Faculty: Engineering and Technology

School of Computer Engineering & Technology
Programme: B.Tech Computer Sc. & Engineering

Name: Devanshu Surana
Roll No.: 23
Prn:1032210755
Panel: C batch:C1

Lab A3: Implement Diffie-Hellman Key Exchange Algorithm using python or java or C++

Objective of Lab

1. To introduce the fundamental concepts of discrete logarithmic problem, public key encryption , man in the middle attack and implement Diffie-Hellman Key Exchange Algorithm

Theory:

Diffie-Hellman algorithm:
The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b. P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt

Example of Using Diffie-Hellman Key Exchange Algorithm

Step 1: Alice and Bob get public numbers P = 23, G = 9

Step 2: Alice selected a private key a = 4 and
Bob selected a private key b = 3

Step 3: Alice and Bob compute public values
Alice: x =(9^4 mod 23) = (6561 mod 23) = 6
Bob: y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and
Bob receives public key x = 6

Step 6: Alice and Bob compute symmetric keys
Alice: ka = y^a mod p = 65536 mod 23 = 9
Bob: kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.

Algorithm

1. Initialization:

   Both parties, Alice and Bob, agree on two publicly known values: a prime number (p) and a primitive root modulo p (g).
   These values are typically agreed upon in advance and are available to anyone who wants to communicate securely.
2. Private Key Generation:

   Both Alice and Bob independently choose their private keys:
   Alice selects a secret integer "a."
   Bob selects a secret integer "b."
   These private keys are kept confidential and are not shared with anyone.
3. Public Key Calculation:

   Alice computes her public key (A) as follows:
   A = g^a mod p
   Bob computes his public key (B) as follows:
   B = g^b mod p
   Note that Alice and Bob perform these calculations independently and without sharing their private keys.
4. Public Key Exchange:

   Alice sends her public key (A) to Bob.
   Bob sends his public key (B) to Alice.
   These public keys can be openly exchanged over the untrusted communication channel.
5. Shared Secret Key Derivation:

   Alice uses Bob's public key (B) and her private key (a) to derive the shared secret key (s) as follows:
   s = B^a mod p
   Bob uses Alice's public key (A) and his private key (b) to derive the shared secret key (s) as follows:
   s = A^b mod p
   These calculations are performed independently by Alice and Bob.

6. Shared Secret Key:

   Both Alice and Bob now have the same shared secret key (s).
   This shared secret key can be used for secure communication between the two parties.

Code

```
def prime_checker(p):
    # Checks If the number entered is a Prime Number or not
    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):
    # Checks If The Entered Number Is A Primitive Root Or Not
    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 1:
    P = int(input("Enter P : "))
    if prime_checker(P) == -1:
        print("Number Is Not Prime, Please Enter Again!")
        continue
    break

while 1:
    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 1:
    if x1 >= P or x2 >= P:
        print(f"Private Key Of Both The Users Should Be Less Than {P}!")
        continue
    break

# Calculate Public Keys
```

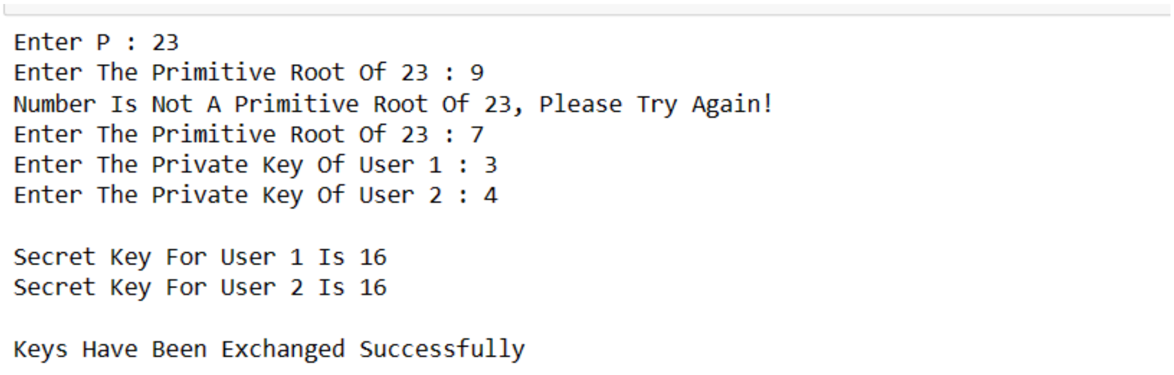
```
y1, y2 = pow(G, x1) % P, pow(G, x2) % P

# Generate Secret Keys
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 Screen shots



```
Enter P : 23
Enter The Primitive Root Of 23 : 9
Number Is Not A Primitive Root Of 23, Please Try Again!
Enter The Primitive Root Of 23 : 7
Enter The Private Key Of User 1 : 3
Enter The Private Key Of User 2 : 4

Secret Key For User 1 Is 16
Secret Key For User 2 Is 16

Keys Have Been Exchanged Successfully
```

Conclusion: Thus, we have learned and implemented the Diffie-Hellman Key Exchange Algorithm successfully.

FAQs:

1. What is Diffie Hellman Key Exchange
2. What is Diffie Hellman most commonly used for?
3. Is Diffie Hellman Symmetric algorithm
4. Is Diffie Hellman secured and still used today