Program – 7

AIM: To implement a program to show key exchange using Diffie-Hallman key exchange.

Introduction and Theory

Diffie-Hellman key exchange, also called exponential key exchange, is a method of digital encryption that uses numbers raised to specific powers to produce decryption keys on the basis of components that are never directly transmitted, making the task of a would-be code breaker mathematically overwhelming.
To implement Diffie-Hellman, the two end users Alice and Bob, while communicating over a channel they know to be private, mutually agree on positive whole numbers p and q, such that p is a prime number and q is a generator of p. The generator q is a number that, when raised to positive whole-number powers less than p, never produces the same result for any two such whole numbers. The value of p may be large but the value of q is usually small.
Once Alice and Bob have agreed on p and q in private, they choose positive whole-number personal keys a and b, both less than the prime-number modulus p. Neither user divulges their personal key to anyone; ideally they memorize these numbers and do not write them down or store them anywhere. Next, Alice and Bob compute public keys a* and b* based on their personal keys according to the formulas

$$a^* = q^a mod\ p$$

And

$$b^* = q^b mod\ p$$

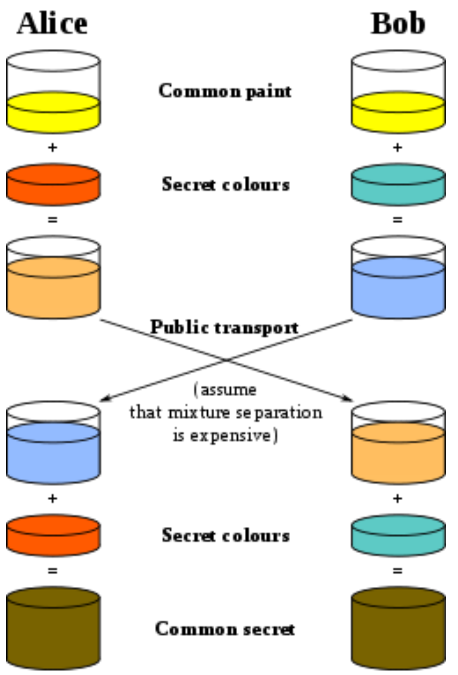
The two users can share their public keys a* and b* over a communications medium assumed to be insecure, such as the Internet or a corporate wide area network (WAN). From these public keys, a number x can be generated by either user on the basis of their own personal keys. Alice computes x using the formula

$$x = (b^*)^a\ mod\ p$$

Bob computes x using the formula

$$x = (a^*)^b\ mod\ p$$

The value of x turns out to be the same according to either of the above two formulas. However, the personal keys a and b, which are critical in the calculation of x, have not been transmitted over a public medium. Because it is a large and apparently random number, a potential hacker has almost no chance of correctly guessing x, even with the help of a powerful computer to conduct millions of trials. The two users can therefore, in theory, communicate privately over a public medium with an encryption method of their choice using the decryption key x.



Program – 7

Code

```
#include<iostream>
#include<string>
#include<vector>
#include<cstdlib>
#include<cmath>

using namespace std;

long long int power(long long int a, long long int b,
                                    long long int P)
{
    if (b == 1)
        return a;

    else
        return (((long long int)pow(a, b)) % P);
}

class Node
{
    private:
        long long int g;
        long long int modulus;
        long long int power_element;
        long long int key_1;
        long long int key;
    public:
        Node(int p, int G)
        {
            g = G;
            modulus = p;
        }

        void select_a();
        long long int generate_key1();
        void generate_key(long long int);
        bool check_key(long long int);
        void print_power()
        {
            cout << "selected power: " << power_element << endl;
        }

        void print_key()
        {
            cout << "selected key: " << key << endl;
        }

        long long int get_key() { return key; }
};

void Node::select_a()
{
    power_element = rand() % 10; // a = [0, 100)
}

```

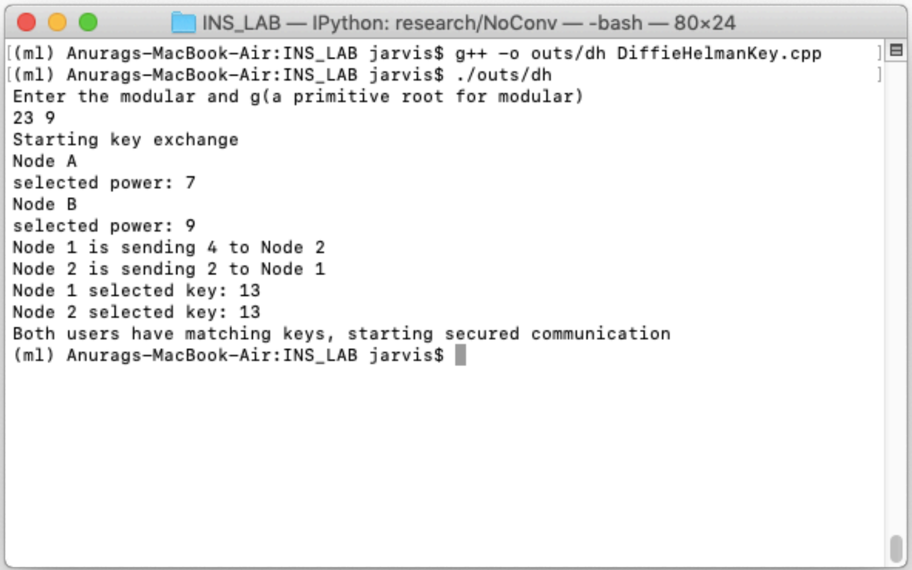
Program – 7

```cpp
long long int Node::generate_key1()
{
    // cout << g << "^" << power_element << "\%" << modulus << endl;
    long long int A = power(g, power_element, modulus);
    // cout << A << endl;
    key_1 = A;
    return A;
}

void Node::generate_key(long long int A)
{
    long long int s = power(A, power_element, modulus);
    key = s;
}

bool Node::check_key(long long int O)
{
    if (O == key)
        return true;
    return false;
}

int main()
{
    int p, g;
    cout << "Enter the modular and g(a primitive root for modular)" << endl;
    cin >> p >> g;
    Node A(p, g);
    Node B(p, g);

    cout << "Starting key exchange" << endl;
    cout << "Node A" << endl;
    A.select_a();
    A.print_power();
    cout << "Node B" << endl;
    B.select_a();
    B.print_power();

    long long int channel1 = A.generate_key1();
    long long int channel2 = B.generate_key1();
    cout << "Node 1 is sending " << channel1 << " to Node 2" << endl;
    cout << "Node 2 is sending " << channel2 << " to Node 1" << endl;
    A.generate_key(channel2);
    B.generate_key(channel1);
    cout << "Node 1 ";
    A.print_key();
    cout << "Node 2 ";
    B.print_key();

    if (A.check_key(B.get_key()) && B.check_key(A.get_key()))
    {
        cout << "Both users have matching keys, starting secured communication" << endl;
    }
```

Program – 7

```
    else
    {
        cout << "Key generation failed, exiting" << endl;
    }
    return 0;
}
```

Results and Outputs:



Findings and Learnings:

1. We have implemented Diffie Hellman key exchange.