Tarea 6: Cifrado RSA

Nombres y Apellidos: Giulia Alexa Naval Fernández

Enlace del Github: https://github.com/ganfy/Giulia\_Naval-Fern-ndez

Comentarios:

* En la generación de números primos grandes aleatorios (p y q) se utiliza la fórmula de Euler que, aunque tiene limitaciones (genera números desde el 41 hasta el 1601) cumple este propósito dentro del algoritmo.

Estructura del Algoritmo:

* Clases:
  + RSA: clase que contiene las claves y los métodos cifrado y descifrado
* Funciones
  + *Cifrado:* utiliza la exponenciación modular para hallar me mod n.
  + *Descifrado:* utiliza la exponenciación modular para hallar md mod n.
  + *Generar claves:* elige 2 números primos aleatorios con la función de Euler como p y q, luego halla n (p\*q) y oN ((p-1)(q-1)). Luego halla la clave e, generando un número aleatorio entre 0 y oN que sea coprimo con oN, verificando esto mediante el algoritmo de Euclides. Por último, halla la inversa de e mod oN, que sería la clave d.
  + *Euler:* fórmula de Euler para generar números primos
  + *ExpMod:* Exponeciación modular, halla el módulo de la base con exponente 1, si el exponente es impar, coloca este valor como valor inicial del resultado mod n, caso contrario, este sería 1, luego utiliza este módulo para hallar el resultado mod n, con la base al cuadrado, y divide el exponente entre 2; luego a la 4, y así sucesivamente, hasta que el exponente sea 0. En cada iteración, si el exponente es impar, multiplica el resultado mod n por el módulo hallado.

1. Código

.cpp

#include <iostream>

#include <string>

#include "álgebra.h"

using namespace std;

class RSA {

int d,p,q,oN;

public:

int e,n;

RSA(){

Generar\_claves();

cout<<p<<" "<<q<<" "<<n<<" "<<oN<<" "<<e<<" "<<d<<endl;

}

RSA(int e, int n){

this->e=e;

this->n=n;

}

int cifrado(int m)

{

int C=ExpMod(m,e,n);

return C;

}

int descifrado(int m)

{

int D=ExpMod(m,d,n);

return D;

}

void Generar\_claves(){

p=Euler(time(NULL));

q=Euler(time(NULL)+2);

while (p==q){

q=Euler(time(NULL)+2);

}

n=p\*q;

oN=(p-1)\*(q-1);

e=rand()%oN;

int ee=Euclides(e,oN);

while(ee!=1){

e=rand()%oN;

ee=Euclides(e,oN);

}

d=inversa\_1(e,oN);

}

};

int main()

{

RSA Receptor;

RSA Emisor(Receptor.e,Receptor.n);

int m=42;

int m\_C=Emisor.cifrado(m);

int m\_D=Receptor.descifrado(m\_C);

cout<<m\_C<<" "<<m\_D<<endl;

}

.h

#include<time.h>

#include <math.h>

using namespace std;

int divi(int long a, int n){//función módulo con restricción: n es positivo

int r=a-((a/n)\*n);

if (r<0){

return r+n;

}

return r;

}

int \*Euclides\_ext(int a, int b){//Euclides extendido

int x0=1,y0=0,x=0,y=1;

while(b>0){

int aux=x;

int q=a/b;

int r=a%b;

x=x0-q\*x;

x0=aux;

aux=y;

y=y0-q\*y;

y0=aux;

a=b;

b=r;

}

int result[]={a,x0,y0};

int \*p=result;

return p;//devuelve un puntero a un array con el mcd, x, y

}

int Euclides(int a, int b){//mcd

while(b>0){

int r=a%b;

a=b;

b=r;

}

return a;

}

int inversa\_1(int a, int b){//función inversa (asumiendo que el mcd es 1)

int mcd=\*Euclides\_ext(a,b);

int inv=\*(Euclides\_ext(a,b)+1);

if (inv<0){

return(divi(inv,b));

}

return inv;

}

int Euler(int seed){

srand (seed);

int n=rand()%40;

return n\*n+n+41;

}

int long ExpMod(int a, int n, int m){

int long res\_mod;

if (n%2==1){

res\_mod=a;}

else{

res\_mod=1;}

int long mod=a;

n/=2;

while(n>0){

mod=divi(mod\*mod,m);

if (n%2==1){

res\_mod=divi(res\_mod\*mod,m);

}

n/=2;

}

return res\_mod;

}