**Final Year B. Tech., Sem VII 2022-23**

**Cryptography And Network Security**

**PRN/ Roll No: 2020BTECS00206**

**Full Name: SAYALI YOGESH DESAI**

**Batch: B4**

**Assignment No. 11**

1. **Aim:**

Implementation of Diffie Hellman Key Exchange Algorithm

1. **Theory:**

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:**

**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.

1. **Code:**

#include <bits/stdc++.h>

using namespace std;

long long powM(long long a, long long b, long long n)

{

if (b == 1)

return a % n;

long long x = powM(a, b / 2, n);

x = (x \* x) % n;

if (b % 2)

x = (x \* a) % n;

return x;

}

bool checkPrimitiveRoot(long long alpha, long long q)

{

map<long long, int> m;

for (long long i = 1; i < q; i++)

{

long long x = powM(alpha, i, q);

//cout << x << endl;

if (m.find(x) != m.end())

return 0;

m[x] = 1;

}

return 1;

}

int main()

{

long long q, alpha;

q = 7; // A prime number q is taken

alpha = 5; // A primitive root of q

if (checkPrimitiveRoot(alpha, q) == 0)

{

cout << "alpha is not primitive root of q";

return 0;

}

else

{

cout << alpha << " is private root of " << q << endl;

}

long long xa, ya;

xa = 3; // xa is the chosen private key

ya = powM(alpha, xa, q); // public key of alice

cout << "\n Private key of alice is " << xa << endl;

cout << "\n Public key of alice is " << ya << endl << endl;

long long xb, yb;

xb = 4; // xb is the chosen private key

yb = powM(alpha, xb, q); // public key of bob

cout << "\n Private key of bob is " << xb << endl;

cout << "\n Public key of bob is " << yb << endl << endl;

//key generation

long long k1, k2;

k1 = powM(yb, xa, q); // Secret key for Alice

k2 = powM(ya, xb, q); // Secret key for Bob

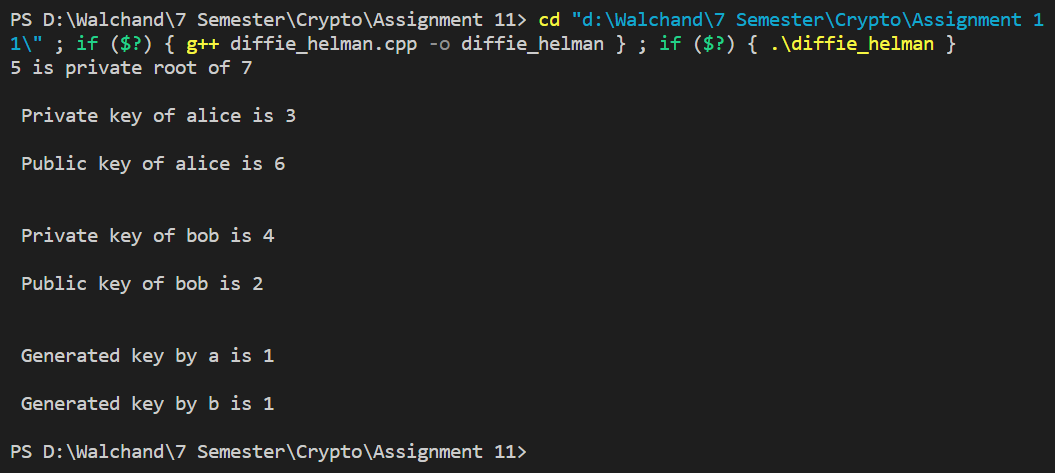
cout << "\n Generated key by a is " << k1 << endl;

cout << "\n Generated key by b is " << k2 << endl << endl;

return 0;

}

1. **Output:**

****

1. **Conclusion:**

Successfully implemented Diffie Hellman Key Exchange Algorithm.