**EX.NO – 07 DEFFIE HELLMAN EXCHANGE ALGORITHM T.HARSHINI**

**22/9/2020 2017103532**

**AIM :**

Implementation of Diffie hellman’s exchange algorithm for symmetric key cryptography.

**UNDERSTANDING OF ALGORITHM-**

**Elliptic Curve Cryptography (ECC)** is an approach to public-key cryptography, based on the algebraic structure of elliptic curves over finite fields. ECC requires a smaller key as compared to non-ECC cryptography to provide equivalent security (a 256-bit ECC security has an equivalent security attained by 3072-bit RSA cryptography).

* 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 received the key and from that generates a secret key after which they have the same secret key to encrypt.



The **Diffie**–Hellman **key** exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared **secret key** over an insecure channel. This **key can** then be used to encrypt subsequent communications using a symmetric **key** cipher.

**Symmetric**-key algorithms are algorithms for **cryptography** that use the same cryptographic keys for both **encryption** of plaintext and decryption of ciphertext.

**Symmetric key chipher – for encryption and decryption**

It is also referred to as ‘Shift Cipher’ or ‘Caesar Cipher’. As the name suggests, ‘addition modulus 2’ operation is performed on the plain-text to obtain a cipher-text.

C = (M + k) mod n  
M = (C – k) mod n

where,  
C -> cipher-text  
M -> message/plain-text  
k -> key

**CODE-**

#include<stdio.h>

#include<math.h>

void decrypt(int key)

{

char message[100], ch;

int i;

for(i = 0; message[i] != '\0'; ++i){

ch = message[i];

if(ch >= 'a' && ch <= 'z'){

ch = ch - key;

if(ch < 'a'){

ch = ch + 'z' - 'a' + 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z'){

ch = ch - key;

if(ch < 'A'){

ch = ch + 'Z' - 'A' + 1;

}

message[i] = ch;

}

}

printf("Decrypted message: %s", message);

printf("\n");

}

void encrypt(int key)

{

char message[100], ch;

int i;

printf("Enter a message to encrypt: ");

gets(message);

for(i = 0; message[i] != '\0'; ++i){

ch = message[i];

if(ch >= 'a' && ch <= 'z'){

ch = ch + key;

if(ch > 'z'){

ch = ch - 'z' + 'a' - 1;

}

message[i] = ch;

}

else if(ch >= 'A' && ch <= 'Z'){

ch = ch + key;

if(ch > 'Z'){

ch = ch - 'Z' + 'A' - 1;

}

message[i] = ch;

}

}

printf("Encrypted message: %s", message);

printf("\n");

return 0;

}

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);

}

int main()

{

long long int P, G, x, a, y, b, ka, kb;

P = 23;

printf("The value of P : %lld\n", P);

G = 9;

printf("The value of G : %lld\n\n", G);

a = 4;

printf("The private key a : %lld\n", a);

x = power(G, a, P);

b = 3;

printf("The private key b : %lld\n\n", b);

y = power(G, b, P);

ka = power(y, a, P);

kb = power(x, b, P);

printf("Secret key for Sender is : %lld\n", ka);

printf("Secret Key for reciever is : %lld\n", kb);

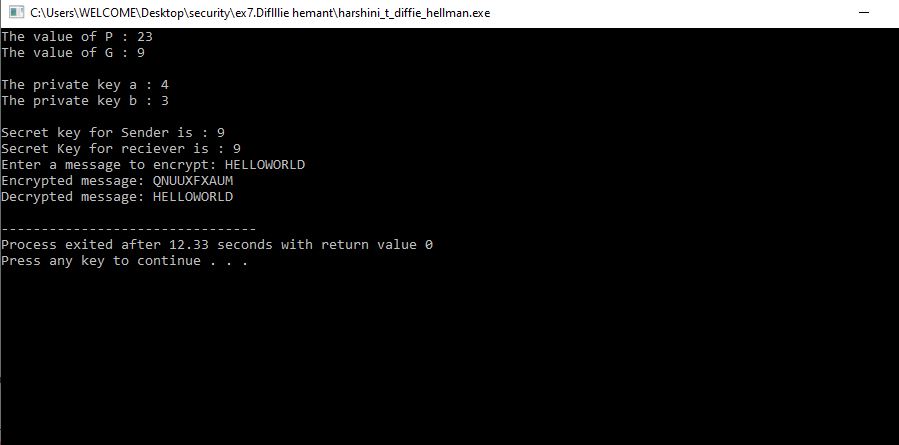
encrypt(kb);

decrypt(kb);

return 0;

}

**OUTPUT –**

****

**HARSHINI T(2017103532)**