Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 11

Title: Implementation of Diffie-Hellman Key Exchange.

Theory:

The Diffie-Hellman key exchange was one of the most important developments in public-key cryptography and it is still frequently implemented in a range of today's different security protocols.

It allows two parties who have not previously met to securely establish a key which they can use to secure their communications. In this article, we'll explain what it's used for, how it works on a step-by-step basis, its different variations, as well as the security considerations that need to be noted in order to implement it safely.

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.

Step by Step Explanation

Alice	Bob
Public Keys available = P, G	Public Keys available = P, G
Private Key Selected = a	Private Key Selected = b
Key generated = $x = G^a mod P$	Key generated = $y = G^b mod P$
Exchange of generated keys takes place	
Key received = y	key received = x
Generated Secret Key = $k_a = y^a mod P$	Generated Secret Key = $k_b = x^b mod P$
Algebraically, it can be shown that $k_a = k_b$	
Users now have a symmetric 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.
```

Code Snapshots:

```cpp
#include <bits/stdc++.h>
#define ll long long
#define ul unsigned long long
#define pb emplace_back
#define po pop_back
#define vi vector<ll>
#define vii vector<vector<ll>>
using namespace std;

vector<int> primeNums;
vector<bool> prime(100000001,1);

void SeiveOfEratosthenes(int n){
    for(int p=2; p*p<=n; p++){
        if(prime[p] ==  true){
            for (int i = p * p; i <= n; i += p)
            prime[i] = false;
        }
    }
```

```
    for(int i=3;i<n;i+=2){
        if(prime[i]) primeNums.push_back(i);
    }
}


ll power(ll a, ll b, ll p){
    if (b == 1)
        return a;
    else
        return (((long long int)pow(a, b)) % p);
}


void findPrimefactors(unordered_set<int> &s, int n){
    while (n%2 == 0){
        s.insert(2);
        n = n/2;
    }
    for (int i = 3; i <= sqrt(n); i = i+2){
        while (n%i == 0){
            s.insert(i);
            n = n/i;
        }
    }
    if (n > 2)
        s.insert(n);
}


int primitiveRoot(int n){
    unordered_set<int> s;
    int phi = n-1;
    findPrimefactors(s, phi);
    for (int r=2; r<=phi; r++){
        bool flag = false;
        for (auto it = s.begin(); it != s.end(); it++){
            if (power((ll)r, (ll)phi/(*it),(ll)n) == 1)
            {
                flag = true;
                break;
            }
         }
         if (flag == false)
           return r;
    }
    return -1;
}
```

```cpp
int main(){
    // prime number till 100000000
    SeiveOfEratosthenes(100000000);
    int privateNumberA, privateNumberB;
    cout<<"Enter the privateNumber of A and B respectively : ";
    cin>>privateNumberA>>privateNumberB;
    cout<<"\nFinding prime Number and a primitive root ...\n";

    srand(time(0));
    int  p = primeNums[rand() % primeNums.size()];
    int g = primitiveRoot(p);

    cout<<"\tPrime Number : "<<p<<"\n";
    cout<<"\tPrimitive Root :"<<g<<"\n";

    // calculating the private key for  a
    ll x = power(g,privateNumberA,p);
    if(x<0) x = p + x;
    cout<<"\nThe private key a for A is : "<<x<<"\n";

    // calculate private key for b
    ll y = power(g,privateNumberB,p);
    if(y<0) y = p + y;
    cout<<"The private key b for B is : "<<y<<"\n";

    ll ka = power(y, privateNumberA, p); // Secret key for A
    if(ka<0) ka = p + ka;
    ll  kb = power(x, privateNumberB, p); // Secret key for B
    if(kb<0) kb = p + kb;

    cout<<"\n\nSecret key for the A is :"<<ka;
    cout<<"\nSecret key for the B is : "<<kb<<endl;
    return 0;
}
```

Output Snapshots:

```
PROBLEMS    OUTPUT    TERMINAL    GITLENS    DEBUG CONSOLE

PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive\De
sktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ DiffieHelman.cpp -o DiffieHelman } ; if ($?) {
 .\DiffieHelman }
        Prime Number : 12589
        Primitive Root :2

The private key a for A is : 4334
The private key b for B is : 512


Secret key for the A is :10937
Secret key for the B is : 10937
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> 
```

Conclusion:

1. The Diffie Hellman key Exchange has proved to be a useful key exchange system due to its advantages.
2. While it is really tough for someone snooping the network to decrypt the data and get the keys, it is still possible if the numbers generated are not entirely random.