**IMPLEMENTATION OF RSA, DIFFIE HELLMAN AND MAN-IN-THE-MIDDLE**

**ATTACK**

**AIM**: To implement RSA public-key cryptosystem, Diffie Hellman Key technique and the Man-in-the-middle attack.

**CODE**:

RSA (Rivest, Shamir, and Adleman) Cryptosystem

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <math.h>

#include <set>

using namespace std;

bool isPrime(int num) {

    if (num <= 1)

        return false;

    for (int i = 2; i <= sqrt(num); i++) {

        if (num % i == 0) {

            return false;

        }

    }

    return true;

}

int generatePrime() {

    int num;

    while (true) {

        num=rand()%415+1;

        if (isPrime(num)) {

            return num;

        }

    }

}

int gcd(int a, int b) {

    if (b == 0)

        return a;

    return gcd(b, a % b);

}

int coprime(int n){

    int flag=0;

    for(int i=3;i<n;i++){

        if(gcd(n,i)==1){

            return i;

            break;

        }

    }

    return 0;

}

int privateKey(int copr, int tot){

    //e x d === 1 mod tot value

    for(int i=2;i<=1000000;i++){

        int val=(copr\*i)%tot;

        if(val==1){

            return i;

            break;

        }

        else continue;

    }

    return 0;

}

int modExp(int base, int exponent, int modulus) {

    int result = 1;

    base = base % modulus;

    while (exponent > 0) {

        if (exponent % 2 == 1) {

            result = (result \* base) % modulus;

        }

        exponent = exponent >> 1;

        base = (base \* base) % modulus;

    }

    return result;

}

int main(){

    int p=generatePrime();

    p=23;

    int q=generatePrime();

    q=29;

    cout << "Here is the value for p: " << p << endl;

    cout << "Here is the value for q: " << q << endl;

    cout << "------------------------" << endl;

    int n=p\*q;

    cout << "Here is the value for n: " << n << endl;

    cout << "------------------------" << endl;

    int tot=(p-1)\*(q-1);

    cout << "Here is the totient value for n: " << tot << endl;

    cout << "------------------------" << endl;

    int copr=coprime(n);

    cout << "Here is the coprime value for n: " << copr << endl;

    cout << "------------------------" << endl;

    int privKey=privateKey(copr,tot);

    cout << "Here is the private key value for n: " << privKey << endl;

    cout << "------------------------" << endl;

    cout << "Now enter the message to be encrypted: "; int m;

    cin >> m;

    int encryption=modExp(m,copr,n);

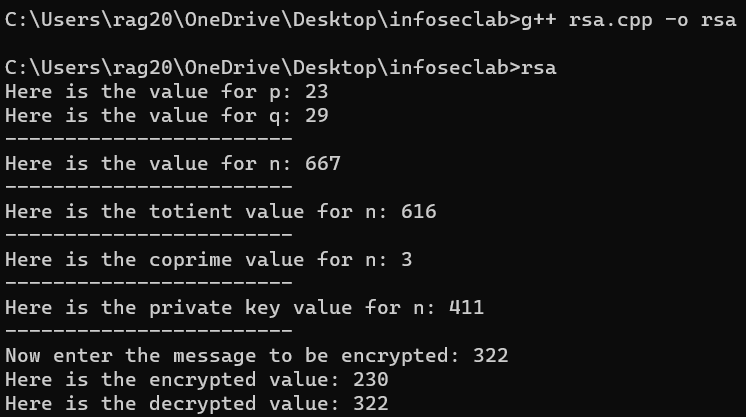
    int decryption=modExp(encryption,privKey,n);

    cout << "Here is the encrypted value: " << encryption << endl;

    cout << "Here is the decrypted value: " << decryption << endl;

}

OUTPUT:



DIFFIE-HELLMAN KEY TECHNIQUE

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <math.h>

#include <set>

#include <iostream>

#include <fstream>

using namespace std;

//To generate a prime number, i.e., n

bool isPrime(int num) {

    if (num <= 1)

        return false;

    for (int i = 2; i <= sqrt(num); i++) {

        if (num % i == 0) {

            return false;

        }

    }

    return true;

}

int generatePrime() {

    int num;

    while (true) {

        num=rand()%40+100;

        if (isPrime(num)) {

            return num;

        }

    }

}

// To find the primitive root, i.e., p

int gcd(int a, int b) {

    if (b == 0)

        return a;

    return gcd(b, a % b);

}

int findPrimitiveRoot(int n) {

    if (n <= 2)

        return -1; // No primitive root exists for n<=2

    int totient = n - 1;

    std::set<int> factors;

    // Find all prime factors of totient

    int temp = totient;

    for (int i = 2; i <= sqrt(temp); i++) {

        while (temp % i == 0) {

            factors.insert(i);

            temp /= i;

        }

    }

    if (temp > 1)

        factors.insert(temp);

    // Check potential primitive roots

    for (int i = 2; i <= n; i++) {

        bool isPrimitiveRoot = true;

        for (auto factor : factors) {

            if (gcd(i, totient / factor) == 1) {

                isPrimitiveRoot = false;

                break;

            }

        }

        if (isPrimitiveRoot)

            return i;

    }

    return -1; // No primitive root found

}

int main() {

    //Create an output stream object

    std::ofstream outputfile;

    outputfile.open("actuallog.txt");

    // Generate a prime number

    int n = generatePrime();

    cout << "Here is the value for n: " << n << endl;

    // Generate the primitive root of n

    int p = findPrimitiveRoot(n);

    cout << "Here is the primitive root of n: " << p << endl;

    cout << "------------------------" << endl;

    //Now to generate two keys, one for the sender and another for the receiver

    int X=rand()%21;

    int Y=rand()%56;

    cout << "The private keys, from sender: " << X << endl;

    cout << "The private keys, from receiver: " << Y << endl;

    cout << "------------------------" << endl;

    //Public key generation, i.e., A and B

    int A=(int)pow(p,(X%n));

    int B=(int)pow(p,(Y%n));

    cout << "The public key, from sender: " << A << endl;

    outputfile << A << std::endl;

    cout << "The public key, from receiver: " << B << endl;

    outputfile << B << std::endl;

    cout << "------------------------" << endl;

    //Final key generation, k1 and k2

    int k\_A=((int)(pow(B,X))%n);

    int k\_B=((int)(pow(A,Y))%n);

    cout << "The final key k1, from sender: " << k\_A << endl;

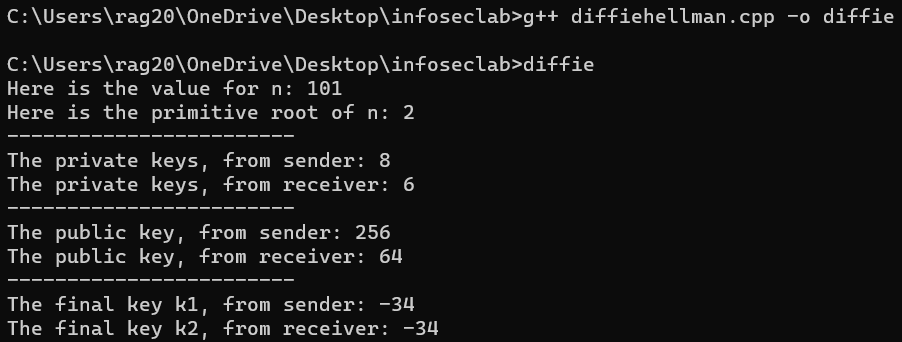
    cout << "The final key k2, from receiver: " << k\_B << endl;

    cout << "------------------------" << endl;

    outputfile.close();

}

OUTPUT:



MAN-IN-THE-MIDDLE ATTACK (ON DIFFIE HELLMAN)

(authentic encryption process)

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <math.h>

#include <set>

#include <iostream>

#include <fstream>

using namespace std;

//To generate a prime number, i.e., n

bool isPrime(int num) {

    if (num <= 1)

        return false;

    for (int i = 2; i <= sqrt(num); i++) {

        if (num % i == 0) {

            return false;

        }

    }

    return true;

}

int generatePrime() {

    int num;

    while (true) {

        num=rand()%40+100;

        if (isPrime(num)) {

            return num;

        }

    }

}

// To find the primitive root, i.e., p

int gcd(int a, int b) {

    if (b == 0)

        return a;

    return gcd(b, a % b);

}

int findPrimitiveRoot(int n) {

    if (n <= 2)

        return -1; // No primitive root exists for n<=2

    int totient = n - 1;

    std::set<int> factors;

    // Find all prime factors of totient

    int temp = totient;

    for (int i = 2; i <= sqrt(temp); i++) {

        while (temp % i == 0) {

            factors.insert(i);

            temp /= i;

        }

    }

    if (temp > 1)

        factors.insert(temp);

    // Check potential primitive roots

    for (int i = 2; i <= n; i++) {

        bool isPrimitiveRoot = true;

        for (auto factor : factors) {

            if (gcd(i, totient / factor) == 1) {

                isPrimitiveRoot = false;

                break;

            }

        }

        if (isPrimitiveRoot)

            return i;

    }

    return -1; // No primitive root found

}

int main() {

    //Create an output stream object

    std::ofstream outputfile;

    outputfile.open("actuallog.txt");

    // Generate a prime number

    int n = generatePrime();

    cout << "Here is the value for n: " << n << endl;

    // Generate the primitive root of n

    int p = findPrimitiveRoot(n);

    cout << "Here is the primitive root of n: " << p << endl;

    cout << "------------------------" << endl;

    //Now to generate two keys, one for the sender and another for the receiver

    int X=rand()%21;

    int Y=rand()%56;

    cout << "The private keys, from sender: " << X << endl;

    cout << "The private keys, from receiver: " << Y << endl;

    cout << "------------------------" << endl;

    //Public key generation, i.e., A and B

    int A=(int)pow(p,(X%n));

    int B=(int)pow(p,(Y%n));

    cout << "The public key, from sender: " << A << endl;

    outputfile << A << std::endl;

    cout << "The public key, from receiver: " << B << endl;

    outputfile << B << std::endl;

    outputfile << X << std::endl;

    outputfile << Y << std::endl;

    cout << "------------------------" << endl;

    //Final key generation, k1 and k2

    int k\_A=((int)(pow(B,X))%n);

    int k\_B=((int)(pow(A,Y))%n);

    cout << "The final key k1, from sender: " << k\_A << endl;

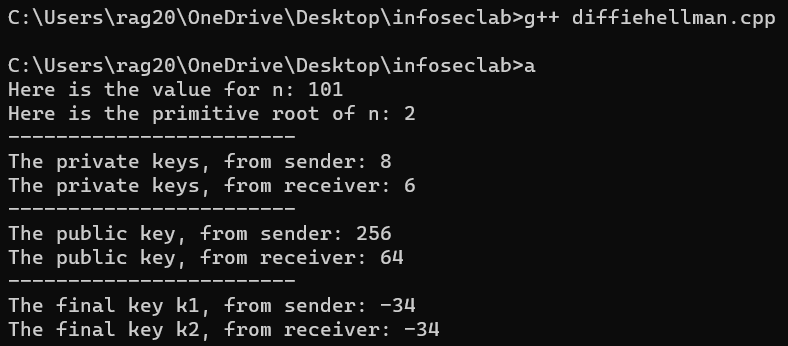
    cout << "The final key k2, from receiver: " << k\_B << endl;

    cout << "------------------------" << endl;

    outputfile.close();

}

OUTPUT:



(attacker)

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <math.h>

#include <set>

#include <iostream>

#include <fstream>

using namespace std;

bool isPrime(int num); // Function prototype

int generatePrime() {

    int num;

    while (true) {

        num = rand()%40 + 100;

        if (isPrime(num)) {

            return num;

        }

    }

}

bool isPrime(int num) {

    if (num <= 1)

        return false;

    for (int i = 2; i <= sqrt(num); i++) {

        if (num % i == 0) {

            return false;

        }

    }

    return true;

}

int gcd(int a, int b) {

    if (b == 0)

        return a;

    return gcd(b, a % b);

}

int findPrimitiveRoot(int n) {

    if (n <= 2)

        return -1; // No primitive root exists for n<=2

    int totient = n - 1;

    std::set<int> factors;

    // Find all prime factors of totient

    int temp = totient;

    for (int i = 2; i <= sqrt(temp); i++) {

        while (temp % i == 0) {

            factors.insert(i);

            temp /= i;

        }

    }

    if (temp > 1)

        factors.insert(temp);

    // Check potential primitive roots

    for (int i = 2; i <= n; i++) {

        bool isPrimitiveRoot = true;

        for (auto factor : factors) {

            if (gcd(i, totient / factor) == 1) {

                isPrimitiveRoot = false;

                break;

            }

        }

        if (isPrimitiveRoot)

            return i;

    }

    return -1; // No primitive root found

}

int main() {

    int n = generatePrime();

    cout << "Value for n: " << n << endl;

    int p = findPrimitiveRoot(n);

    cout << "Here is the primitive root of n: " << p << endl;

    int X=rand()%15;

    int Y=rand()%59;

    cout << "X: " << X << endl;

    cout << "Y: " << Y << endl;

    int A=(int)pow(p,(X%n));

    int B=(int)pow(p,(Y%n));

    cout << "A: " << A << endl;

    cout << "B: " << B << endl;

    cout << "------------------------" << endl;

    //Public Keys A and B need to be inserted in another txt file

    std::ofstream outputfile("attackerlog.txt");

    outputfile << A << endl << B <<endl;

    outputfile << X << endl << Y <<endl;

    //Man-in-the-middle attack demonstrated now

    //Both public keys are available to the attacker

    //From the log.txt file

    std::ifstream inputfile("actuallog.txt");

    int value1,value2,cmpr1,cmpr2;

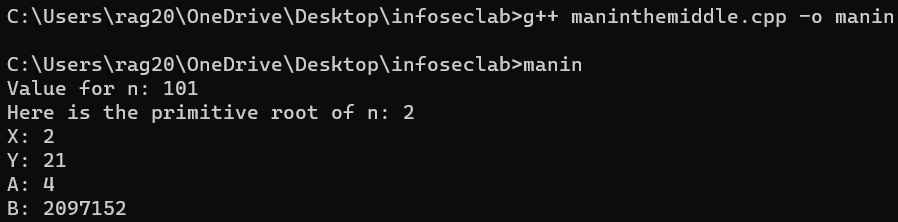
    inputfile >> value1;

    inputfile >> value2;

    //Successfully retrieved the public keys (A and B of the other side)

}

OUTPUT:



(attackvalidator – attacker side program)

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <math.h>

#include <iostream>

#include <fstream>

using namespace std;

int AfromActual,BfromActual,XfromActual,YfromActual;

int AfromAttacker,BfromAttacker,XfromAttacker,YfromAttacker;

void func1(){

std::ifstream inputfile("actuallog.txt");

    inputfile >> AfromActual;

    inputfile >> BfromActual;

    inputfile >> XfromActual;

    inputfile >> YfromActual;

    inputfile.close();

}

void func2(){

std::ifstream inputfile2("attackerlog.txt");

    inputfile2 >> AfromAttacker;

    inputfile2 >> BfromAttacker;

    inputfile2 >> XfromAttacker;

    inputfile2 >> YfromAttacker;

    inputfile2.close();

}

int main(){

    //Set the values to the global variables defined above

    func1();

    func2();

    //Now to calculate the keys

    int keyOneofActual=((int)(pow(BfromAttacker,XfromActual))%101);

    int keyTwoofAttacker=((int)(pow(AfromActual,YfromAttacker))%101);

    bool flag=false; //Attack set unsuccessful

    if(keyOneofActual == keyTwoofAttacker){

        flag=true;

    }

    int keyOneofAttacker=((int)(pow(BfromActual,XfromAttacker))%101);

    int keyTwoofActual=((int)(pow(AfromAttacker,YfromActual))%101);

    if(keyOneofAttacker!=keyTwoofActual){

        flag=false;

    }

    if(flag){

        cout << "Attack successful!";

    }

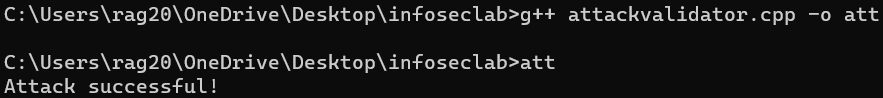
    else{

        cout << "Keys not matched. Attack unsuccessful.";

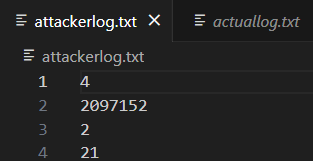
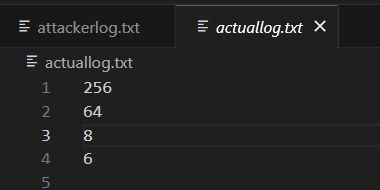
    }

}

OUTPUT:



(log files)



RESULT:

Thus, RSA public-key cryptosystem, Diffie Hellman Key technique and the Man-in-the-middle attack have been implemented accordingly.