**Cryptography and Network Security Lab**

**Practical 09**

**Title:** Diffie-Hellman Key Exchange Implementation.

**Aim**: To Study and Implement Diffie-Hellman Key Exchange using Socket Programming.

**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.

● We will take four variables, i.e., P (prime), G (the primitive root of P), and a and b (private values).

● The variables P and G both are publicly available. The sender selects a private value, 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:**

#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 isPrimitiveRoot(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*);

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

return 0;

}

m[x] = 1;

}

return 1;

}

int main(){

long long q, alpha;

q = 17; *// A prime number q is taken*

alpha = 5; *// A primitive root of q*

if (isPrimitiveRoot(alpha, q) == 0){

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

return 0;

}else {

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

}

long long xa, ya;

xa = 4; *// xa is the chosen private key*

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

cout << "private key of A is " << xa << endl;

cout << "public key of A is " << ya << endl << endl;

long long xb, yb;

xb = 6; *// xb is the chosen private key*

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

cout << "private key of B is " << xb << endl;

cout << "public key of B 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*

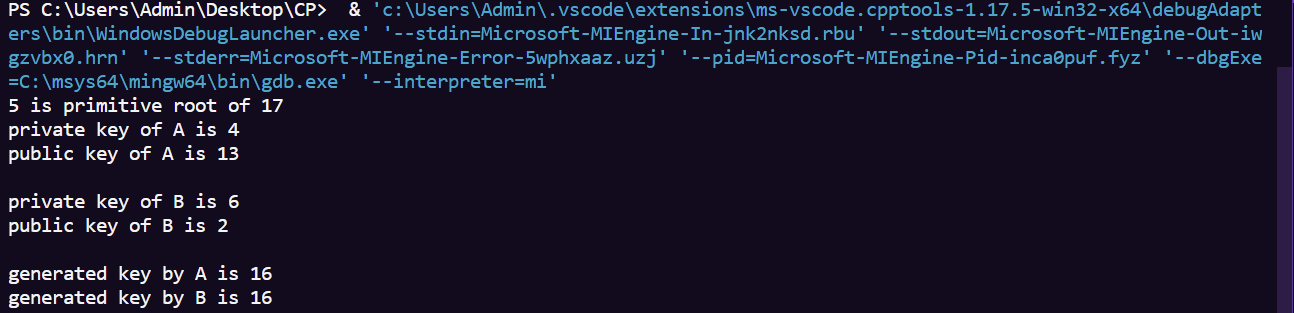
cout << "generated key by A is " << k1 << endl;

cout << "generated key by B is " << k2 << endl;

return 0;

}

**Results:**



**Applications**:

● Public Key Infrastructure: The public-key infrastructure (PKI) is a set of tools and rules to enforce public key cryptography with multiple entities. It also governs the issuance of digital certificates over the internet to maintain data confidentiality. With the Diffie-Hellman algorithm as the base, the PKI system was created to enable the exchange of public keys with anyone who requests for it and has the appropriate permissions.

● SSL/TLS Handshake: Internet browsers are authenticated with website servers using an SSL/TLS certificate and many keys. This is possible only because of the key exchange algorithm, which enables the secure exchange of cryptographic entities over all channels.

● Secure Shell Access (SSH): SSH is a cryptographic protocol used to access system terminals from a third-party appliance or application. The Diffie-Hellman algorithm assists in exchanging the keys between both systems before enabling remote access.

**Diffie-Hellman Implementation using Socket Programming:**

1. To implement Diffie-Hellman using socket programming, we need to create two programs. Alice as client and Bob as Server.

2. Bob (Server) waits for Alice (client) to connect. They exchange public values (A and B) and calculate the shared secret key.

3. Firstly, compile both programs, run bob.exe first and then alice.exe

Alice.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;

    }

    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

#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;

    }

    serverAddress.sin\_family = AF\_INET;

    serverAddress.sin\_port = htons(8080);

    serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

    if (bind(serverSocket, (struct sockaddr\*)&serverAddress, sizeof(serverAddress)) == SOCKET\_ERROR) {

        std::cerr << "Error binding to port" << std::endl;

        closesocket(serverSocket);

        WSACleanup();

        return -1;

    }

    listen(serverSocket, 1);

    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);

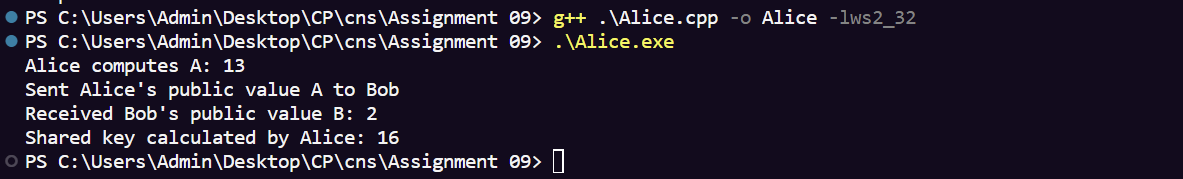
    WSACleanup();

    return 0;

}

Output:

Alice:



Bob:

