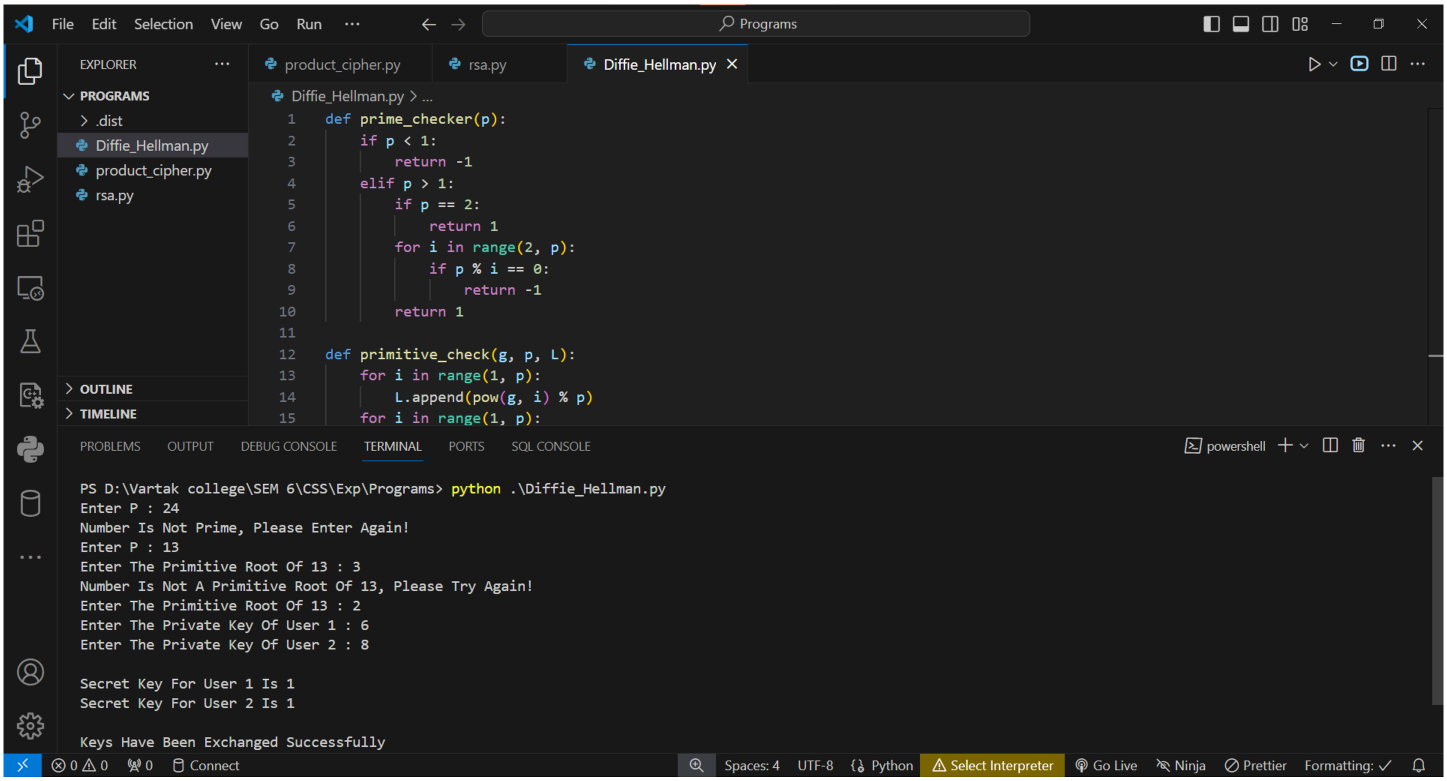
Diffie-Hellman key exchange provides a way for two parties to establish a shared secret even if an eavesdropper intercepts all communications between them. This is because the shared secret is never exchanged directly and relies on the computational difficulty of solving the discrete logarithm problem in a finite field.

Code:

```
def prime_checker(p):
    if p < 1:
        return -1
    elif p > 1:
        if p == 2:
            return 1
        for i in range(2, p):
            if p % i == 0:
                return -1
        return 1

def primitive_check(g, p, L):
    for i in range(1, p):
```

```
        L.append(pow(g, i) % p)
    for i in range(1, p):
        if L.count(i) > 1:
            L.clear()
            return -1
    return 1

l = []
while True:
    P = int(input("Enter P : "))
    if prime_checker(P) == -1:
        print("Number Is Not Prime, Please Enter Again!")
        continue
    break

while True:
    G = int(input(f"Enter The Primitive Root Of {P} : "))
    if primitive_check(G, P, l) == -1:
        print(f"Number Is Not A Primitive Root Of {P}, Please Try Again!")
        continue
    break

# Private Keys
x1, x2 = int(input("Enter The Private Key Of User 1 : ")), int(input("Enter The Private Key Of User 2 : "))
while True:
    if x1 >= P or x2 >= P:
        print(f"Private Key Of Both The Users Should Be Less Than {P}!")
        continue
    break

y1, y2 = pow(G, x1) % P, pow(G, x2) % P

k1, k2 = pow(y2, x1) % P, pow(y1, x2) % P

print(f"\nSecret Key For User 1 Is {k1}\nSecret Key For User 2 Is {k2}\n")

if k1 == k2:
    print("Keys Have Been Exchanged Successfully")
else:
    print("Keys Have Not Been Exchanged Successfully")
```

Output:



Conclusion:

The Diffie-Hellman key exchange algorithm was implemented, allowing two parties to establish a shared secret over an insecure channel. The algorithm relies on the computational difficulty of solving the discrete logarithm problem in a finite field, ensuring security even if an eavesdropper intercepts all communications. By agreeing on a large prime number p and a primitive root g modulo p, both parties compute their public keys and derive a shared secret using modular exponentiation. The shared secret can then be used as a symmetric key for secure communication.