BARI ANKIT (56)

Exp 3 : Implementation of Diffie Hellman Key exchange algorithm

Diffie-Hellman Key Exchange Algorithm

The Diffie-Hellman key exchange algorithm allows two parties to securely exchange cryptographic keys over an insecure communication channel. It is based on the mathematical concept of modular exponentiation and the Discrete Logarithm Problem (DLP), which is computationally hard to solve.

Steps of Diffie-Hellman Key Exchange:

1. **Public Parameters:**
    - **Both parties agree on two public parameters:**
        - **A large prime number ppp**
        - **A primitive root modulo ppp, denoted by ggg.**
2. **Private Keys:**
    - **Each party selects a private key:**
        - **Party A selects a private key aaa.**
        - **Party B selects a private key bbb.**
3. **Public Keys:**
    - **Each party computes its public key:**
        - **Party A computes A=gamod pA = g^a \mod pA=gamodp.**
        - **Party B computes B=gbmod pB = g^b \mod pB=gbmodp.**
4. **Key Exchange:**
    - **The two parties exchange their public keys:**
        - **Party A sends AAA to Party B.**
        - **Party B sends BBB to Party A.**
5. **Shared Secret:**
    - **Once the public keys are exchanged, both parties can compute the shared secret key:**
        - **Party A computes KA=Bamod pK_A = B^a \mod pKA=Bamodp.**
        - **Party B computes KB=Abmod pK_B = A^b \mod pKB=Abmodp.**
    - **Due to the properties of modular arithmetic, both computations yield the same shared secret key: KA=KB=gabmod pK_A = K_B = g^{ab} \mod pKA=KB=gabmodp**

Example of Diffie-Hellman Key Exchange:

- **Suppose Party A and Party B want to agree on a secret key without**

directly sharing it.

- **They agree on public parameters p=23p = 23p=23 and g=5g = 5g=5.**
- **Party A selects a private key a=6a = 6a=6, and Party B selects a private key b=15b = 15b=15.**

Now, let's implement this algorithm in Python.

Code :

```
# Diffie-Hellman Key Exchange Implementation


def power_mod(base, exponent, modulus):
    """Efficiently computes base^exponent % modulus using binary exponentiation"""
    result = 1
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent //= 2
    return result

# Step 1: Choose public parameters (p, g)
p = 23  # prime number
g = 5   # primitive root modulo p

# Step 2: Party A selects private key 'a' and Party B selects private key 'b'
a = 6  # Private key of Party A
b = 15 # Private key of Party B

# Step 3: Calculate public keys
A = power_mod(g, a, p)  # A = g^a % p
B = power_mod(g, b, p)  # B = g^b % p

# Step 4: Exchange public keys (A and B)
print(f"Party A's public key: {A}")
print(f"Party B's public key: {B}")
```

```
# Step 5: Both parties compute the shared secret key
K_A = power_mod(B, a, p)  # A computes shared secret: K_A = B^a % p
K_B = power_mod(A, b, p)  # B computes shared secret: K_B = A^b % p

print(f"Party A's shared secret key: {K_A}")
print(f"Party B's shared secret key: {K_B}")
```

Output :

```
# Both keys should be the same
assert K_A == K_B, "The shared secret keys do not match!"
print("Shared secret keys match!")
```