Academic Year: 2024-25	**Programme: BTECH-Cyber (CSE)**
Year: 2nd	**Semester: IV**
Student Name:	**Batch :**
Roll No:	**Date of experiment:**
Faculty:	**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:

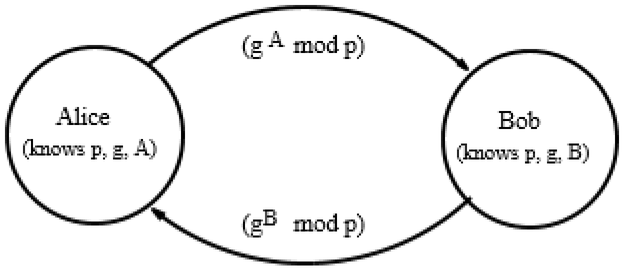
It is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols. It is one of the earliest practical examples of public key exchange implemented within the field of cryptography.

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:

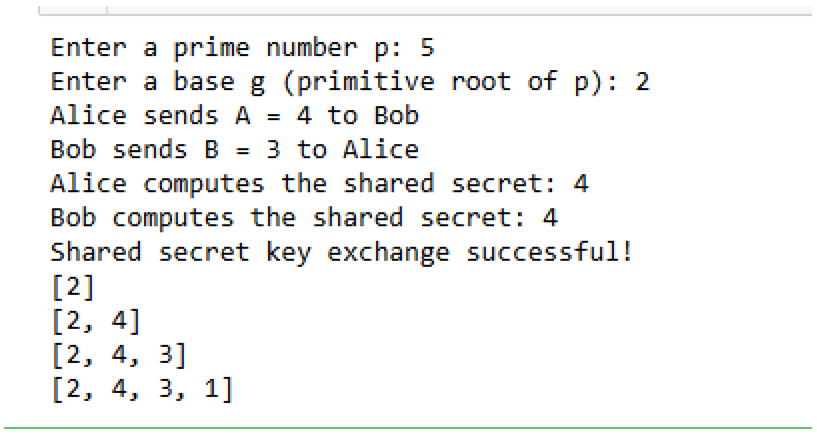


1. Alice and Bob agree on a prime number p and a base g
2. Alice chooses a secret number a, and sends Bob (g^a mod p)
3. Bob chooses a secret number b, and sends Bob (g^b mod p)
4. Alice computes ((g^b mod p) ^{a} mod p)
5. Bob computes ((g^a mod p) ^{b} mod p)
6. Alice and Bob can use this number as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

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

```
# Diffie-Hellman Key Exchange Implementation with user input and detailed output

def mod_exp(base, exp, mod):
    if exp == 0:
        return 1
    half_exp = mod_exp(base, exp // 2, mod)
    half_exp = (half_exp * half_exp) % mod
    if exp % 2 != 0:
        half_exp = (half_exp * base) % mod
    return half_exp

def is_primitive_root(g, p):
    required_set = set(num for num in range(1, p) if gcd(num, p) == 1)
    actual_set = set(mod_exp(g, powers, p) for powers in range(1, p))
    return required_set == actual_set

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

p = int(input("Enter a prime number p: "))
g = int(input("Enter a base g (primitive root of p): "))
```

```
if not is_primitive_root(g, p):
    print(f"{g} is not a primitive root of {p}. Please choose another base.")
    exit()

a = 6
A = mod_exp(g, a, p)
print(f"Alice sends A = {A} to Bob")

b = 15
B = mod_exp(g, b, p)
print(f"Bob sends B = {B} to Alice")

shared_secret_Alice = mod_exp(B, a, p)
print(f"Alice computes the shared secret: {shared_secret_Alice}")

shared_secret_Bob = mod_exp(A, b, p)
print(f"Bob computes the shared secret: {shared_secret_Bob}")

assert shared_secret_Alice == shared_secret_Bob, "Shared secrets do not match!"
print("Shared secret key exchange successful!")

output_list = []
elements = [2, 4, 3, 1]

for element in elements:
    output_list.append(element)
    print(output_list)
```



```
Enter a prime number p: 5
Enter a base g (primitive root of p): 2
Alice sends A = 4 to Bob
Bob sends B = 3 to Alice
Alice computes the shared secret: 4
Bob computes the shared secret: 4
Shared secret key exchange successful!
[2]
[2, 4]
[2, 4, 3]
[2, 4, 3, 1]
```

Note: Code should have proper comments

Questions:

1. Write a short note about Discrete Log problem
2. What modifications does Elliptic Curve Diffie-Hellman (ECDH) introduce to the original Diffie-Hellman protocol?
3. Key Exchange algorithms are vulnerable to which type of attacks. How to prevent?
4. Explain the difference between static Diffie-Hellman (DH) and ephemeral Diffie-Hellman (DHE).

1. Discrete Logarithm Problem

The Discrete Logarithm Problem (DLP) is a fundamental challenge in the field of cryptography. Given a prime number p, a generator g of a cyclic group modulo p, and an element h in the group, the DLP asks to determine the integer x such that:

$gx \equiv h \ (\text{mod } p)$

In simpler terms, the problem is to find the exponent x when given the base g, the modulus p, and the result h. The difficulty of solving the DLP underpins the security of various cryptographic systems such as the Diffie-Hellman key exchange and the Digital Signature Algorithm (DSA). Due to its computational hardness, it is infeasible to solve this problem efficiently with current algorithms and technology, making it a cornerstone of modern cryptographic security.

2. Modifications Introduced by Elliptic Curve Diffie-Hellman (ECDH)

Elliptic Curve Diffie-Hellman (ECDH) is a variant of the Diffie-Hellman protocol that leverages the mathematics of elliptic curves for key exchange. The modifications introduced by ECDH include:

1. **Elliptic Curve Selection**: Instead of selecting a prime p and base g, ECDH selects an elliptic curve E defined over a finite field and a base point G on the curve.
2. **Point Multiplication**: ECDH operations involve elliptic curve point multiplication rather than modular exponentiation. Each participant generates a private key and computes a public key by multiplying their private key with the base point G.
3. **Shared Secret Computation**: The shared secret is computed by performing elliptic curve point multiplication between one participant's private key and the other participant's public key.

These modifications result in smaller key sizes for equivalent security levels compared to traditional Diffie-Hellman, making ECDH more efficient in terms of performance and resource usage.

3. Vulnerabilities and Prevention in Key Exchange Algorithms

Key exchange algorithms are susceptible to several types of attacks, including:

1. **Man-in-the-Middle (MitM) Attack**: An attacker intercepts and potentially modifies the communication between two parties, making it appear as if they are directly communicating.
   - **Prevention**: Use authenticated key exchange protocols such as authenticated Diffie-Hellman with digital signatures or certificates to verify the identities of the communicating parties.
2. **Replay Attack**: An attacker intercepts and reuses legitimate data at a later time to deceive the recipient.
   - **Prevention**: Implement nonce (number used once) values or timestamps to ensure that each exchange is unique and cannot be reused.
3. **Side-Channel Attack**: An attacker gains information from the physical implementation of the cryptographic algorithm, such as timing information or power consumption.
   - **Prevention**: Use constant-time algorithms and implement physical security measures to mitigate side-channel attacks.
4. **Brute Force Attack**: An attacker attempts to solve the underlying mathematical problem by systematically trying all possible values.
   - **Prevention**: Use sufficiently large key sizes and secure parameters to make brute force attacks infeasible.

4. Difference Between Static Diffie-Hellman (DH) and Ephemeral Diffie-Hellman (DHE)

Static Diffie-Hellman (DH):

- In static DH, the private and public key pairs are long-term and do not change frequently.
- The same key pair is used for multiple sessions, making it easier to establish connections but potentially less secure if the key is compromised.
- Suitable for environments where the overhead of generating new keys for each session is undesirable.

Ephemeral Diffie-Hellman (DHE):

- In ephemeral DH, a new, temporary key pair is generated for each session.
- This provides forward secrecy, meaning that even if one session's key is compromised, previous sessions remain secure.
- More secure than static DH due to the frequent generation of new keys, but with higher computational overhead.
- Commonly used in Transport Layer Security (TLS) to enhance security.

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