Academic Year: 2024-25	**Programme: BTECH-Cyber (CSE)**
Year: 2nd	**Semester: IV**
Student Name: Mishitha	**Batch : K2**
Roll No: K073	**Date of experiment: 10/2/25**
Faculty: Rejo Mathew	**Signature with Date:**

Experiment 3: Diffie-Hellman Key Exchange

Aim: Write a program to implement Diffie-Hellman Key exchange.

Learning Outcomes:
After completion of this experiment, student should be able to

1. Differentiate between symmetric and asymmetric key cryptography.
2. Describe working of Diffie-Hellman key exchange.
3. Understand application of Diffie-Hellman along with its advantage and limitations.

Theory:

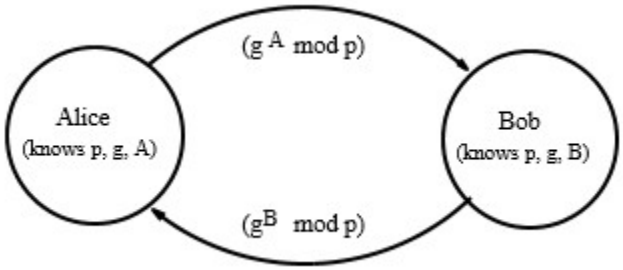
It is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols. It is one of the earliest practical examples of public key exchange implemented within the field of cryptography.

Traditionally, secure encrypted communication between two parties required that they first exchange keys by some secure physical channel, such as paper key lists transported by a trusted courier. This method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

It is used to secure a variety of Internet services. Although Diffie–Hellman key agreement itself is a non-authenticated key-agreement protocol, it provides the basis for a variety of authenticated protocols, and is used to provide forward secrecy in Transport Layer Security's ephemeral modes (referred to as EDH or DHE depending on the cipher suite).

The method was followed shortly afterwards by RSA, an implementation of public-key cryptography using asymmetric algorithms

Algorithm:



1. Alice and Bob agree on a prime number p and a base g
2. Alice chooses a secret number a, and sends Bob (g^a mod p)
3. Bob chooses a secret number b, and sends Bob (g^b mod p)
4. Alice computes ((g^b mod p) ^{a} mod p)
5. Bob computes ((g^a mod p) ^{b} mod p)
6. Alice and Bob can use this number as key

Code: ***type or copy your completed working code here***

Code:

```
# Step 1: Input prime number p
p = int(input("Enter prime number p: "))

# Step 1.1: Check if p is prime using loops
is_prime = True
if p <= 1:
    is_prime = False
elif p == 2:
    is_prime = True
elif p % 2 == 0:
    is_prime = False
else:
    # Check divisors from 3 to sqrt(p)
    i = 3
    while i * i <= p:
        if p % i == 0:
            is_prime = False
            break
        i += 2

if not is_prime:
```

```
    print("Error: p must be a prime number.")
    exit()

# Step 2: Input base g and check if it's primitive root
g = int(input("Enter base g: "))

# Step 2.1: Check primitive root requirements
phi = p - 1  # For prime p, ϕ(p) = p-1

# Find prime factors of ϕ(p) using loops
factors = set()
temp = phi

# Factor out 2s
while temp % 2 == 0:
    factors.add(2)
    temp //= 2

# Factor out odd numbers
i = 3
while i * i <= temp:
    while temp % i == 0:
        factors.add(i)
        temp //= i
    i += 2
if temp > 2:
    factors.add(temp)

# Check primitive root condition using loops
is_primitive = True
for factor in factors:
    # Calculate (g^(ϕ(p)/factor)) mod p
    exponent = phi // factor
    result = 1
    base = g % p
    # Modular exponentiation using loop
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % p
        exponent = exponent // 2
        base = (base * base) % p
    if result == 1:
        is_primitive = False
        break
```

```
if not is_primitive:
    print(f"Error: {g} is not a primitive root of {p}")
    exit()

# Step 3: Input secret numbers
a = int(input("Enter Alice's secret number a: "))
b = int(input("Enter Bob's secret number b: "))

# Step 4: Calculate public values
# Calculate g^a mod p
A = 1
base = g % p
exponent = a
while exponent > 0:
    if exponent % 2 == 1:
        A = (A * base) % p
    exponent = exponent // 2
    base = (base * base) % p

# Calculate g^b mod p
B = 1
base = g % p
exponent = b
while exponent > 0:
    if exponent % 2 == 1:
        B = (B * base) % p
    exponent = exponent // 2
    base = (base * base) % p

print(f"\nAlice sends to Bob: {A}")
print(f"Bob sends to Alice: {B}")

# Step 6 & 7: Calculate shared secret
# Alice computes (B^a mod p)
shared_alice = 1
base = B % p
exponent = a
while exponent > 0:
    if exponent % 2 == 1:
        shared_alice = (shared_alice * base) % p
    exponent = exponent // 2
    base = (base * base) % p
```

```python
# Bob computes (A^b mod p)
shared_bob = 1
base = A % p
exponent = b
while exponent > 0:
    if exponent % 2 == 1:
        shared_bob = (shared_bob * base) % p
    exponent = exponent // 2
    base = (base * base) % p

print(f"\nShared key computed by Alice: {shared_alice}")
print(f"Shared key computed by Bob: {shared_bob}")

# Final verification
if shared_alice == shared_bob:
    print("\nSuccess! Shared keys match.")
else:
    print("\nError! Shared keys do not match.")
```

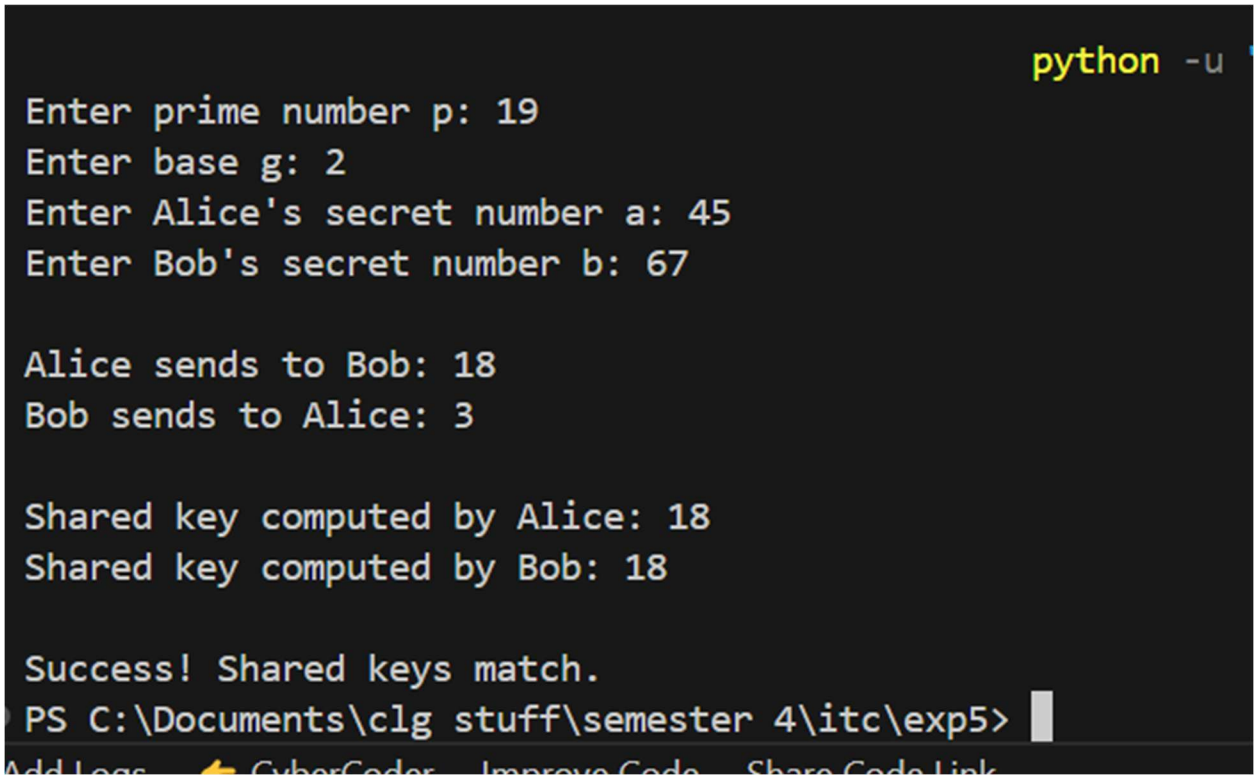
Output:

```
                                                    python -u
Enter prime number p: 43
Enter base g: 3
Enter Alice's secret number a: 27
Enter Bob's secret number b: 39

Alice sends to Bob: 2
Bob sends to Alice: 8

Shared key computed by Alice: 27
Shared key computed by Bob: 27

Success! Shared keys match.
PS C:\Documents\clg stuff\semester 4\itc\exp5>
```


python -u
Enter prime number p: 19
Enter base g: 2
Enter Alice's secret number a: 45
Enter Bob's secret number b: 67
Alice sends to Bob: 18
Bob sends to Alice: 3
Shared key computed by Alice: 18
Shared key computed by Bob: 18
Success! Shared keys match.
PS C:\Documents\clg stuff\semester 4\itc\exp5>

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL

PS C:\Documents\clg stuff\semester 4\itc\exp5> python -
Enter prime number p: 41
Enter base g: 6
Enter Alice's secret number a: 23
Enter Bob's secret number b: 12

Alice sends to Bob: 30
Bob sends to Alice: 4

Shared key computed by Alice: 23
Shared key computed by Bob: 23

Success! Shared keys match.
PS C:\Documents\clg stuff\semester 4\itc\exp5>
```

Note: Code should have proper comments

Questions:

1. Write a short note about Discrete Log problem

Ans: The **Discrete Log Problem (DLP)** is a very hard math problem that is used in cryptography to keep secret keys safe. It involves finding a missing exponent in an equation of the form **y = g^x mod p**, where **g** is a known base, **p** is a large prime number, and **y** is the result. Even if someone knows **g**, **p**, and **y**, finding **x** is extremely difficult. This problem is the foundation of many encryption systems like Diffie-Hellman and RSA, making them secure.

2. What modifications does Elliptic Curve Diffie-Hellman (ECDH) introduce to the original Diffie-Hellman protocol?

Ans: The **Elliptic Curve Diffie-Hellman (ECDH)** protocol is a modified version of the original Diffie-Hellman (DH) key exchange. Traditional DH uses very large numbers to provide security, but ECDH replaces these numbers with **elliptic curve mathematics**, which allows for **smaller key sizes** while maintaining strong security. This makes ECDH more efficient because it is **faster, requires less memory, and provides better**

security with fewer resources. As a result, ECDH is widely used in modern cryptographic systems, including secure messaging apps and internet encryption protocols.

3. Key Exchange algorithms are vulnerable to which type of attacks. How to prevent?

Ans: Key exchange methods, including DH and ECDH, can be vulnerable to **Man-in-the-Middle (MITM) attacks**. In this type of attack, a hacker secretly intercepts messages between two people and pretends to be each of them, allowing the hacker to read and even modify the communication. To prevent MITM attacks, security measures such as **digital signatures, authentication protocols, and certificates (like SSL/TLS)** are used. These methods help verify the identities of the communicating parties, ensuring that no unauthorized person can intercept their messages.

4. Explain the difference between static Diffie-Hellman (DH) and ephemeral Diffie-Hellman (DHE).

Ans: There are two types of Diffie-Hellman key exchange: **Static DH** and **Ephemeral DH (DHE)**. In **Static DH**, the key remains the same for multiple communications. While this can be efficient, it has a major security risk—if an attacker gets access to the key, they can decrypt all past and future messages. On the other hand, **Ephemeral DH (DHE)** generates a new key for every session. This provides **Perfect Forward Secrecy (PFS)**, meaning that even if a hacker steals one session's key, they cannot use it to decrypt old or future messages. Because of this advantage, DHE is commonly used in secure communication protocols, including modern web encryption.

Conclusion: *[Write your own conclusion regarding the lab performed]*

In conclusion, the lab demonstrated the importance of secure key exchange in cryptography. The **Discrete Log Problem** is the foundation of many encryption methods, while **ECDH improves security and efficiency** by using elliptic curves. Key exchange methods can be attacked, but **digital signatures and authentication** help prevent unauthorized access. Finally, **DHE provides better security than Static DH** by ensuring that past communications remain safe, even if a key is compromised.