Name: Ritesh Pawar

Batch: B4

Subject: CNS Lab

PRN: 2020BTECS00068

Aim: Diffi-helman key exchange Algorithm

Theory:

Diffie-Hellman algorithm is one of the most important algorithms used for establishing a shared secret. At the time of exchanging data over a public network, we can use the shared secret for secret communication. We use an elliptic curve for generating points and getting a secret key using the parameters.

1. We will take four variables, i.e., P (prime), G (the primitive root of P), and a and b(private values).
2. The variables P and G both are publicly available. The sender selects a privatevalue, either a or b, for generating a key to exchange publicly. The receiver receives the key, and that generates a secret key, after which the sender and receiver both have the same secret key to encrypt.

Code:

Alice.cpp

alice.cpp

diffiehellman > alice.cpp

```cpp
#include <iostream>
#include <cmath>
#include <winsock2.h>

long long p = 17; // Large prime number (public)
long long alpha = 5;  // Primitive root modulo p (public)

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;
}

int main() {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "Failed to initialize Winsock" << std::endl;
        return -1;
    }

    SOCKET clientSocket;
    struct sockaddr_in serverAddress;

    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == INVALID_SOCKET) {
        std::cerr << "Error creating socket" << std::endl;
        WSACleanup();
        return -1;
    }
```

alice.cpp

diffiehellman > alice.cpp

```cpp

    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(8080);
    serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1"); //Localhost

    if (connect(clientSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress)) == SOCKET_ERROR) {
        std::cerr << "Error connecting to Bob" << std::endl;
        closesocket(clientSocket);
        WSACleanup();
        return -1;
    }

    int xa = 4; // Alice's private key

    // Alice computes A = (alpha^xa) % p
    int A = powM(alpha, xa, p);
    std::cout << "Alice computes A: " << A << std::endl;

    // Send Alice's public value A to Bob
    send(clientSocket, (char*)&A, sizeof(A), 0);
    std::cout << "Sent Alice's public value A to Bob" << std::endl;

    // Receive Bob's public value B
    int B;
    recv(clientSocket, (char*)&B, sizeof(B), 0);
    std::cout << "Received Bob's public value B: " << B << std::endl;

    // Calculate the shared secret key
    int shared_key_alice = powM(B, xa, p);
    std::cout << "Shared key calculated by Alice: " << shared_key_alice << std::endl;

    closesocket(clientSocket);
    WSACleanup();

    return 0;
}

```

Bob.cpp

bob.cpp

diffiehellman > bob.cpp

```cpp
#include <iostream>
#include <cmath>
#include <winsock2.h>

long long p = 17; // Large prime number (public)
long long alpha = 5;  // Primitive root modulo p (public)

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;
}

int main() {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "Failed to initialize Winsock" << std::endl;
        return -1;
    }

    SOCKET serverSocket;
    struct sockaddr_in serverAddress;
    SOCKET clientSocket;
    struct sockaddr_in clientAddress;

    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == INVALID_SOCKET) {
        std::cerr << "Error creating socket" << std::endl;
        WSACleanup();
        return -1;
    }
```

bob.cpp

diffiehellman > bob.cpp

```cpp
    std::cout << "Bob is waiting for Alice to connect..." << std::endl;

    int clientAddress_size = sizeof(clientAddress);
    clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddress, &clientAddress_size);

    if (clientSocket == INVALID_SOCKET) {
        std::cerr << "Error accepting the connection" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return -1;
    }

    int xb = 6; // Bob's private key

    // Receive Alice's public value A
    int A;
    recv(clientSocket, (char*)&A, sizeof(A), 0);
    std::cout << "Received Alice's public value A: " << A << std::endl;

    // Bob computes B = (alpha^xb) % p
    int B = powM(alpha, xb, p);
    std::cout << "Bob computes B: " << B << std::endl;

    // Send Bob's public value B to Alice
    send(clientSocket, (char*)&B, sizeof(B), 0);
    std::cout << "Sent Bob's public value B to Alice" << std::endl;

    // Calculate the shared secret key
    int shared_key_bob = powM(A, xb, p);
    std::cout << "Shared key calculated by Bob: " << shared_key_bob << std::endl;

    closesocket(serverSocket);
    closesocket(clientSocket);
    WSACleanup();

    return 0;
}
```

Output:

```
PS E:\CNS\diffiehellman> g++ bob.cpp -o bob -lws2_32
PS E:\CNS\diffiehellman> .\bob.exe
Bob is waiting for Alice to connect...
Received Alice's public value A: 13
Bob computes B: 2
Sent Bob's public value B to Alice
Shared key calculated by Bob: 16
PS E:\CNS\diffiehellman>
```

```
PS E:\CNS\diffiehellman> .\alice.exe
Alice computes A: 13
Sent Alice's public value A to Bob
Received Bob's public value B: 2
Shared key calculated by Alice: 16
PS E:\CNS\diffiehellman>
```