Walchand College of Engineering, Sangli ***(Government Aided Autonomous Institute)***	
AY 2025-26	
Course Information	
Programme	B.Tech. (Computer Science and Engineering)
Class, Semester	Final Year B. Tech., Sem VII
Course Code	6CS451
Course Name	Cryptography and Network Security Lab
PRN	22510016

Experiment No. 08

Title – Implement the Diffie-Hellman Key Exchange algorithm for a given problem.

Objectives:

To implement the ***Diffie-Hellman Key Exchange Algorithm*** to enable two parties to securely share a secret key over an insecure communication channel. This shared key can then be used for symmetric encryption or decryption in secure communication.

Problem Statement:

In secure communications, two parties (commonly referred to as Alice and Bob) need to agree on a secret key that can be used for encrypting and decrypting messages. However, they must do this over a public channel where an attacker (Eve) might be listening.

Implement the ***Diffie-Hellman Key Exchange Algorithm***, which allows Alice and Bob to securely compute a shared secret key without directly transmitting it over the insecure channel. The algorithm should:

1. Accept a large prime number p and a primitive root modulo p, g.
2. Allow each party to select a private key (a for Alice, b for Bob).
3. Compute the corresponding public keys:
   - Alice computes A = g^a mod p
   - Bob computes B = g^b mod p
4. Exchange public keys between Alice and Bob.
5. Compute the shared secret key:
   - Alice computes K = B^a mod p
   - Bob computes K = A^b mod p
6. Validate that both computedsecretkeysareequal,i.e., `K_Alice == K_Bob`.
7. Additionally, demonstrate the correctness of the algorithm with an example and optionally simulate an attacker attempting to derive the secret key without access to the private keys.

Equipment/Tools:

- Hardware*:* Computer System
- Software*:* Python
- Tools: Code Editor – VS Code

Theory:

The Diffie-Hellman Key Exchange is a public-key cryptographic algorithm that enables two parties to establish a shared secret over an insecure channel.
It is based on the difficulty of computing discrete logarithms in modular arithmetic.

Mathematical Steps:

1. Choose a large prime numberpanda primitive root modulo p, g.
2. Alice chooses a private keya,andBob chooses a private key b.
3. They compute their publickeys:
   - Alice: A = g^a mod p
   - Bob: B = g^b mod p
4. Alice and Bob exchangepublickeys.
5. Each computes the sharedsecret:
   - Alice: K = B^amodp
   - Bob: K = A^bmodp
6. Both results are equal:
   K_Alice=K_Bob = g^(ab)modp

This shared key can now be used forsymmetric encryption.

Procedure:

1. Input a large ***primenumber (p)*** and a ***primitive root (g)***.
2. Input ***private keys***for Alice and Bob (a and b).
3. Compute ***public keys***:
   ```
   A = g^a mod p
   B = g^b mod p
   ```
4. Exchange ***public keys***.
5. Compute ***shared secrets***:
   ```
   Alice: K_Alice = B^a mod p
   Bob: K_Bob = A^b mod p
   ```
6. Verify that both secret keys are ***equal***.
7. Simulate an attacker (Eve) trying to compute the secret without private keys.

Steps:

```python
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def diffie_hellman(p, g, a, b):
    if not is_prime(p):
        print("Error: p must be a prime number.")
        return
    if g <= 1 or g >= p:
        print("Error: g must be between 2 and p-1.")
        return
    if not (1 <= a < p - 1) or not (1 <= b < p - 1):
        print("Error: Private keys (a, b) must be between 1 and p-1.")
        return
    print("\n--- Input is Valid ---")
    print(f"Prime number (p): {p}")
    print(f"Primitive root (g): {g}")
    print(f"Alice's private key (a): {a}")
    print(f"Bob's private key (b): {b}")
    A = pow(g, a, p)
    B = pow(g, b, p)
    print("\n--- Public Keys ---")
    print(f"Alice's Public Key (A) = {A}")
    print(f"Bob's Public Key   (B) = {B}")

    K_Alice = pow(B, a, p)
    K_Bob = pow(A, b, p)

    print("\n--- Shared Secret Computation ---")
    print(f"Alice's Shared Secret (K_Alice) = {K_Alice}")
    print(f"Bob's Shared Secret   (K_Bob)   = {K_Bob}")
```

```python
    if K_Alice == K_Bob:
        print("\nShared secret key successfully established!")
        print(f"Final Shared Secret Key = {K_Alice}")
    else:
        print("\nKeys do not match. Error in computation!")
    print()

print("=== Diffie-Hellman Key Exchange ===")

p = int(input("Enter a large prime number (p): "))
g = int(input("Enter a primitive root modulo p (g): "))
a = int(input("Enter Alice's private key (a): "))
b = int(input("Enter Bob's private key (b): "))

diffie_hellman(p, g, a, b)
```

```
=== Diffie-Hellman Key Exchange ===
Enter a large prime number (p): 23
Enter a primitive root modulo p (g): 5
Enter Alice's private key (a): 6
Enter Bob's private key (b): 15

--- Input is Valid ---
Prime number (p): 23
Primitive root (g): 5
Alice's private key (a): 6
Bob's private key (b): 15

--- Public Keys ---
Alice's Public Key (A) = 8
Bob's Public Key   (B) = 19

--- Shared Secret Computation ---
Alice's Shared Secret (K_Alice) = 2
Bob's Shared Secret   (K_Bob)   = 2

Shared secret key successfully established!
Final Shared Secret Key = 2
```

```
=== Diffie-Hellman Key Exchange ===
Enter a large prime number (p): 29
Enter a primitive root modulo p (g): 2
Enter Alice's private key (a): 19
Enter Bob's private key (b): 13

--- Input is Valid ---
Prime number (p): 29
Primitive root (g): 2
Alice's private key (a): 19
Bob's private key (b): 13

--- Public Keys ---
Alice's Public Key (A) = 26
Bob's Public Key   (B) = 14

--- Shared Secret Computation ---
Alice's Shared Secret (K_Alice) = 10
Bob's Shared Secret   (K_Bob)   = 10

Shared secret key successfully established!
Final Shared Secret Key = 10
```

```
=== Diffie-Hellman Key Exchange ===
Enter a large prime number (p): 21
Enter a primitive root modulo p (g): 5
Enter Alice's private key (a): 6
Enter Bob's private key (b): 15
Error: p must be a prime number.
```

```
=== Diffie-Hellman Key Exchange ===
Enter a large prime number (p): 23
Enter a primitive root modulo p (g): 5
Enter Alice's private key (a): 0
Enter Bob's private key (b): 15
Error: Private keys (a, b) must be between 1 and p-1.
```

Observations and Conclusion:

- ***Valid Inputs Produce Matching Shared Keys:***
  - When a prime number p, a valid primitive root g, and private keys a and b within the allowed range are used, both Alice and Bob compute the ***same shared secret key***.
  - Example:
    - `p=23, g=5, a=6, b=15 → Shared key = 2`
    - `p=29, g=2, a=19, b=13 → Shared key = 10`
  - This demonstrates the correctness of the Diffie-Hellman key exchange algorithm.

- ***Invalid Prime Number:***
  - Inputting a non-prime p (e.g., p=21) triggers an ***error***: "Error: p must be a prime number."
  - This ensures that the algorithm only works in a proper modular arithmetic group, preserving security.

- ***Invalid Private Keys:***
  - Inputting a private key outside the range $1 \leq key \leq p-1$ (e.g., a=0) triggers an ***error***: "Error: Private keys (a, b) must be between 1 and p-1."
  - This prevents invalid computation and maintains proper key generation.

- ***Algorithm Behavior:***
  - The program correctly identifies and handles ***wrong inputs***, preventing generation of insecure or meaningless shared keys.
  - Public keys are correctly computed using modular exponentiation.
  - Shared secret computation is consistent for valid inputs.

The Diffie-Hellman Key Exchange algorithm successfully allows two parties to securely generate a shared secret over an insecure channel when inputs are valid.

Input validation is crucial: the prime number p, the primitive root g, and private keys a and b must meet specific criteria to ensure correctness and security.

The program demonstrates that invalid inputs are detected, preventing computation of insecure or incorrect keys.

This implementation provides a secure foundation for further use of the shared key in symmetric encryption and secure communication.