INFORME RSA

Galo De Paula Jimenez Araujo-33%.

Fernando Jair Peralta Bustamante-33%.

Valeria Fernanda Calle Rodriguez- 0% para este código, pero ayudó en la investigación de los algoritmos en las primeras investigaciones grupales lo cual permitió escoger los algoritmos más rápidos.

Brigham Jeffrey Caceres Gutierrez-33%.

Enlace GITHUB:

https://github.com/Brigham-CG/Brigham_CaceresGutierrez/tree/main/RSA%20con%20firma%20digital

Consideraciones:

El responsable del grupo debe de responder a este correo con el link del repositorio del github donde esté la carpeta del código. Anexe el informe en formato pdf.

Fecha de entrega: jueves 08 de julio 11:00 a.m.

De acuerdo al envío de sus trabajos, se programará las reuniones para la evaluación del código, el día martes de las 13:45 a 15:00 hras. Tiempo máximo de evaluación por grupo: 15 minutos.

Explicar y colocar el código que usaron para el algoritmo de criptografía RSA con firma digital, siguiendo la estructura que se les pide a continuación:

1. Estructura del main()

El main tiene un menú para seleccionar lo que se desea en el momento, en primero tiene “Muestra”, hace una prueba de cifrado y descifrado automático, con emisor y receptor ya establecidos.

“Enviar mensaje”, instancia un emisor con una cantidad de bits a elegir, al cual establecerá la clave pública del receptor y su N, para el cifrado; también imprime tu clave pública y N para enviar al emisor para el descifrado de la firma digital.

“Recibir mensaje”, instancia un receptor, es similar al enviar mensaje, con la diferencia de que se usa el método de descifrado.

```
int select = 0;

   cout << "1) Muestra\n";

   cout << "2) Enviar mensaje\n";

   cout << "3) Recibir mensaje\n";

   cout << "Seleccione: ";

   cin >> select;


   if (select == 1){


       int bits = 0;

       cout << "Ingrese la cantidad de bits: "; cin >> bits;

       cin.ignore();

       RSA receptor(bits);


       RSA emisor(bits);

       receptor.establecer(emisor.getE(), emisor.getN());

       emisor.establecer(receptor.getE(), receptor.getN());

       string mensaje;
```

```
        getline(cin, mensaje);

        cout << "\nMensaje cifrado\n";

        string mensajeCifrado = emisor.cifrar(mensaje);

        string firma =
emisor.firma_digital_cifrado(emisor.leer_datos("firma.txt"));


        cout << "Emisor: " << mensajeCifrado << endl;

        cout << "Firma: " << firma << endl;


        cout << "\nMensaje descifrado\n";

        string mensajeDescifrado = receptor.descifrar(mensajeCifrado);

        string firmaDescifrada = receptor.firma_digital_descifrado(firma);


        cout << "Receptor: " << "'" << mensajeDescifrado <<"'"<< endl;

        cout << "Firma: " << "'" << firmaDescifrada << "'" << endl;

    }

    else if(select == 2)

    {

        int bits = 0;

        cout << "Ingrese la cantidad de bits: "; cin >> bits;


        ZZ e, N;

        cout << "Receptor e: ";cin >> e;
```

```
        cout << "Receptor N: ";cin >> N;


        cin.ignore();
        RSA emisor(bits);
        emisor.establecer(e, N);
        string mensaje;
        getline(cin, mensaje);
        string mensajeCifrado = emisor.cifrar(mensaje);
        string firma =
emisor.firma_digital_cifrado(emisor.leer_datos("firma.txt"));
        cout << "Mensaje cifrado: " << mensaje << endl;
        cout << "Emisor: " << mensajeCifrado << endl;
        cout << "Firma: " << firma << endl;
    }
    else if(select == 3)
    {
        int bits = 0;
        cout << "Ingrese la cantidad de bits\n"; cin >> bits;
        RSA receptor(bits);
        cout << "e: " << receptor.getE() << endl;
        cout << "n: " << receptor.getN() << endl;


        ZZ e, N;
```

```
        cout << "Emisor e:";cin >> e;
        cout << "Emisor N: ";cin >> N;
        receptor.establecer(e, N);

        cin.ignore();
        string mensaje;
        cout << "Mensaje cifrado: ";
        getline(cin, mensaje);
        string firma;

        cout << "Firma: ";
        getline(cin, firma);
        cout << "Mensaje descifrado: " <<receptor.descifrar(mensaje) <<
endl;
        cout << "Firma: " << receptor.firma_digital_descifrado(firma) <<
endl;
    }
```

2. Generación de claves

- Generación de números aleatorios

Para la generación de números aleatorios se utiliza tres métodos, “generate_bit”, “generate_random” y

“generar_aleatorio”.
El primero establece la semilla en función del reloj del CPU el cual se sitúa en un tiempo inicial y en uno final, obteniendo microsegundos.

Luego está generate_random, utiliza el método anterior. básicamente itera el número de veces que se le establece a bits. Se genera preliminarmente usa generate bit como base para luego volver a generar más bits y posicionarnos con el power, luego multiplicando por dos y así en cada iteración para luego llegar a la longitud en bits deseado.

El último lo único que hace es utilizar generate_random, pero asegurarse que se establece entre un parámetro de números dado(2^(bits/2) y 2^(bits-1)).

```
ZZ RSA::generate_bit() {

   std::chrono::steady_clock::time_point begin =
std::chrono::steady_clock::now();// Start clock
   cout << "."; // Linea important para incrementar la variabilidad
   std::chrono::steady_clock::time_point end =
std::chrono::steady_clock::now();// End clock
ZZ seed(std::chrono::duration_cast<std::chrono::microseconds> (end -
begin).count());
   ZZ seed_2 = modulo(seed, ZZ(2));
   return seed_2;
}

ZZ RSA::generate_random(int bits)
{
   ZZ random(0);
   ZZ power(2);
   random = random + (generate_bit());
   for (int i = 1;i<bits;i++) {
       random = random + (generate_bit()*power);
       power = power * 2;
   }
```

```
    return random;
}
ZZ RSA::generar_aleatorio(int bits){

    ZZ min(exponenciacion(to_ZZ(2), to_ZZ(bits))>>1),
max(exponenciacion(to_ZZ(2), to_ZZ(bits)) - 1);

    ZZ number;
    do{
        number = RandomLen_ZZ(bits);
        // number = generate_random(bits); // No funcional para Linux
    }while(number < min || number > max);

    return number;
}
```

- Generación de primos

Lo que hace es entrar en un bucle FOR(para más velocidad en ejecución), el cual genera números aleatorios, y se detiene hasta hacer la prueba probabilística con Miller-Rabin el cual fue el más rápido de nuestra investigación para números primos.

```
ZZ RSA::generar_primo(int bits){

    ZZ prime;

    for (;!miller_rabin(prime);prime = generar_aleatorio(bits));

    return prime;
}
```

```
bool RSA::miller_rabin(ZZ n){

    if ((n & 1) == 0){
        return false;
    }
    ZZ s(0);
    ZZ t = n - 1;
    while ((t & 1) == 0) {
        s++;
        t >>= 1;
    }
    ZZ a(2);
    for (int i = 0; i < 10; i++){

        ZZ x = left_to_right_binary_exponenciacion(a, t, n);
        if (x == 1 || x == (n-1))
            continue;
        for (ZZ r(0); r < (s-1); r++){
            x = left_to_right_binary_exponenciacion(x, to_ZZ(2), n);
            if(x == 1){
                return false;
            }
            else if(x == n - 1)
                break;
        }
        if(x != n - 1 )
            return false;
        ZZ a = RandomBnd(n-3) + 3;
    }

    return true;
```

```
}
```

- Algoritmo de Euclides

El de menor resto fue el más rápido en nuestra investigación.

```
ZZ RSA::euclides_menor_resto(ZZ a, ZZ b){

   ZZ c, d, r;
   if (a == 0)
       c = b;
   else
   {
       c = a;
       d = b;
       while(d != 0)
       {
           r = c - d * (c/d + 1/2);
           c = d;
           d = r;
       }
   }
   return abs(c);
}
```

- Inversa

▪ Euclides extendido

```
vector<ZZ> RSA::euclides_extendido(ZZ a, ZZ b)
{
   // a >= b
   // numbers[0] -> x
   // numbers[1] -> y
```

```
    // numbers[2] -> d

    vector<ZZ> numbers(3);
    if (b == 0)
    {
        numbers[0]= 1;
        numbers[1] = 0;
        numbers[2] = a;
    }

    ZZ x2(1), x1(0), y2(0), y1(1);
    while(b > 0)
    {
        ZZ q = a / b;
        ZZ r = a - q*b;
        numbers[0] = x2 - q*x1;
        numbers[1] = y2 - q*y1;
        a = b;
        b = r;
        x2 = x1;
        x1 = numbers[0];
        y2 = y1;
        y1 = numbers[1];
    }

    numbers[2] = a;
    numbers[0] = x2;
    numbers[1] = y2;

    return numbers;
}
```

3. Formación de Bloques

- Llenar ceros

```
// cantidad de digitos del alfabeto

   int cantA = to_string(alfabeto.length() - 1).length();

   // separando en numeros

   for (int i = 0; i < mensaje.length(); i ++){
       int posI = alfabeto.find(mensaje[i]);
       string pos = to_string(posI);

       int tamanio = cantA-pos.length();

       transform.append(tamanio, '0');
       transform +=pos;

   }
```

- Convertir string a enteros (ZZ)

Este es el único lugar donde se utiliza, y es principalmente el conv<ZZ>().

```
       ZZ number(conv<ZZ>(transform.substr(i,cantNR).c_str()));
```

- ZZ a string

se utiliza la libreria “sstream”, básicamente pasa los bits de ZZ almacenarlos en un buffer y convertirlos a string;

```
string ZZ_to_string(ZZ num)
{
```

```
    stringstream buffer;
    buffer<<num;
    return buffer.str();
};
```

- Dividir bloques

```
for (int i = 0; i < mensaje.length(); i ++){
        int posI = alfabeto.find(mensaje[i]);
        string pos = to_string(posI);

        int tamanio = cantA-pos.length();

        transform.append(tamanio, '0');
        transform +=pos;

    }
```

4. Exponenciación modular

Se utilizó el algoritmo de “left to right binary” el cual fue el más rápido de nuestra investigación, aunque solo fue para el cifrado.

Tambien se utilizo el resto chino, pero solo para el descifrado.

```
ZZ RSA::left_to_right_binary_exponenciacion(ZZ a, ZZ e, ZZ n) {

    ZZ A(1);

    string bin = toBinary(e);

    for (int i = bin.size(); i != -1; i--) {
```

```
        A = modulo(A * A, n);

        if (bin[i] == '1') {

            A = modulo(A * a, n);

        }

    }

    return A;

}

ZZ RSA::restoChino(ZZ c, ZZ d, ZZ N)
{
    vector <ZZ> a = {left_to_right_binary_exponenciacion(c,modulo(d, p - 1),
N), left_to_right_binary_exponenciacion(c,modulo(d, q- 1), N)};

    vector<ZZ> inversos = euclides_extendido(p, q);
    return modulo(a[0]*q*modulo(inversos[1],N) +
a[1]*p*modulo(inversos[0],N), N);

}
```

5. Función de cifrado

Hace el procedimiento con bloques, después añade el tamaño del alfabeto hasta que se divisible por el N del receptor, y luego se pasa a la exponenciación modular.

```
string RSA::cifrar(string mensaje){
```

```
   // obtener la cantidad de digitos de N

   int cantNR = ZZ_to_string(N).length();

   cantNR--;

   // cantidad de digitos del alfabeto

   int cantA = to_string(alfabeto.length() - 1).length();

   // separando en numeros

   string transform = bloques(mensaje, cantA);

   // tamaño de alfabeto

   string alfLen = to_string(alfabeto.length());

   // dividir

   while(modulo(to_ZZ(transform.size()), to_ZZ(cantNR)) != 0){
transform+=alfLen;
}

   // exponenciacion

     string cifrado;

   for (int i = 0; i < transform.length(); i += cantNR)

   {
       ZZ number(conv<ZZ>(transform.substr(i,cantNR).c_str()));
       ZZ exp = left_to_right_binary_exponenciacion(number, eR, NR);
       string ci = ZZ_to_string(exp);
```

```
        int tamanio = cantNR-ci.length()+1;
        cifrado.append(tamanio, '0');
        cifrado +=ci;
    }
    return cifrado;

}
```

6. Función de descifrado

El descifrado es parecido, pero aplicado de forma inversa para la exponenciación se utiliza el “resto chino”.

Para detectar hasta donde corresponde los datos cifrados íntegros, es decir que parte son los bloques para procesarlos y buscarlos en el alfabeto del extra que se concatena con el tamaño del alfabeto, busca de atrás hacia delante hasta que ese número ya no se encuentre, y se corta.

```
string RSA::descifrar(string mensaje){

    // obtener la cantidad de digitos de N

int cantN = ZZ_to_string(N).length();

    // exponenciacion

    string transform;

    for (int i = 0; i < mensaje.length(); i+=cantN)

    {
        ZZ number(conv<ZZ>(mensaje.substr(i,cantN).c_str()));
```

```
        ZZ des = restoChino(number, d, N);
        string di = ZZ_to_string(des);
        int tamanio = cantN-di.length()-1;
        transform.append(tamanio, '0');
        transform += di;
    }

    // cantidad de digitos del alfabeto

    int cantA = to_string(alfabeto.length() - 1).length();

    // tamaño de alfabeto

    string alfLen = to_string(alfabeto.length());

    int fin = 0;

    for (fin = transform.length() - cantA; fin > 0; fin -= cantA)
    {
        if(transform.substr(fin, cantA) != alfLen)

            break;

    }

    transform = transform.substr(0, fin + cantA);

    // separando en numeros

    string descifrado;

    for (int i = 0; i < transform.length(); i+=cantA){
        int pos = stoi(transform.substr(i, cantA));
        char carc = alfabeto[pos];
        descifrado += carc;
```

```
    }
    return descifrado;
}
```

7. Firma digital

Es una copia de cifrado y descifrado con la diferencia de que se añade el “resto chino” en caso del cifrado, y el descifrado con “left to right binary”.

Aunque no es necesario se utiliza un lector de texto para obtener los datos de la firma de un fichero txt, esto es omitible ya que se utiliza como argumento al cifrar la firma.

```
string RSA::firma_digital_cifrado(string data)
{
    // obtener la cantidad de digitos de N
    int cantN = ZZ_to_string(NR).length();
    cantN--;

    // cantidad de digitos del alfabeto
    int cantA = to_string(alfabeto.length() - 1).length();

    // separando en numeros
    string transform = bloques(data, cantA);

    // tamaño de alfabeto
    string alfLen = to_string(alfabeto.length());

    // dividir
    while(modulo(to_ZZ(transform.size()), to_ZZ(cantN)) != 0){
        transform+=alfLen;
```

```
    }


     // exponenciación
    string firma;
    for (int i = 0; i < transform.length(); i += cantN)
    {
        ZZ number(conv<ZZ>(transform.substr(i,cantN).c_str()));
        ZZ exp = restoChino(number, d, N);
        exp = left_to_right_binary_exponenciacion(exp, eR, NR);
        string ci = ZZ_to_string(exp);
        int tamanio = cantN-ci.length()+1;
        firma.append(tamanio, '0');
        firma+=ci;
    }

    return firma;
}

string RSA::firma_digital_descifrado(string firma)
{
    // obtener la cantidad de digitos de N
    int cantN = ZZ_to_string(N).length();

    // exponenciacion
    string transform;

    for (int i = 0; i < firma.length(); i+=cantN)
    {
        ZZ number(conv<ZZ>(firma.substr(i,cantN).c_str()));
        ZZ des = restoChino(number, d, N);
        des = left_to_right_binary_exponenciacion(des, eR, NR);
        string di = ZZ_to_string(des);
```

```
        int tamanio = cantN-di.length()-1;
        transform.append(tamanio, '0');
        transform += di;
    }

    // cantidad de digitos del alfabeto
    int cantA = to_string(alfabeto.length() - 1).length();

    // tamaño de alfabeto

    string alfLen = to_string(alfabeto.length());
    int fin = 0;
    for (fin = transform.length() - cantA; fin > 0; fin -= cantA)
    {
        if(transform.substr(fin, cantA) != alfLen)
            break;
    }

    transform = transform.substr(0, fin + cantA);
    // separando en numeros
    string descifrado;
    for (int i = 0; i < transform.length(); i+=cantA){
        int pos = stoi(transform.substr(i, cantA));
        char carc = alfabeto[pos];
        descifrado += carc;
    }
  return descifrado;

}

string RSA::leer_datos(string fichero)
```

```
{


   ifstream data (fichero, ifstream::in);
   string readData, line;
   while(getline(data, line))
   {
       readData+=line;
   }
   data.close();
   return readData;
}
```