UNIVERSIDAD VERACRUZANA

Facultad de Estadística e informática

Un análisis computacional del RSA: Un enfoque didáctico

MODALIDAD:

PRACTICO EDUCATIVO

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN TECNOLOGÍAS COMPUTACIONALES

PRESENTA:

Luis Abraham Tlapa Garcia

DIRECTORA:

Dra. Juana Elisa Escalante vega

CODIRECTOR:

Dr. Sergio Francisco Salem Silva

XALAPA DE ENRIQUEZ,VER. FEBRERO 2019

Dedicatoria...

A mis padres por ser el pilar fundamental en todo lo que soy,en toda mi educación, tanto académica, como de la vida,por su incondicional apoyo perfectamente mantenido a través del tiempo.

A mis maestros la Dra. Juana Elisa Escalante vega y el Dr.Sergio Francisco Salem Silva por su gran apoyo y motivación para la culminación de nuestros estudios profesionales y para la elaboración de esta tesis.

Todo este trabajo ha sido posible gracias a ellos

Resumen

En este trabajo se desarrollaron y analizaron los algoritmos fundaméntales de un criptosistema que se basan en teoría de números como: Encontrar un número primo, Exponenciación y Encontrar inverso multiplicativo modulo (n), para un número de al menos 1024 bits. Para el análisis se implementaron los programas necesarios para un sistema de seguridad, codificados en Python en el entorno de desarrollo Jupyter. Los programas implementados son eficientes en cuanto al tiempo de ejecución y tamaño del número que producen o maneja. Para esto se realizó una interfaz didáctica de un sistema de seguridad en la que el usuario puede interactuar. Este posibilita la comprensión del funcionamiento de un sistema criptográfico como es el RSA, lo que permite mejorar algunas propiedades del sistema, como la seguridad y rapidez.

Índice general

Índice de figuras

Capítulo 1

Introducción

Las áreas de la información y de la comunicación se han visto en aumento gracias al desarrollo de la tecnología (Diaz Lazo Peréz Gutierrez, 2010). Conformándose en un área de conocimiento que se conoce bajo el nombre de Tecnologías de la información y la comunicación. Las tecnologías desarrolladas en esta área han revolucionado los procedimientos de transmisión de la información. Hoy en día el uso de las Tecnologías de la información y la comunicación se encuentra en aumento (Vorndran, 2014) y son fundamentales para saber qué recursos se necesitan para proteger la transmisión y así garantizar la seguridad de los datos (Burgos Salazar Campos, 2008). Se entiende por seguridad al conjunto de normas, procedimientos y herramientas, que tienen como objetivo garantizar la disponibilidad, integridad, confidencialidad y buen uso de la información (Voutssas, 2010). En la actualidad existen una serie de sistemas de seguridad para la transmisión de datos la mayoría de estos sistemas están basados en métodos de criptografía (Alvarez , Sánchez, Garcia, 2015). Cabe mencionar que el uso de la criptología incluye técnicas para ocultar de forma segura la información a todos los destinatarios, excepto los que son destinatarios finales (Cooper Goldreich, 2000). Existen dos tipos la criptografía, simétrica y asimétrica:

- La criptografía simétrica se refiere al conjunto de métodos que permiten tener comunicación segura entre las partes siempre y cuando anteriormente se hayan intercambiado la clave correspondiente que llamaremos clave simétrica. La simetría se refiere a que las partes tienen la misma clave tanto para cifrar como para descifrar (Litwak Escalante, 2004).

- La criptografía asimétrica aparece en los años 70 y utiliza algoritmos matemáticos relacionados con números primos. Y para este caso cada usuario pose una pareja de claves Clave privada y clave pública (Litwak Escalante, 2004). Un ejemplo de criptografía asimétrica es el RSA. Utilizaremos criptografía asimétrica porque es uno de los métodos más utilizados en los sistemas de seguridad en la actualidad (Sánchez Acosta, 2012).

RSA es un algoritmo de cifrado asimétrico desarrollado en el año 1977. Este algoritmo se basa en escoger dos números primos grandes elegidos de forma aleatoria. La principal ventaja de este algoritmo desde el punto de vista de seguridad se basa en

la dificultad de factorizar números "grandes"(López Lucen , 2011). Es por eso que se han hecho muchos avances en este problema (factorizar).

En este trabajo se desarrolla la implementación de un sistema de seguridad criptográfico RSA con fines didácticos para lo cual se analizaron y desarrollaron los algoritmos fundamentales que requiere este sistema así mismo se analiza la seguridad del sistema presentando algunas técnicas para factorizar n.

1.1. Antecedentes

La criptografía nace debido a la necesidad del hombre para comunicarse por escrito y preservar la privacidad de la información, ya sea por motivos militares, diplomáticos, comerciales, entre otros (Gutierrez, 2017). Mantener la información en secreto es esencial para conservar la integridad de un individuo o en ocasiones hasta de una comunidad completa (Maubert, 2015).

Los primeros mensajes cifrados de los que se tiene registro aparecieron durante el siglo V antes de Cristo, cuando los griegos crearon un instrumento para cifrar mensajes. Dicho instrumento es conocido como escítala espartana y consistía en un cilindro de madera en el cual se enrollaba una cinta de cuero o tela y se escribía en un lado del cilindro. Es importante mencionar que el mensaje en la cinta sin estar enrollado en el cilindro resultaba confuso e incoherente para cualquier persona que intentara leerlo (Fernandez, 2004).

Años después, a mediados del siglo II antes de Cristo, aparece el primer sistema de cifrado en la historia que funciona por sustitución de caracteres, la invención se le atribuye al historiador griego Polybios. Este procedimiento de cifrado consistía en la sustitución de un carácter por un par de caracteres que le correspondían según una tabla que se diseñaba con este propósito (Aguirre Ramió, 2016).

Cincuenta años más tarde, en el siglo I a.C. aparece un nuevo procedimiento de cifrado, el cual fue utilizado por Julio César para la comunicación con sus oficiales y es conocido como cifrado por desplazamiento.

Cientos de años después, en 1790 Thomas Jefferson creó un cilindro formado por varios discos coaxiales en donde cada uno tenía grabado en la parte exterior el alfabeto. Cada disco se ajustaba de tal modo que en una generatriz del cilindro se formara el mensaje y el criptograma se obtenía de cualquiera de las otras generatrices (UNAM Facultad de ingeniería, 2017).

El ingeniero alemán Arthur Scherbius en 1923 creo la máquina de cifrar más famosa de la historia debido a su protagonismo en la segunda guerra mundial: la maquina Enigma. Esta consistía en un conjunto de 26 rotores de contactos eléctricos, uno por

cada letra, montados sobre un eje que dada una vuelta completa a cada rotor a partir de un movimiento del siguiente producido al presionar cada tecla del teclado con el que contaba (Sánchez Muñoz, 2013).

1.2. Planteamiento del problema

Actualmente no es raro escuchar sobre violaciones a la seguridad informática, por ejemplo: algunas personas manifiestan que sean hecho compras con sus tarjetas que ellos no reconocen o bien que su identidad ha sido usurpada. Hacia finales del 2018 escuchamos una noticia de que la seguridad de muchos bancos mexicanos había sido quebrantada lo que generó cuantiosas pérdidas a los bancos. De esto puede deducir que los ataques a la seguridad informática están evolucionando. Por tanto es necesario mejorar constantemente estos sistemas informáticos. Para llevar a cabo estas mejoras debemos conocer muy bien los elementos que constituyen un buen sistema de seguridad informático. Por ejemplo:

- Alto costo de un ataque en términos de complejidad.
- Corto tiempo de ejecución del algoritmo
- Algoritmo criptográfico que permita mandar un mensaje en código y descifrar

En este trabajo se pretende estudiar con cierta profundidad estos y algunos elementos más, para después integrarlos para construir una primera versión de un prototipo de seguridad informática que sea flexible en el sentido que sea fácil mejorar los elementos de estos para mejorar la seguridad.

Este es un primer acercamiento a seguridad informática por esto y cuestiones de tiempo nuestro prototipo solo será probado en situaciones artificiales.

1.3. Objetivos generales y específicos

Analizar los algoritmos fundamentales del sistema criptográfico RSA como: generación de números primos muy grandes, elevación de un número a una potencia grande modulo n. Determinación del inverso multiplicativo de un número. Lo anterior con el objetivo de implementar una primera versión del sistema de seguridad basado en RSA con los algoritmos anteriores.

1.3.1. Específicos

1. Realizar y documentar los algoritmos matemáticos para RSA

2. Probar los algoritmos matemáticos
3. Desarrollar prototipo para el sistema RSA
4. Implementar el prototipo del sistema RSA
5. Probar el sistema de seguridad

Capítulo 2

Marco Teórico

La criptografía moderna se basa mucho en conceptos de álgebra, y para eso necesitamos conocer algunos temas importantes como divisibilidad, números primos, enteros modulo (n) (denotados por Z_n), entre otros para poder desarrollar algunos métodos criptográficos.

2.1. Teoría de números

La teoría de los números es el estudio de números enteros, el conjunto de enteros se denomina con el símbolo Z. Los números enteros se pueden agregar, restar, multiplicar de manera habitual, que satisfacen las reglas de la aritmética ley conmutativa, ley asociativa, ley distributiva . (Hoffstein , Pipher , Silverman, 2000).Entre los números enteros positivos hay una subclase muy importante, la clase de los primos. Un número entero positivo p se considera primo si cumple las siguientes condiciones:

- $p > 1$,
- p no tiene divisores positivos además de 1 y p.

Por ejemplo, el número 37 es un primo dado que es mayor que 1 y no posee divisores enteros positivos además 1 y el propio 37. Es importante resaltar que 1 no se considera primo dado que no cumple la primera condición. Normalmente reservamos la letra p para los números primos.

Un concepto importante para esta investigación es el término de divisibilidad (Yan, Yung Rief, 2013).

La teoría de la divisibilidad ha sido estudiada durante aproximadamente 3000 años, cuando los antiguos griegos consideraron problemas sobre los números particularmente problemas sobre primos, algunos de los cuales aún no se han resuelto.

Definición **2.1** *Los números* $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$ *denotados por:* Z*, se les llama enteros positivos a los números* $1, 2, 3$ *denotados por:* Z^+

Definición **2.2** *Dados $a, b \in Z$, con $a \neq 0$ se dice que a divide a b y se denota por $a|b$ si existe $k \in Z$ tal que, $ak = b$*

El máximo común divisor de a y b se denota MCD (a, b). Una forma de conocer el $MCD(748, 2024) = 44$ es hacer listas de todos los enteros positivos divisores de 748 y de 2024 como las siguientes:

- Divisores de $748 = \{1, 2, 4, 11, 17, 22, 34, 44, 68, 187, 374, 748\ \}$,
- Divisores de $2024 = \{1, 2, 4, 8, 11, 22, 23, 44, 46, 88, 92, 184, 253, 506, 1012, 2024\}$.

Cuando revisamos estas dos listas encontramos que la entrada común más grande es 44. Que incluso a partir de este pequeño ejemplo, está claro que este no es un método muy eficiente si necesitamos calcular el máximo común divisor de números grandes.

Entonces calculamos $MCD(2024, 748)$ usando el algoritmo euclidiano, que es un método de división más rápido y eficiente para calcular el MCD como demostraremos a continuación:

2024 = 748 * 2 + 528

748 = 528 * 1 + 220

528 = 220 * 2 + 88

220 = 88 * 2 + 44 –MCD = 44

88 = 44 * 2 + 0

A continuación en la Figura 2.1 mostraremos el código en Python y la documentación del algoritmo euclidiano con los datos que usamos anteriormente.

```python
from math import *
import time
#Declaramos mis variables e introducimos los numeros para conocer su MCD
nume1 = int(input("Introduce el primer numero: "))
nume2 = int(input("Introduce el segundo numero: "))
a=0
b=0
#Conocer el tiempo que tarda el proceso con la libreria time
inicio_de_tiempo = time.time()
# Solo por si el primer numero es mas chico que el segundo cambian los valores
if nume1 < nume2:
    a= nume2
    b= nume1
else:
    a= nume1
    b= nume2
while b != 0:
    print('%s = %s * %s + %s' % (a, floor(a/b), b, a % b))
    (a, b) = (b, a % b)

print('---MCD es  %s----' % a)
tiempo_final = time.time()
tiempo_transcurrido = tiempo_final - inicio_de_tiempo
print ("\nTomo %d segundos." % (tiempo_transcurrido))
```

```
Introduce el primer numero: 2024
Introduce el segundo numero: 748
2024 = 2 * 748 + 528
748 = 1 * 528 + 220
528 = 2 * 220 + 88
220 = 2 * 88 + 44
88 = 2 * 44 + 0
---MCD es  44----

Tomo 0 segundos.
```

Figura 2.1: Algoritmo Euclidiano Extendido

2.2. Enteros modulo n

Sea $n \in Z^+$ y a,$b \in Z$ entonces a es congruente con b modulo n (Denotado por: $a \equiv b \ mod \ n$), si a y b dejan el mismo residuo cuando se divide por n o de manera equivalente si $n|(a-b)$.

La relación en $Z \times Z$, definida por: $a \sim b$ si solo si $a \equiv b \ mod \ n$. Es una relación de equivalencia (Bor, 1998), al conjunto $[a] = \{x \mid x \equiv a \ mod \ n\}$ se le llama la clase de residuo modulo n, a es un representante de la clase. El conjunto de todas las clases de equivalencias $\{[a] \mid a \in Z\}$ se le denomina los enteros modulo n y es denotado por Z_n (Philippe Aumasson, 2018).

Como para $z \in Z$, existe un único r tal que: $z \equiv r \ mod \ n$, donde $0 \leq r \leq n-1$, entonces solo existen r clases de equivalencias en Z_n, es decir $Z_n = \{[0], [1], \ldots, [n-1]\}$. Algunas veces identificamos las clases residuales con sus representantes naturales i.e. identificamos a Z_n con $\{0, 1, \ldots, n-1\}$.

Definición **2.3** *En Z_n definimos la suma y el producto de clases residuales de la siguiente manera:*

$$[a] + [b] = [a + b]$$

$$[a] * [b] = [a * b]$$

Se puede probar que la suma y el producto están bien definidos (Vergnaud, 1990) no dependen de los representantes, es decir $a \equiv a' \ mod \ (n)$ y $b \equiv a' \ mod \ (n)$, entonces $a + b \equiv (a' + a') \ mod \ (n)$ y $a * b \equiv (a' * a') \ mod \ (n)$

Ejemplo para la suma y producto

Sea $n = 18$, $a = 7$, $b = 11$, $a1 = 25$, $b1 = 29$ realizamos la suma de clases que es igual a $x = (a + b)\,\%n$ y $x1 = (a1 + b1)\,\%n$ entonces: $x = (7 + 11)\,\%18 = 0$, $x1 = (25 + 29)\,\%18 = 0$, siguiendo la definición podemos realizar lo siguiente: $x2 = (a + b1)\,\%n$ entonces: $x = (7 + 29)\,\%10 = 0$ y como tenemos el mismo resultado decimos que $a1$ y $b1$ también pertenecen a la clase $[0]$.

Para la multiplicación sea $n = 35$, $a = 9$, $b = 6$, $a1 = 44$, $b1 = 41$ y realizamos la multiplicación que es igual a $x = (a * b)\,\%n$ y $x1 = (a1 * b1)\,\%n$ entonces: $x1 = (9{*}6)\,\%35 = 19$, $x1 = (44{*}41)\,\%35 = 19$, podemos demostrar que multiplicando $x2 = (a * b1)\,\%35$ es: $x2 = (9 * 41)\,\%35 = 19$ entonces $a1$ y $b1$ pertenecen a la clase de $[19]$

Otro hecho importante es cuando $[a] * [b] = [0]$ puede ser que $[a] \neq [0]$ y $[b] \neq [0]$, por ejemplo, $[2].[3] = [0]$ en Z_6 se les llama divisores de cero. En la figura 2.2 se encuentra el código del algoritmo que encuentra los números que pertenecen a $[0]$.

```python
print("Ingresa un numero entero para conocer que numeros pertenecen a [0] ")
n = int(input("numero: "))
x=0
FIN = False
l=0
d=0
fin=0
if n>0:

    for p in range(1,n):
        for l in range(1,n):
            d =(p)*(l)
            fin=d%n
            #print(p,"por",l,"es igual a",d,"mod",n,"=", fin)

            if fin == 0:
                print(p,"y",l,"son divisores de 0")
else:
    print("El numero ingresado no es correcto. Intentalo de nuevo")
```

```
Ingresa un numero entero para conocer que numeros pertenecen a [0]
numero: 9
3 y 3 son divisores de 0
3 y 6 son divisores de 0
6 y 3 son divisores de 0
6 y 6 son divisores de 0
```

Figura 2.2: Código para encontrar los números que pertenecen a [0]

En la figura anterior el usuario tiene que ingresar un número positivo, entonces el programa busca mediante fuerza bruta todos los números que hay en ese rango multiplicando y sacando su módulo y si después de eso el resultado final es 0 entonces los números pertenecen a esa clase.

En Z los únicos elementos que satisfacen $a * b = 1$, son $a = b = 1$ y $a = b = -1$. Sin embargo en Z_n tenemos lo siguiente:

Un elemento $[a]$ en Z_n tiene inverso multiplicativo $[b]$, si $a * b \equiv 1 \ mod \ n$, por lo tanto se cumple una ecuación de la forma $nm + ab = 1$ con m adecuada, esto implica que $MCD(a, n) = MCD(b, n) = 1$.

Definición **2.4** *Decimos que* $[a]$ *es una unidad en* Z_n *si tiene inverso multiplicativo.*

El conjunto de unidades en Z_n es:

$$Z^*{}_n = \{[a] \mid 1 \leq a \leq n-1 \text{ y } MCD\,(a, n) = 1\}$$

$|Z^*{}_n|$, es igual al número de primos relativos con n en el intervalo $1, 2, \ldots, n-1$ (donde $|Z^*{}_n|$ denota la cardinalidad $Z^*{}_n$).

Definición **2.5** *La función definida por:* $\varphi(n) = |Z^*{}_n|$ *será llamada la función de Euler.*

El grupo $Z^*{}_p$ es un grupo con $p-1$ elementos es decir: $Z^*{}_p = \{g, g^2, \ldots, g^{p-1} = [1]\}$ cuando $g \in Z_p \setminus \{0\}$

En la Figura 2.3 encontramos el código del programa para encontrar inversos multiplicativos que lo hace mediante la multiplicación con todos los números menores al número ingresado por el usuario.

```
#Librerias necesarias para el codigo ya que usaremos numpy y pandas para la creacion de tablas
import numpy as np
import pandas as pd
from IPython.display import display
#Pedimos el nuemero al usuario para conocer sus inversas
numero = int(input("Ingresa un numero positivo para conocer sus inversas "))
print("")

#Declarar variables
f= numero
c= numero
matriz=[]
Total=0
#Con este for creamos la matriz del tamaño que el usuario necesite
for i in range(f):
    matriz.append([0]*c)
#Con estos for anidadados recorremos la matriz
for i in range(f):
    for j in range(c):
        Total=i*j%numero#realizamos la operacion multiplicando los iteradores y hacer la funcion mod con el numero ingresado
        matriz[i][j]=Total# y se guarda en esa posicion
        if(Total == 1):
            #print("Las inversas del numero ",numero," es ",i," * ",j,"mod",numero," es igual a",Total)
            print("los numeros ",i," * ",j, " son inversos multiplicativos a", numero )

print("\n")
print("*******************************************")
#print("")
df=pd.DataFrame(matriz)#Es para crear la matriz en un DataFrame
print("Grupo de unidades modulo",numero)
df #imprimir la matriz  en pandas
```

```
Ingresa un numero positivo para conocer sus inversas 12

los numeros  1  *  1  son inversos multiplicativos a 12
los numeros  5  *  5  son inversos multiplicativos a 12
los numeros  7  *  7  son inversos multiplicativos a 12
los numeros  11  *  11  son inversos multiplicativos a 12


*******************************************
Grupo de unidades modulo 12
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11
2	0	2	4	6	8	10	0	2	4	6	8	10
3	0	3	6	9	0	3	6	9	0	3	6	9
4	0	4	8	0	4	8	0	4	8	0	4	8
5	0	5	10	3	8	1	6	11	4	9	2	7
6	0	6	0	6	0	6	0	6	0	6	0	6
7	0	7	2	9	4	11	6	1	8	3	10	5
8	0	8	4	0	8	4	0	8	4	0	8	4
9	0	9	6	3	0	9	6	3	0	9	6	3
10	0	10	8	6	4	2	0	10	8	6	4	2
11	0	11	10	9	8	7	6	5	4	3	2	1

Figura 2.3: Encontrar inverso multiplicativo mediante fuerza bruta.

Como se muestra en la figura 2.3 el usuario ingresa un número y el programa multiplica con todos los números menores a él, y si el resultado de las operaciones es igual a 1 significa que son inversos multiplicativos pero este método funciona solo para números pequeños porque si utilizamos números muy grandes tardaría demasiado tiempo en

realizar las operaciones y la tabla que se imprime no sería capaz de mostrar la tabla completa.

Definición **2.6** *Para cada elemento a en un grupo finito G , tenemos un* $a^{|G|} = e$, *donde e es elemento neutral de G, por lo tanto tenemos para un numero* $a \in Z_p^*$, *con p primo.*

$$a^{\phi(p)} \equiv 1 \ mod \ (p)$$

Ya que Z_p^* es un grupo bajo la multiplicación donde $|Z_p^*| = p - 1 = \varphi(p)$

A este resultado se le llama el teorema pequeño de Fermat en el Teorema 2.55 en (Yan, Yung Rief, 2013).

$Z^*{}_p = \{g, g^2, \ldots, g^{p-1} = [1]\}$,para algún $g \in Z_{p-1}$, a la g seleccionada se le llamara un generador de $Z^*{}_p$. A los generadores también se les llaman raíces primitivas modulo p.

Ahora podemos introducir tres funciones que se pueden usar como funciones de trampa que usaremos en nuestro estudio de la criptografía.

2.2.1. Exponenciación discreta

Definición **2.7** *Sea p un número primo y g una raíz primitiva modulo p, definimos la exponencial discreta como:*

$$Exp_p : Z_{p-1} \longrightarrow Z^*{}_p,$$
$$x \longrightarrow g^x$$

Se puede ver que Exp_p es una función que preserva las operaciones, *i.e.*).

$$Exp_p(x + y) = Exp_p(x) * Exp_p(y).$$

Ya que:

$$Exp_p(x + y) = g^{(x+y)} = g^x * g^y = Exp_p(x) * Exp_p(y)$$

Además Exp_p es biyectiva, por lo tanto tiene una función inversa denotada por Log_p, esta función será llamada logaritmo discreto. Se puede ver que Exp_p se calcula fácilmente, por ejemplo por medio del algoritmo de *exponenciación rápida* que a continuación se describe véase en (Yan, Yung Rief, 2013).

La idea es que si el exponente e es una potencia de 2, digamos $e = 2^k$, entonces podemos exponenciar sucesivamente al cuadrado:

$$x^e = x^{2^k} = ((((\ldots(x^2)^2)^2\ldots)^2)^2)^2$$

De esta manera calculamos x^e por k cuadrados. Por ejemplo: $x^{16} = (((x^2)^2)^2)^2$

Si el exponente no es una potencia de 2, entonces usamos su representación binaria.

Suponga que e es un número de k bits, $2^{k1} \leq e < 2^k$. Entonces:

$$\begin{aligned} e &= 2^{k-1}e_{k-1} + 2^{k-2}e_{k-2} + \ldots + 2^1e_1 + 2^0e_0, \ \ (con\ e_{k-1} = 1) \\ &= (2^{k-2}e_{k-1}, 2^{k-3}e_{k-2} + \ldots + e_1) * 2 + e_0 \\ &= (\ldots((2e_{k-1} + e_{k-2}) * 2 + e_{k-3}) * 2 + \ldots + e_1) * 2 + e_0 \end{aligned}$$

Entonces:

$$\begin{aligned} x^e &= x^{(\ldots((2e_{k-1}+e_{k-2})*2+e_{k-3})*2+\ldots+e_1)*2+e_0} = \\ &= x^{(\ldots((2e_{k-1}+e_{k-2})*2+e_{k-3})*2+\ldots+e_1)^2} * x^{e_0} = \\ &(\ldots(((x^2 * x^{e_{k-2}})^2 * x^{e_{k-3}})^2 * \ldots)^2 * x^{e_1})^2 * x^{e_0} \end{aligned}$$

Vemos que x^e se puede calcular en $k-1$ pasos, cada paso consiste en cuadrar el resultado intermedio y, si el dígito binario correspondiente e_i de e es 1, una multiplicación adicional por x. Si queremos calcular la potencia modular $x^e\, mod n$, tomamos el resto del módulo n después de cada cuadratura y multiplicación:

$$x^e\, mod n = (\ldots(((x^2 * x^{e_{k-2}} mod\ n)^2 * x^{e_{k-3}}\ mod\ n)^2 \ldots)^2 * x^{e_1}\ mod\ n)^2 * x^{e_0}\ mod\ n$$

Teorema 2.2.1 *Sea p un primo. Entonces:*

$$x^2 \equiv 1\ mod\ p$$

Si solo si $x \equiv \pm 1\ mod\ p$ Demostración: véase el teorema 3.9 de (Yan, Yung Rief, 2013).

Definición **2.8** *A x se le llama raíz primitiva de 1 modulo n si $x^2 \equiv 1\ mod\ n$, pero $x \not\equiv \pm 1\ mod\ n$.*

2.2.2. Potencias modulares

Sea n el producto de dos primos distintos p y q y e un primo entero positivo a $\varphi(n)$, $e \equiv 1 mod \varphi(n)$

$$RSA_{n,e} : Z_n \longrightarrow Z_n, x \longrightarrow x^e$$

Se llama la función RSA, los valores de esta función se pueden calcular de manera eficiente. A la inversa es prácticamente inviable calcular la pre-imagen x de $y = x^e$, siempre que los factores p y q de n no se conozcan, donde n es decir RSA_n es una función unidireccional que depende de la dureza computacional de la factorización, no existe un algoritmo. Para calcular los factores de n, si son primos muy grandes, es posible hacer n y e públicos y al mismo tiempo mantener p y q en secreto.

La función RSA es biyectiva, si se conoce los números primos de n se puede invertir fácilmente, demostración véase proposición 3.3 (Delfs Knebl, 2015).

2.2.3. Cuadrados modulares

Definición **2.9** *Sea p y q primos distintos y $n = p * q$*

$$Cuadrado : Z_n \longrightarrow Z_n, x \longrightarrow x^2$$

se llama la función cuadrada. Cada elemento $y \neq 0$ tiene $0, 2$ o 4 pre-imágenes. Si tanto como p y q son números primos $\equiv 3$ *mod* 4 entonces -1 no es un cuadrado modulo p y modulo q.

Si se conocen los factores de n, las imágenes previas de cuadrado (denominadas raíces cuadradas) son computables de manera eficiente.

Capítulo 3

RSA

3.1. El algoritmo RSA

El algoritmo RSA, que de acuerdo con varios autores (Aumasson, 2018; Talbot Welsh, 2006) es el algoritmo de clave pública más utilizado en las aplicaciones, que fue inventado por Rivest,shamir y Adleman en 1977, se ha convertido en la base de toda una generación de productos en tecnología de seguridad.

Utiliza la función trampilla unidireccional $f : m \longrightarrow c$ que es fácil calcular, pero su inversa $f^{-1} : C \longrightarrow M$ es computacionalmente difícil para aquellos que no conocen la clave privada d, tendrá que calcular n y calcular $\varphi(n)$ para encontrar d.

RSA es un sistema de cifrado que se basa en hechos de la teoría numérica, y para eso tenemos que multiplicar dos números primos muy grandes y hacer que su producto n sea parte de la clave pública, mientras que los factores de n se mantengan en secreto y se usan en la clave secreta. La seguridad de RSA depende de la dificultad de factorizar n.

A continuación éste se describe a través de un ejemplo:

Dos personas quieren comunicarse a través de un mensaje, y quieren ocultar el contenido de un atacante externo. Para esto, el receptor generas unas claves, una pública que es enviada al emisor ésta puede ser conocida por el atacante y la clave privada que solo la conoce el receptor. El emisor cifra el mensaje con la clave pública y la envía por un canal inseguro de tal suerte que cualquiera puede ver el mensaje cifrado, pero no se puede obtener el mensaje original sin conocer su clave privada. El receptor descifra el mensaje con su clave privada. Y con ayuda de la figura 4 se explicará como sucede todo esto.

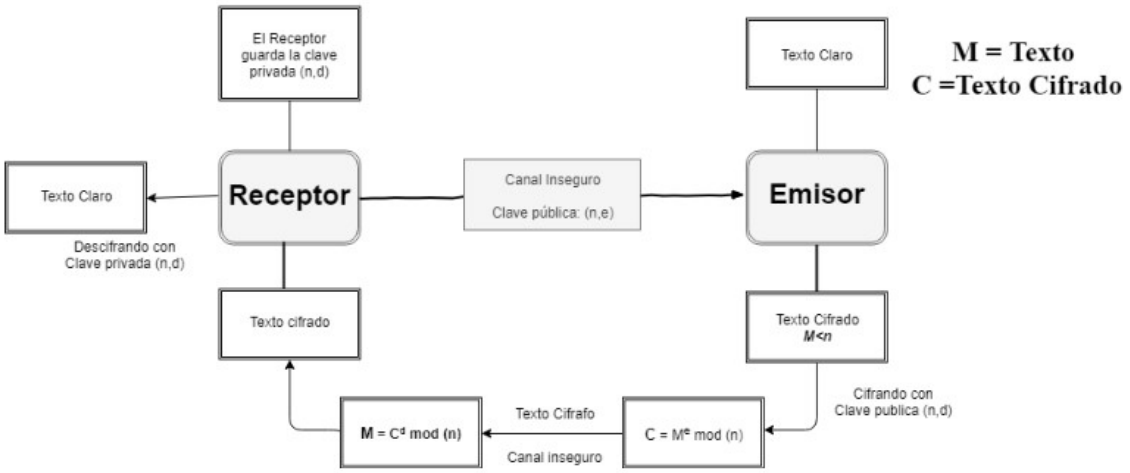


Figura 3.1: Diagrama RSA

EL cifrado RSA necesita lo siguiente.

Lista 1:

1. Que el Receptor genere las claves, para eso se seleccionan dos números primos p, q muy grandes y obtiene $n = p * q$.
2. Se escoge un $e \in Z^+{}_n$ tal que $1 < e < \varphi(n)$.
3. Se calcula d que es el inverso multiplicativo de $e\ mod\ \varphi(n)$, i.e) $ed1\ mod\ \varphi(n)$.
4. Ahora el Emisor puede cifrar el mensaje de la siguiente manera $C = M^e mod(n)$.
5. Para que Receptor descifre el mensaje realiza lo siguiente $C^d = M^{ed} mod(n)$, en el punto 3 de esta lista se asegura que $ed \equiv 1\ mod\ \varphi(n)$ no es inmediato que $ed \equiv 1\ mod\ (n)$ sin embargo en la proposición 3.3.1 (Stein, 2009) se asegura que esto es suficiente para que $M = M^{ed} mod(n)$. Y así recuperar el mensaje original.

A continuación se mostraran dos corridas del sistema de cifrado RSA que se encuentran en el libro cracking (Sweigart, 2018).

Lista 2:

a) En la primera corrida muestra como un usuario ingresa un mensaje y es cifrado, para esto se generan las claves de las que se hablan en los puntos 1, 2, 3 de la lista anterior. Al final de todo el proceso muestra como es el mensaje cifrado.

b) En la segunda corrida el usuario ingresa el archivo con texto cifrado y el sistema produce texto original.

c) Por otro lado el usuario puede usar las claves que generó anteriormente o generar nuevas claves y pasa al punto b.

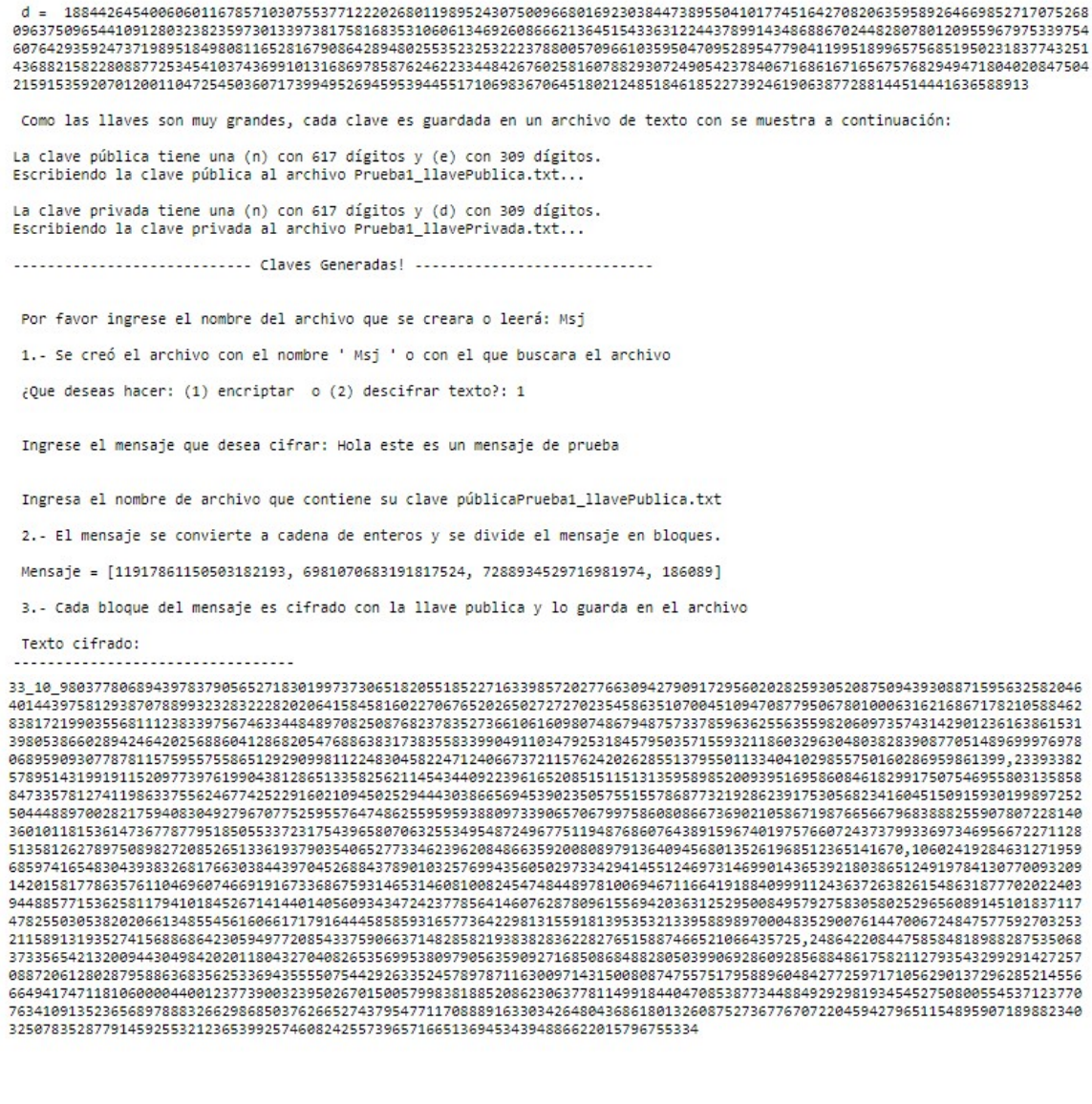
d) Las claves son guardadas en archivos texto que fueron nombrados por el usuario porque en general las claves son números muy grandes.

e) El programa tiene 315 líneas de código por lo que no se muestra, para poder correrlo el código debe de estar en el directorio de trabajo.

Para esta investigación se trabajó en Jupyter con el lenguaje de Python3 por lo que para ejecutar el programa basta con llamarlo de una celda como se muestra a continuación en la figura 3.2:

```
1 run SistemaRSA.py
```

```
Sistema de cifrado RSA

Para cifrar y descifrar un mensaje necesitamos de dos claves una privada y una publica
si es la primera vez que entras al sistema, genera las claves y si no puedes utilizar las antes creadas
Quieres generar las claves, (1) Si o (2) No: 1

 Ingrese el nombre para crear sus claves : Prueba1

 Genera dos números primos...

 Genera p =  17426916294627874704859066445272164432687323316793572823065366745446215396951584207534416054518203448811785497950
749473703652650056150481232559475642484324264603289754910939905128522033760688994237613215651298748292348433792338423692659893
914348069721477042910790433588521072518640868873404023079248764739107 9

 Genera q =  14779378722139091364434268304578042222699848842773661663506641297616757373572033090839130932640720543833113846678
675040386367495972065876987114192366729314098127960436936816585809463779923128966137757816201487270786430236996122281643357083
447693509838844843375239976089463244693363619542583373739303695590453 3

 Genera n = p*q:  257558995877322227886377690988451054383938322113182928048084088045881788020048453664206225499326709436883210
783657866086682389764003632791242950027629110384075295609810427049765089002375748064130087084740329301734386088169841946692364
774390420510594845038593365062864235621192603261616724950504405786459292329742684714086383024320660932551737980550108679667072
385424876985563191341064407262584170748911095151070391207729216745849264484737468104791761299094055309476420877430969197303936
933099785786860724656316771875271036479087422576847274518956200204271763337285848153500778679212579485241306649173476134820398
61107

 Crea un número e que sea primo relativo a (p-1)*(q-1)...

 e =  152870815601278557825658759794120053509798844241167870056383739269039540278142546847984272152204455740965042314247991159
799474936593116680225579768736990878000215018250105091095588316743367166888136692235845432890111711434024446897004908504288371
571944168048214144366657351893127870364938065779099737993201289

 Calcula d que es modulo inverso de e...
```



Como las llaves son muy grandes, cada clave es guardada en un archivo de texto con se muestra a continuación:

```
La clave pública tiene una (n) con 617 dígitos y (e) con 309 dígitos.
Escribiendo la clave pública al archivo Prueba1_llavePublica.txt...

La clave privada tiene una (n) con 617 dígitos y (d) con 309 dígitos.
Escribiendo la clave privada al archivo Prueba1_llavePrivada.txt...

---------------------------- Claves Generadas! ----------------------------
```

```
Por favor ingrese el nombre del archivo que se creara o leerá: Msj

1.- Se creó el archivo con el nombre ' Msj ' o con el que buscara el archivo

¿Que deseas hacer: (1) encriptar  o (2) descifrar texto?: 1


Ingrese el mensaje que desea cifrar: Hola este es un mensaje de prueba


Ingresa el nombre de archivo que contiene su clave públicaPrueba1_llavePublica.txt

2.- El mensaje se convierte a cadena de enteros y se divide el mensaje en bloques.

Mensaje = [11917861150503182193, 69810706831918175 24, 7288934529716981974, 186089]

3.- Cada bloque del mensaje es cifrado con la llave publica y lo guarda en el archivo

Texto cifrado:
----------------------------------
```

```
33_10_9803778068943978379056527183019973730651820551852271633985720277663094279091729560202825930520875094393088715956325820464014439758129387078899323283222820206415845816022706765202650272727023545863510700451094708779506780100063162168671782105884628381721990355681112383397567463344848970825087682378352736610616098074867948757337859636255635598206097357431429012361638615313980538660289424642025688604128682054768863831738355833990491103479253184579503571559321186032963048038283908770514896999769786895909307787811575955755865129290998112248304582247124066737211576242026285513795501133404102985575016028659861399,23393382578951431991911520977397619904381286513358256211454344092239616520851511513135958985200939516958608461829917507546955803135858847335781274119863375562467742522916021094502529444303866569453902350575515578687732192862391753056823416045150915930199897252504448897002821759408304929767077525955764748625595959388097339065706799758608086673690210586719876655667968388825559078072281403601011815361473677877951850553372317543965807063255349548724967751194876860764389159674019757660724373799336973469566721711285135812627897508982720852651336193790354065277334623962084866359200808979136409456801352619685123651416701060241928463127195968597416548304393832681766303844397045268843789010325769943560502973342941455124697314699014365392180386512491978413077009320914201581778635761104696074669191673368675931465314608100824547484489781006946711664191884099911243637263826154863187770202240394488577153625811794101845267141440140560934347242377856414607628780961556942036312529500849579275830580252965608914510183711747825503053820206613485545616066171791644458585931657736422981315591813953532133958898970004835290076144700672484757759270325321158913193527415688686423059497720854337590663714828582193838283622827651588746652106643572524864220844758584818988287535068373356542132009443049842020118043270408265356995380979056359092716850868488280503990692860928568848617582112793543299291427257088720612802879588636835625336943555507544292633524578978711630097143150080874755751795889604842772597171056290137296285214556664941747118106000044001237739003239502670150057998381885208623063778114991844047085387734488492929819345452750800554537123770763410913523656897888326629868503762665274379547711708889163303426480436861801326087527367767072204594279651154895907189882340325078352877914592553212365399257460824255739657166513694534394886622015796755334
```

Figura 3.2: Cifrando un mensaje con el sistema RSA

En la imagen anterior se muestra como es cifrado un mensaje, pero necesita que un usuario ingrese el nombre del archivo con el que se guardara o con el que buscara el archivo para descifrarlo, posteriormente selecciona si cifrara o descifrara el mensaje, si elige cifrar tiene que ingresar el mensaje junto con la clave pública, y el algoritmo muestra el texto cifrado, como se puede observar en el texto cifrado aparecen dos números separados por un guión bajo. El número 33 define el tamaño del mensaje claro. El número 10 es el tamaño de cada bloque, por otro lado si selecciono descifrar el usuario solo tendrá que ingresar la clave privada y mostrara el mensaje descifrado como se mostrara en la figura 3.3.

```
1 run SistemaRSA.py

Sistema de cifrado RSA

Para cifrar y descifrar un mensaje necesitamos de dos claves una privada y una publica
si es la primera vez que entras al sistema, genera las claves y si no puedes utilizar las antes creadas
Quieres generar las claves, (1) Si o (2) No: 2

 Por favor ingrese el nombre del archivo que se creara o leera: Msj

 1.- Se creó el archivo con el nombre ' Msj ' o con el que buscara el archivo

 ¿Que deseas hacer encriptar (1) o descifrar (2) texto?: 2

 Ingresa el nombre de archivo que contiene su clave privadaPrueba1_llavePrivada.txt

 2.- Descifra la lista de bloques encriptados del archivo  Msj  con la llave privada

 3.- El mensaje dividio se convierte a cadena de enteros y muestra el mensaje

 Texto descifrado:
Hola este es un mensaje de prueba
```

Figura 3.3: Descifrando un mensaje con el sistema RSA

3.2. Análisis del algoritmo RSA

En la siguiente sección se analizaran los algoritmos fundamentales de la criptografía que no son eficientes y los que si son eficientes para el algoritmo RSA que son:

I Programa que genera números muy grandes y prueba si son primos, este programa es utilizado por RSA para generar parte de la clave pública n.

II Programa de exponenciación rápida modulo n, Se usa para obtener las claves en los puntos 4, 5 de la lista 1, a partir de la descripción en libro de (Beissinger Pless, 2006).

III Programa para calcular el inverso multiplicativo d de e modulo $\varphi(n)$, es la clave privada que se utiliza para descifrar el mensaje ($M = C^{ed}\ mod\ (n)$).

Para analizar estos programas se desarrollados los algoritmos principales de la criptología como, dado un número p decidir si es primo o no, dado $e \in Z_n$ tal que e es primo relativo con n encontrar el inverso multiplicativo en Z_n y exponenciación.

3.2.1. Algoritmo para generar primos

Para el programa I de la lista anterior se desarrollaron dos algoritmos para generar números y probar si son primos, el primer algoritmo *primo* no es muy eficiente porque tardaría demasiado tiempo con un número muy grande en decir si un número es primo, porque va dividiendo el número, por todos los números hasta él.

A continuación mostraremos el código y la corrida del algoritmo *primo*.

```python
#Sabes si un numero es primo no tan eficiente
import time
#La Funcion se llama primo y recibe un numero
def primo(numero):
    for i in range(2,numero): #Hace un ciclo del 2 hasta el numero
        if (numero%i)==0: #Si es divisible por uno de esos numeros no es primo
            return False #Regresa falso si es divisible
    return True # Regresa True si no es divisible por un numero

numero = int(input("Ingresa un numero para saber si es primo: ")) # el usuario ingresa el numero
inicio_de_tiempo = time.time()
if primo(numero):
    print ("\nEl numero %s es primo" % numero)
else:
    print ("\nEl numero %s NO es primo" % numero)

tiempo_final = time.time()
tiempo_transcurrido = tiempo_final  - inicio_de_tiempo
print ("\nTomo %d segundos." % (tiempo_transcurrido))
```

```
Ingresa un numero para saber si es primo: 165660017

El numero 165660017 es primo

Tomo 65 segundos.
```

Figura 3.4: Código conocer numero primo

En la figura 3.4 anterior se encuentra el código y un ejemplo de la función *primo* que lo único que necesita es que un usuario ingrese el número y el programa te dice si es un numero primo o no. pero con un número mayor a 9 dígitos tarda 65 segundos y para el algoritmo RSA seria ineficiente porque utiliza números de aproximadamente 300 dígitos.

Para eso se implementó el método **RM** que genera números muy grandes, hasta que Rabin-Miller prueba que es un número primo. Rabin-Miller constituye una prueba de primalidad probabilística rápida y práctica (Yan, Yung Rief, 2013), el resultado no es necesariamente correcto para todos los casos sin embargo la probabilidad de error es pequeña.

En la imagen 3.5 se puede observar el código de **RM** y una corrida con un numero de 1024 bits y el tiempo en que tarda en correrlo.

```python
import random, sys, time
def RM(numero):
    #print("Numero: " +str(numero))
    if numero % 2 == 0 or numero < 2:
        return False
    if numero == 3:
        return True
        p=True

    menos = numero -1
    veces = 0


    while menos % 2 == 0:
        menos = menos// 2
        veces += 1

    for checa in range(5):
        NewNum = random.randrange(2,numero-1)
        expRap = pow(NewNum,menos,numero)
        if expRap != 1:
            i = 0
            while expRap != (numero-1):
                if i == veces-1:

                    return False
                else:
                    i = i + 1
                    expRap = (expRap ** 2)%numero
    return True
    p = True

p=False
tamañoClave= int(input("Ingresa el tamaño de la clave: "))
inicio_de_tiempo = time.time()
while p== False:
    numero = random.randrange(2**(tamañoClave-1), 2**(tamañoClave))
    if RM(numero):
        p= True
        print("Total de digitos: ",len(str(numero)))
        print("Numero Primo: " +str(numero))

tiempo_final = time.time()
tiempo_transcurrido = tiempo_final  - inicio_de_tiempo
print ("\nTomó %d segundos." % (tiempo_transcurrido))
```

```
Ingresa el tamaño de la clave: 1024
Total de digitos:  309
Numero Primo: 1400733652825419762474003694162887635327486030976190928530907521481831698046690566617708748820870615115276497101
4051272564313745170189479764155986688865608496277586024721152284032855842178316957880495107489427511369823188060342956772075132
2268133930875669738208332113388044725937623847263404490749405607053817 81

Tomó 2 segundos.
```

Figura 3.5: Algoritmo de primalidad probabilístico

El programa anterior, genero un número de 309 dígitos y solo tardo al aproximadamente 2 segundos. El sistema RSA corre este programa dos veces en la primera genera p y la segunda vez genera q. La función RM en la línea 4 determina que no sea un número entero. En la línea 10 al número le resta 1 y empieza a reducir hasta que sea impar. Posteriormente prueba con 5 números aleatorios la primalidad.

3.2.2. Exponenciación Modular

La exponenciación es necesaria en el paso cuatro de la lista 1, que necesita potenciación de números y para ello se implementaron dos algoritmos, pero uno de ellos no es eficiente porque el sistema RSA eleva números muy grandes de aproximadamente 300 dígitos pero con este algoritmo sería demasiado tardado porque va elevando valor por valor.

En la Figura 3.6 imagen se mostrara dicho código, con una corrida para demostrar el tiempo que tarda en exponenciar con números no muy grandes.

```
1  #Algoritmo de exponenciacion
2  import sys,time
3  a= int(input("Ingresa la base :"))
4  b= int(input("Ingresa el numero exponente :"))
5  mod = int(input("Ingresa el modulo :"))
6  inicio_de_tiempo = time.time()
7  for x in range(b+1):
8      re=a**x
9      t=re%mod
10
11     #print(re)
12 print("EL Resultado es:"+str(a)+" ** "+str(b)+"mod : " +str(mod)+ " = " +str(t))
13 tiempo_final = time.time()
14 tiempo_transcurrido = tiempo_final  - inicio_de_tiempo
15 print ("\nTomo %d segundos." % (tiempo_transcurrido))
```

```
Ingresa la base :35344
Ingresa el numero exponente :34525
Ingresa el modulo :4323
EL Resultado es:35344 ** 34525mod : 4323 = 1486

Tomo 610 segundos.
```

Figura 3.6: Algoritmo de exponenciación

Como se puede observar en la figura anterior es un método ineficiente porque con un número de 5 dígitos el sistema Tarda aproximadamente 610 segundos y para el sistema RSA que utiliza números muy grandes para exponenciar tiene que ser de una manera casi instantánea entonces el algoritmo anterior no serviría para el sistema.

Entonces se necesita un algoritmo para calcular de forma rápida grandes potencias enteras de un número x modulo n. Como en la figura 3.7 se puede apreciar el código de *exp* guiado del pseudocódigo en (Yan, Yung Rief, 2013). Los pasos que realiza el código *exp* son los siguientes:

- Lo primero que hace transforma el exponente a un número en binario.
- Crea una variable x que es igual a la base como se ve en la línea 11.
- Después el programa recorre cada uno de los bits que genero de izquierda a derecha.
- El programa entra en un ciclo para conocer si el bit es igual a 0 o 1.
- Si el bit es igual a 0 entonces calcula: $x = x^2 \; mod \; (n)$.
- Pero si el bits es 1 entonces $x = (x^2 * a) \; mod \; (n)$, donde a es la base ingresada por el usuario, según la orden en la línea 4.

```python
#### Algoritmo de exponenciacion Rapida
import sys,time
print("\n Algortimo de exponenciación rápida")
a= int(input("a = Ingresa la base :"))
b= int(input("b = Ingresa el numero exponente :"))
n= int(input("n = Ingresa el modulo n :"))
inicio_de_tiempo = time.time()
def exp(x, y,n):
    exp = bin(y)
    print("(1).- El exponente se pasa a binario$")
    value = x
    print("(2).- Si el bit es 0 entonces = X = (x* 2 modulo (n)")
    print("(3).- Si el bit es 1 entonces = X = (x**2)*a modulo (n)")
    print("(4).- Entra en un ciclo hasta el ultimo bit")
    print("\n")
    print ("el numero en binario es:",exp[2:])
    print ("Bit\tResultado")

    print(1,":\t","x = ",value)
    for i in range(3, len(exp)):
        if(exp[i:i+1]=='0'):
            o=value
            value = (value * value)%n
            z = (o * o)
            t=z%n
            print(i-1,":\t","x = ",o,"**2","mod",n ,"=",z,"mod",n ,"=",value)
        else:
            if(exp[i:i+1]=='1'):
                e=value
                value=((value ** 2)*x)%n
                q=(e**2)
                q1=(q*x)
                q3=(q1%n)
                #value = (value*x)%mod
                print (i-1,":\t","x = (",e,"**2)*",x,"mod",n ,"=",q1,"mod",n ,"=",value)
    return value

print ("\n Calculamos a^b mod (n)")
print ("a=",a)
print ("b=",b)
print ("n=",n)
print ("================ Pasos ================")
res=exp(a,b,n)
tiempo_final = time.time()
tiempo_transcurrido = tiempo_final  - inicio_de_tiempo
print ("Resultado:",res, "y Tomó %d segundos." % (tiempo_transcurrido))
```

```
 Algortimo de exponenciación rápida
a = Ingresa la base :76234
b = Ingresa el numero exponente :8734
n = Ingresa el modulo n :34224

 Calculamos a^b mod (n)
a= 76234
b= 8734
n= 34224
================ Pasos ================
(1).- El exponente se pasa a binario$
(2).- Si el bit es 0 entonces = X = (x* 2 modulo (n)
(3).- Si el bit es 1 entonces = X = (x**2)*a modulo (n)
(4).- Entra en un ciclo hasta el ultimo bit


el numero en binario es: 10001000011110
Bit     Resultado
1 :      x =  76234
2 :      x =  76234 **2 mod 34224 = 5811622756 mod 34224 = 11092
3 :      x =  11092 **2 mod 34224 = 123032464 mod 34224 = 31408
4 :      x =  31408 **2 mod 34224 = 986462464 mod 34224 = 24112
5 :      x = ( 24112 **2)* 76234 mod 34224 = 44321574263296 mod 34224 = 9232
6 :      x =  9232 **2 mod 34224 = 85229824 mod 34224 = 12064
7 :      x =  12064 **2 mod 34224 = 145540096 mod 34224 = 19648
8 :      x =  19648 **2 mod 34224 = 386043904 mod 34224 = 31408
9 :      x =  31408 **2 mod 34224 = 986462464 mod 34224 = 24112
10 :     x = ( 24112 **2)* 76234 mod 34224 = 44321574263296 mod 34224 = 9232
11 :     x = ( 9232 **2)* 76234 mod 34224 = 6497410402816 mod 34224 = 19648
12 :     x = ( 19648 **2)* 76234 mod 34224 = 29429670977536 mod 34224 = 12208
13 :     x = ( 12208 **2)* 76234 mod 34224 = 11361554315776 mod 34224 = 25600
14 :     x =  25600 **2 mod 34224 = 655360000 mod 34224 = 4624
Resultado: 4624 y Tomó 0 segundos.
```

Figura 3.7: Algoritmo de exponenciación Rápida

En el algoritmo anterior se ha ejemplificado el algoritmo de exponenciación rápida. Donde se utilizaron números pequeños para simplificar el procedimiento. Aunque generalmente el usuario necesita ingresar números muy grandes para la base, el exponente y el modulo, el código del algoritmo sigue funcionando de manera eficiente. En estos casos el despliegue del algoritmo se puede volver confuso para el lector, por lo

que se recomienda quitar las impresiones de cada paso para solo observar el resultado. Por ejemplo:

```
1  run exp.py

 Algortimo de exponenciación rápida
b = Ingresa la base 873895348953885345838495478487958793893874583489594563463546746734867456738424378956848974576 45
e = Ingresa el número exponente 845748699847893468457634856893458963467039434956980345906346845674586740698458934654906849306 0
4683468945690345698045896984860934960934569854096845869045684509698450963456876787800000084857477348684368354684963363
n = Ingresa el modulo n823434837838594930549543539049549385900498834389053890389534950489408958940589589059805890509830984598 0
3459804598098098089038903890348904890389034089456706576075683 0

 Calculamos b^e mod (n)
================= Pasos =================
(1).- El exponente se pasa a binario$
(2).- Si el bit es 0 entonces = X = (x* 2 modulo (n)
(3).- Si el bit es 1 entonces = X = (x**2)*a modulo (n)
(4).- Entra en un ciclo hasta el ultimo bit

 El número en binario es: 1000000010100000101011100010101111001000000011001000110011011010010101011110000001100101001000110010
0001011110110111001110111111100111000111001111011110010011001011100110000100011001100000110010010010010110001000000001000001100
1011001101101010111011001000001100111100000000110011100000010101100110001110010111011011000011110010010000110011010100110100100 1
1100100101001001110101110101101011000110011101101100101111011111000000111101100010101000100111111001010011100000011101111011010
0001000111001001111000101110010000010010010001001110101100100110011110000110110100100010001001111010101000100101110110111000 11
100000001011111001001000110100111000110011111011000010001101000110111111110011001010000000101000100011

 b^e mod (n) : 75500617519673630285112430214810758952514606677153991945502529149103291503995721991023834604791839555799545592 2
2532262481797605779776532190243295169877185211306981 35
=============================================

Tomó 0 segundos.
```

Figura 3.8: Algoritmo de exponenciación rápida sin pasos

Como se había dicho el programa de exponenciación rápida para números muy grandes, pediría al usuario los números para la exponenciación, mostraría los pasos que realizara internamente, el número exponente en número binario y mostraría el resultado final de las operaciones que hizo el programa.

3.2.3. Inverso Multiplicativo

En la criptología es muy importante conocer el inverso multiplicativo de e en Z_n, que forma parte de la clave privada para nuestro sistema de seguridad RSA.

Euclides era un matemático muy importante que creo el algoritmo para encontrar el Máximo Común Divisor de dos números, a partir del cual nace el algoritmo extendido de Euclides, que encuentra el inverso multiplicativo de dos números, solo cuando son primos relativos existe un inverso a ellos. Para finalizar se analizó el punto III de la lista 3, que es encontrar el inverso multiplicativo de un número e modulo n que es otro entero d modulo n tal que el producto $ed \equiv 1 \; mod \; (n)$ como se define en el paso cinco de la lista 1.

Para eso se desarrollaron dos algoritmos para encontrar el inverso multiplicativo de un número, el primero no es muy eficiente ya que con números muy grandes es muy tardado y el segundo que con un número muy grande lo hace de una manera muy rápida.

En la figura 3.9 mostraremos el código del algoritmo que con un numero de 4 dígitos tarda demasiado y para el sistema RSA necesita de un algoritmo más eficiente para

que el sistema funcione.

El algoritmo funciona de la siguiente manera:

- Necesita que un usuario ingrese el número para conocer sus inversas y el modulo.
- Después determina si su máximo común divisor es uno con la librería math utilizando la función math.gcd.
- Entonces entra en un ciclo y va elevando hasta que el resultado de la operación sea igual a 1 y imprime el número que da como resultado 1.

```
import math, time

n = int(input("Ingresa un numero positivo para determinar el modulo (n=(p-1)*(q-1)):"))
x =  int(input("Ingresa el numero para conocer su inversa: "))
inicio_de_tiempo = time.time()
if math.gcd(x,n) == 1:
    for j in range(2,n):
        R=x*j%n
        if R == 1:
            print("La inversa del numero ",x, " es: ",j)
else:
    print("El numero no tiene inverso multiplicativo")
tiempo_final = time.time()
tiempo_transcurrido = tiempo_final  - inicio_de_tiempo
print ("\nTomo %d segundos." % (tiempo_transcurrido))
```

```
Ingresa un numero positivo para determinar el modulo (n=(p-1)*(q-1)):7874400
Ingresa el numero para conocer su inversa: 2389
La inversa del numero  2389  es:  7782109

Tomo 5 segundos.
```

Figura 3.9: Algoritmo para conocer inversa

Existen diversas formas de encontrar el inverso multiplicativo con el algoritmo de Euclides, en la primera forma su desarrollo es poco práctico y puede tener equivocaciones, como se demuestra de manera escrita con los números $(49, 89)$ y se realiza lo siguiente:

I. Prueba que sean primos relativos, para obtener su inverso multiplicativo.

II. Posteriormente se aplica el algoritmo de Euclides extendido.

III. En este paso ordenada cada ecuación por sus resto hasta que sea igual a 1.

IV. Reordenados por sus restos para encontrar el inverso multiplicativo (a, n).

V. Se expresa el MCD de $(a, n) = 1$ como la mínima combinación lineal de esos números: $1 = c1 * a + c * n$.

Lo primero es realizar los puntos uno, dos y tres de la lista anterior que demostraremos con un algoritmo implementado como se muestra en la figura 3.10.

```
1 run ExpresionEnNumeros.py

Introduce el primer numero: 48
Introduce el segundo numero: 89

 Algortimo de Euclides extendido
89 = 1 * 48 + 41
48 = 1 * 41 + 7
41 = 5 * 7 + 6
7 = 1 * 6 + 1
6 = 6 * 1 + 0
MCD is 1

 Ordenando cada ecuación por sus restos hasta su resto igual a 1
41 = 89 - 1 * 48
7 = 48 - 1 * 41
6 = 41 - 5 * 7
1 = 7 - 1 * 6

Tomo 0 segundos.
```

Figura 3.10: Algoritmo de Euclides y reordenando por restos

Posteriormente se realiza el punto cinco y se obtiene lo siguiente:

$33 = 82 - 1 * 49$

$16 = 49 - 1 * 33 = 49 - 1 * (82 - 1 * 49) = 2 * 49 - 82$

$1 = (82 - 1 * 49) - 2 * (2 * 49 - 82) = -5 * 49 + 3 * 82$

Resolviendo la siguiente ecuación se obtendrá: $-5 * 49 + 3 * 82 = 1$

Entonces $1 = (-5 * 49 + 3 * 82)\ mod\ 82 = -5 * 49\ mod\ 82$

Pero como $-5\ mod\ 82 = 77$ se obtiene que 77 es inverso porque $77 * 49\ mod\ 82 = 1$

El método anterior se puede realizar de una manera sencilla y directa pero si se utilizan números muy grandes sería un proceso muy tardado para realizar, por lo cual se implementó la función inverso que demostramos en la figura 3.11 a continuación:

```python
import random, sys, os, time
def MCD(a, b):
    # Devuelve su MCD  de a y b usando el algoritmo de Euclides extendido
    while a != 0:
        (a, b) =(b % a, a)
    return b

def Inverso(a, m):
    # Devuelve el inverso modular de a% m, que es
    # el número x tal que a * x% m = 1

    if MCD(a, m) != 1:
        print("No son primos relativos")
        return None #si a y m no son primos

    # Calcular utilizando el algoritmo euclidiano extendido:

    u1, u2, u3 = 1, 0, a
    v1, v2, v3 = 0, 1, m

    while v3 != 0:
        D = u3 // v3 # // es el operador de división entera
        v1,u1=(u1 - D * v1),v1
        v2,u2=(u2 - D * v2),v2
        v3,u3=(u3 - D * v3),v3

    t=u1%m
    print("\n El inverso multiplicativo del número es",t)

if __name__ == '__main__':
    x =int(input("\n Ingresa el numero para conocer su inversa: "))
    n =int(input("\n Ingresa un numero para el modulo: "))
    inicio_de_tiempo = time.time()
    Inverso(x,n)
    tiempo_final = time.time()
    tiempo_transcurrido = tiempo_final  - inicio_de_tiempo
    print ("Tomó %d segundos." % (tiempo_transcurrido))
```

```
 Ingresa el numero para conocer su inversa: 135131525645211743591665718454653857227844974968528477162727090705051730012740430986203553256464409469342413657495117709651882737642738886968159424803152480227044045652038128097124098791956217648302318603297570101218788268102936093390123709156863060274382253220183125950084004981051071837383020685956665093093

 Ingresa un numero para el modulo: 11893802174143680837545533052555222834992506053051101278698282711402330792446417498206214504836214023723838455736521031652081713829484738607705959684645974464797929370912634611182792228471711926938982030292186327480534631303690584830988305966049877507591151633173978579880291764870363899669649324766509790146378482627293986682549171211094666066199645394587544301765655745183223904678740469092771753482889290784404547742009121294187148973181497098545794233445103940922597992797236549372424052987087615540375800113619309697918390005752947296539344668417099014963633901921744279396831418033931656567861670332479578181664

 El inverso multiplicativo del número es 946344114016584719440941559304656873726125321762914344059569177254750064374839201972199447206590644397312542552386869747942620032512618505350115339203069337030043467159695732437855037689489824018525868798584807408126951068632050207209145341351962452104024526789533946153856338854136473675405662070655303060444509823564363201707255956151455838382946293813382812948564378293309222617220662738062924490756377808859220316628855828512177263913936213350790604053150407263374333024670232015951521131305340735994470061482119373031708353659328625848196589223210108530406773668781192273936184203084828391185041561219868904589
Tomó 0 segundos.
```

Figura 3.11: Algoritmo inverso multiplicativo eficiente

EL programa anterior es sencillo y además eficiente porque encuentra el inverso multiplicativo en muy pocos pasos. Inicia de la misma manera que el método anterior conociendo si son primos relativos como se muestra en la línea 13, en la líneas 19 y 20 son declaradas las variables que utiliza el programa, posteriormente podemos observar que realiza un operación de división entera en la línea 23 y finalmente realiza operaciones sencillas de la línea 24 a la 26, hasta que $v3 = 0$.

3.3. Ataques al RSA

Cuando se desea implementar un sistema de seguridad RSA, existen muchas ideas adicionales, que hacen para un atacante más difícil romperlo, por ejemplo:

Dentro del sistema puedes tener una combinación de símbolos diferente a la de los demás sistemas o incluso tener una combinación diferente de símbolos para cada mensaje.

El sistema RSA es de uso común en OpenSSH, es un protocolo que facilita las comunicaciones seguras entre dos sistemas, El protocolo cifra toda la información y utiliza tres métodos de cifrado:

- Cifrado simétrico.
- Cifrado Asimétrico.
- Hash

A pesar de esta composición del sistema, la seguridad sigue dependiendo en gran medida de la capacidad que se tenga para poder factorizar un número en sus factores primos. .[El] tamaño del número que se puede factorizar aumenta exponencialmente año tras año. Este hecho se debe en parte a los avances en el hardware de computación, y en gran medida a los avances en los algoritmos de factorización."(Case, 2003), se ha realizado mucha investigación para encontrar diferentes formas de factorizar números rápidamente. El algoritmo tamiz de campo numérico general (General Number Field Sieve) es el algoritmo mas eficiente para factorizar en primos un número n según (Case, 2003). En la pagina (Bakogiannis Karapanos, 2019) se describe una implementación de $c++$ este algoritmo y se da un ejemplo de la factorización de un número de 60 dígitos en factores primos, En 1998 Briggs menciona que utilizando este algoritmo se factorizo un numero de 130 dígitos, en 2009 se pudo factorizar un número de 232 dígitos, el algoritmo tardo dos años para factorizar el número (Kleinjung, Aoki, Franke, Lenstra,..., 2010). Para la interfaz del sistema RSA que se implemento tiene una capacidad de generar claves publicas (n) de más de 900 dígitos.

Si se conocen las claves (n, e) y podemos factorizar n en primos p y q, podemos encontrar la clave de descifrado d de la siguiente manera:

1. Sabemos que $n = pq$, por lo que $\varphi(n) = (p-1)(q-1)$.
2. Resolvemos $ed \equiv 1(mod(p-1)(q-1))$.

Resolución de la ecuación 2 de la lista no es muy difícil véase el algoritmo figura 3.11.

Esto quiere decir que si podemos factorizar n es posible romper la seguridad del sistema RSA.

A continuación se explicaran otros métodos para poder factorizar n:

3.3.1. Factorizando n dado φ(n)

Supongamos que n = pq. Dado $\varphi(n)$, es muy fácil calcular p y q. Tenemos:

$$\varphi(n) = (p1)(q1) = pq(p+q) + 1,$$

Así sabemos que tanto $pq = n$ como $p+q = n+1-\varphi(n)$. Así, conocemos el polinomio.

$$x^2(p+q)x + pq = (xp)(xq)$$

cuyas raíces son p y q. Estas raíces se pueden encontrar utilizando la fórmula cuadrática.

Ejemplo: El número $n = pq = 35011$ es un producto de dos primos, y $\varphi(n) = 34632$. Tenemos:

$$\begin{aligned} f &= x^2(n+1\varphi(n))x + n \\ &= x^2 - 380x + 35011 \\ &= (x157)(x223) \end{aligned}$$

La factorización se logra con la fórmula cuadrática:

$$\frac{-b + \sqrt{b^2 - 4ac}}{2}$$

$$\frac{380 + \sqrt{380^2 - 35011}}{2} = 223$$

$$\frac{380 - \sqrt{380^2 - 35011}}{2} = 157$$

Entonces encontramos que $n = 157 * 223 = 35011$

3.3.2. Cuando p y q están cerca

Otra forma de poder factorizar n es cuando p y q están cerca el uno del otro. Entonces es fácil factorizar utilizando un método de Fermat llamado Método de factorización de Fermat.

Supongamos que n = pq con p>q. Entonces:

$$n = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2$$

Siempre que p y q estén cercanos,

$$s = \frac{p-q}{2}$$

es pequeño,

$$t = \frac{p+q}{2}$$

es solo un poco más grande que $\sqrt{n}$, y $t^2 - n = s^2$ es un cuadrado perfecto. entonces realizamos:

$$t = [\sqrt{n}],\ t = [\sqrt{n}] + 1,\ t = [\sqrt{n}] + 2$$

hasta que $t^2 - n$ es un cuadrado perfecto de s^2, Aquí $[x]$ denota el menor entero $n \geq x$. Entonces:

$$p = t + s,\ q = t - s$$

A continuación mostramos un pequeño ejemplo con $n = 35011$:

$$\sqrt{n} = 187.11$$

Si $t = 188$, entonces: $\sqrt{t^2 - n} = 18.24$

Si $t = 189$, entonces: $\sqrt{t^2 - n} = 26.64$

Si $t = 190$, entonces: $\sqrt{t^2 - n} = 33 \in Z$

Entonces s $= 33$. Encontramos que $p = t + s = 223$ y $q = t - s = 157$

Otra forma para poder encontrar p y q cuando están muy cercanos lo primero que se necesita es importar una librería como la siguiente: from decimal import *, que redondea los números pero si posteriormente utilizamos el comando: getcontext().prec, permite configurar los lugares necesarios para un numero muy grande. Y el comando Decimal(n).sqrt(), nos da la raíz cuadrada de un número, como se muestra en la figura 3.12.

```
from decimal import*
n=121268191497299918687598335280859340064664866989
getcontext().prec=50
Decimal(n).sqrt()
```

```
Decimal('11012183775132883182953 44.99999999999999999999837')
```

Figura 3.12: Raíz cuadrada de un numero

El número n es el producto de dos factores primos, y por lo tanto se puede factorizar calculando la raíz cuadrada. Entonces si es producto de dos números muy cercanos, un numero tiene que ser mayor que la raíz cuadrada y otro menor, por lo tanto empezamos a probar con números cercanos a la raíz cuadrada saltando hacia atrás en dos como lo mostraremos a continuación:

```
num=0
for p in range(11012183775132883182953 45,11012183775132883182953 00,-2):
    num+=1
    x=n%p
    if(x==0):
        print(num,".-",p,x)
    #print(num,".-",p,n%p)
```

```
4 .- 11012183775132883182953 39 0
```

Figura 3.13: ciclo para buscar

Se puede saltar de dos en dos porque no necesitamos los números que son pares. Entonces como se puede ver en la imagen anterior se encontró un numero donde el modulo de n es 0, lo que nos dice que es un factor primo. a partir del cual se puede obtener el otro factor primo dividiendo n por el primero como se mostrara a continuación:

```
from decimal import*
n=121268191497299918687598335280859340064664866989
p=11012183775132883182953 39
x=n//p
getcontext().prec=100
Decimal(x)

Decimal('11012183775132883182953 51')
```

Figura 3.14: División Entera

Y entonces podemos multiplicar:

```
p=11012183775132883182953 39
q=11012183775132883182953 51
n=p*q
print(n)

121268191497299918687598335280859340064664866989
```

Figura 3.15: División Entera

Entonces podemos demostrar que $p * q$ es el número n original que por lo tanto hemos factorizado un número.

Capítulo 4

Estado del arte

En este apartado se conocerán las investigaciones y prototipos que nos ayuden a entender más sobre nuestra investigación.

4.1. GenRSA

GenRSA es una aplicación gratuita criptográfica que tiene como finalidad facilitar el aprendizaje y el uso del algoritmo RSA, desarrollada por el Dr. Juan Carlos Pérez García en el año del 2004, y fue actualizado después de 13 años por el Dr. Rodrigo Díaz Arroyo.

Cuenta con una serie de características este software como:

- Genera claves de forma manual o automática.
- Permite generar claves de hasta 4096 bits.
- Podemos generar claves con números primos de igual tamaño o diferente.
- Muestra el tiempo necesario para generar claves.
- Permite elegir el valor mínimo para generar la clave pública.

También cuenta con diferentes opciones adicionales como:

- Opciones de cifrado y descifrado.
- Funciones de firmas.
- Test de primalidad.

A continuación se muestra en la figura 4.1 la interfaz principal del sistema GenRSA con una breve descripción.

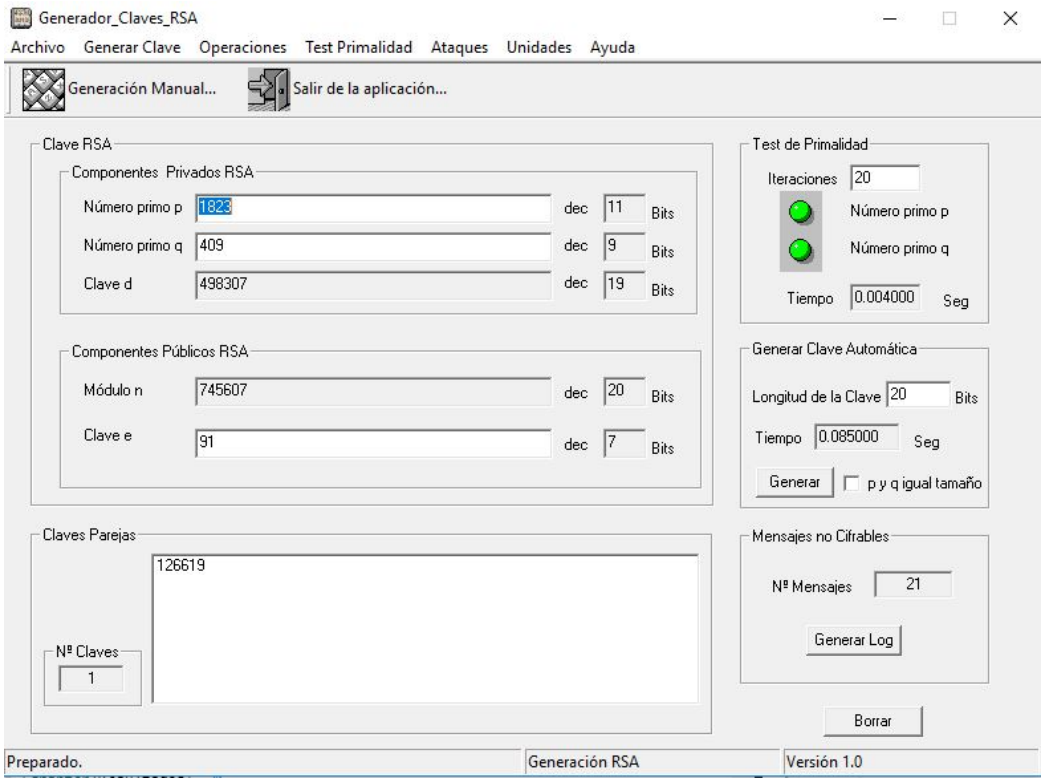


Figura 4.1: Aplicación GenRSA

Esta aplicación[1] está desarrollada en Java, también tiene un manual de usuario que explica a detalle cómo usar la herramienta y una serie de pruebas, pero lo que no explica es como funciona cada parte del código, de esta herramienta tampoco las funciones matemáticas necesarias para poder implementarlo, o incluso puede no ser claro su funcionamiento.

[1] http://www.criptored.upm.es/descarga/genRSA.zip

Capítulo 5

Implementaciones

En esta sección describiremos y explicaremos las pantallas principales del sistema de seguridad RSA implementado con ayuda de todos los algoritmos vistos anteriormente.

5.1. Funciones Hash

Las funciones hash son una función muy importante para esta investigación ya que para la implementación de prototipo necesita realizar un hash a la contraseña y así mantener segura la contraseña del usuario, pero existen muchas funciones como: MD5, SHA-1, SHA-256, SHA-3 y BLAKE2. Esta investigación se implementó una MD5 que toma una entrada de una longitud variable a una salida fija que se denota como: $H : V \longrightarrow F$.

Se utiliza con dos finalidades:

1. Proteger la confidencialidad de una contraseña:

   La contraseña puede estar en texto claro y ser accesible para cualquier intruso, entonces esta se puede proteger aplicando una función hash que puede ser guardada en una base de datos y ser vista por cualquier persona ya que un hash debería ser imposible descifrarlo.

2. Firma digital.

   Es una función que identifica si el archivo recibido no ha sido modificado en su envió, que también ayuda a verificar que el archivo sea del emisor deseado y así comprobar que es de la persona correcta y no de otra persona que haya modificado el archivo. Consiste en crear un hash del archivo y cifrarlo con la clave privada para que cualquier persona que conozca su clave pública pueda ver de donde proviene el archivo.

Incluso con un pequeño cambio en la entrada de una función hash su salida cambiaría de una manera considerable.

5.2. Prototipo

En esta sección describiremos y explicaremos las pantallas principales del sistema de seguridad RSA implementado con ayuda de todos los algoritmos desarrollados anteriormente.

5.2.1. Inicio de sesión

En la figura 5.1 se muestra como es el inicio de sesión del sistema de seguridad. Se necesita que el usuario ingrese su nombre de usuario y su contraseña para poder ingresar. Una vez realizado lo anterior tiene que dar click al botón iniciar sesión entonces el sistema tiene que validar si está registrado en la base de datos. Otra etapa importante de la funcionalidad del sistema es que la contraseña sea guardada en la base de datos con su valor Hash, entonces cada vez que inicie sesión se pasara a su valor Hash y será validado ya que dos palabras no pueden tener el mismo Hash. En caso de que el sistema detecte que el usuario no está registrado este podrá hacerlo oprimiendo el botón registrarse y aparecerá una nueva ventana como mostraremos en la figura 5.2.

Figura 5.1: Pantalla inicio de sesión

5.2.2. Registro

En la siguiente imagen el usuario ingresará los datos con los cuales será registrado que son:

- Ingresa el usuario

- Ingresa el correo electrónico

- Ingresa la contraseña

- Validación de contraseña

La contraseña se pido una vez más para validar que es la misma y cuando el usuario oprime guardar el sistema realiza una función hash a la contraseña y esa es guardada en la base de datos que después la recuperará para iniciar sesión.

Figura 5.2: Pantalla de registro

5.2.3. Bienvenida

En la figura 5.3 se puede observar en la pantalla de bienvenida algunas funcionalidades del sistema, y el botón que nos enviará a la pantalla principal del sistema.

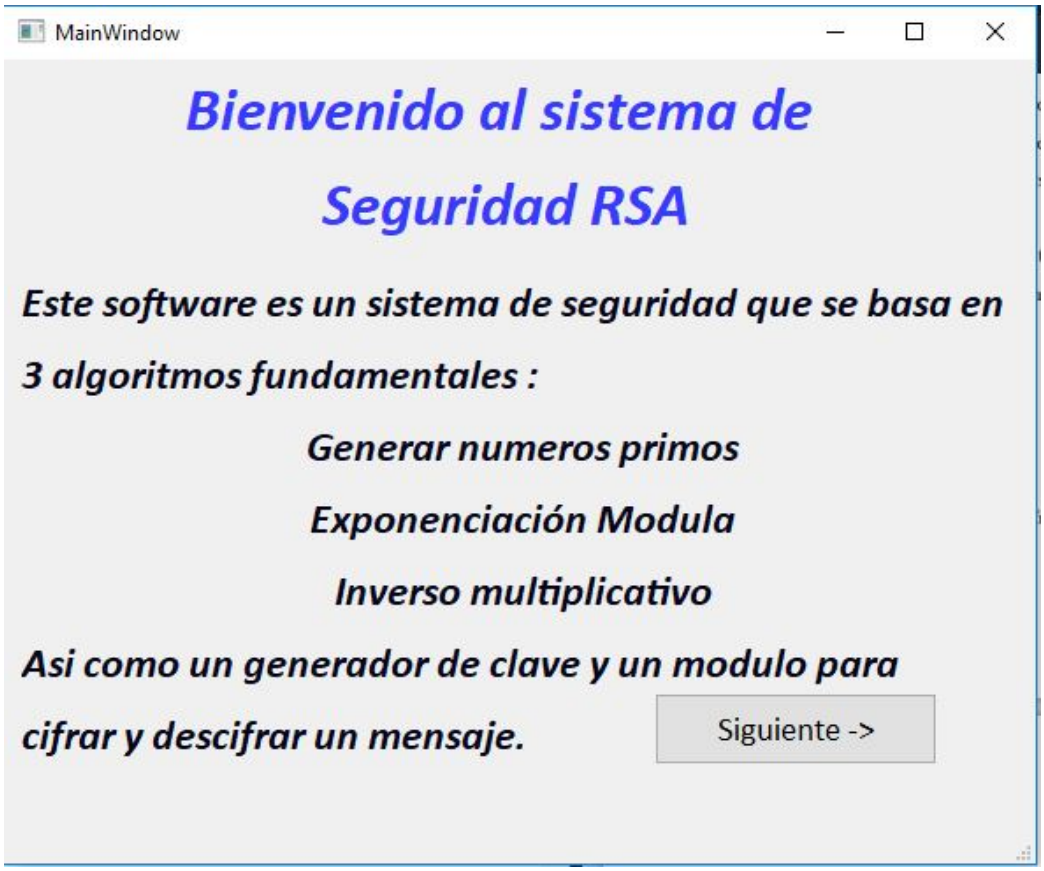


Figura 5.3: Pantalla inicio de bienvenida

5.2.4. Principal

En la siguiente imagen se puede observar la primera pestaña de la pantalla principal que esta desarrollada para el algoritmo de Rabin - Miller.

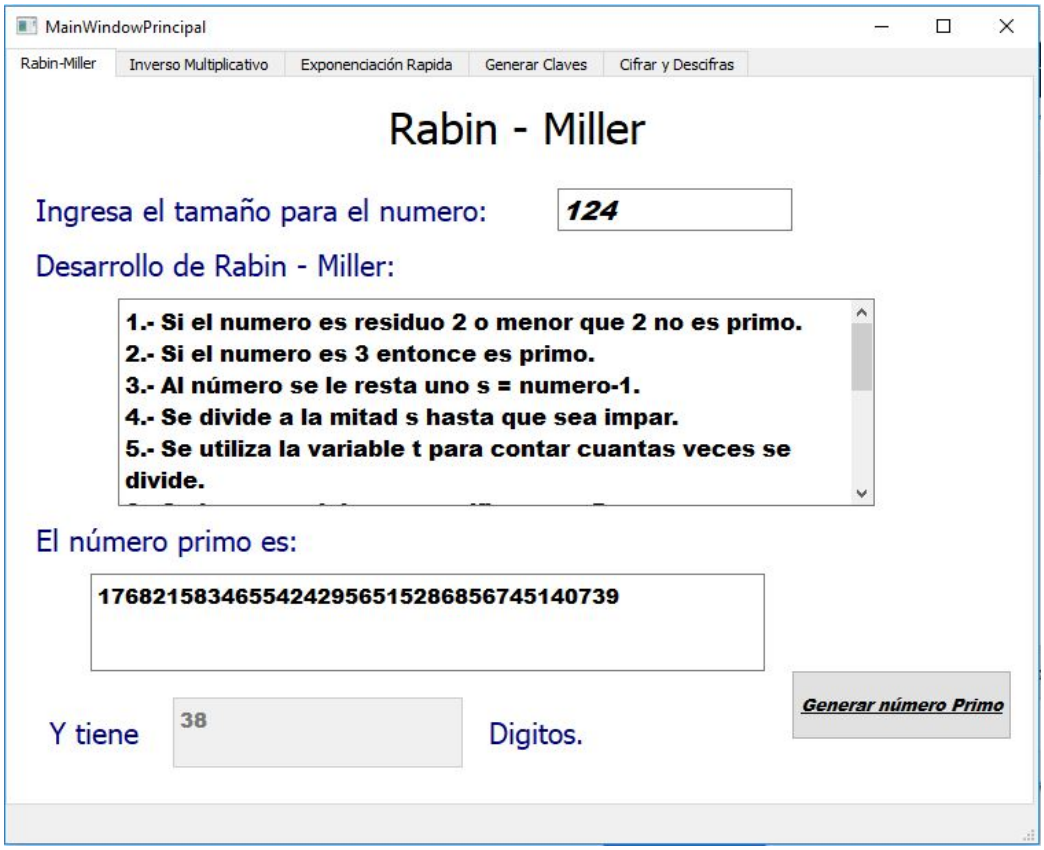


Figura 5.4: Pantalla principal Rabin - Millar

En la figura 5.4 el usuario solo tiene que ingresar el tamaño del número en bits para generar un número primo, al oprimir el botón **generar un número primo**, en la pantalla se despliega el desarrollo de Rabin - Miller, y muestra cual es el número primo generado junto con cuantos dígitos tiene el número.

La segunda pestaña de la pantalla principal fue desarrollada para encontrar el inverso multiplicativo de $(p-1)(q-1)$ donde p y q son los primos necesarios para generar le clave pública n.

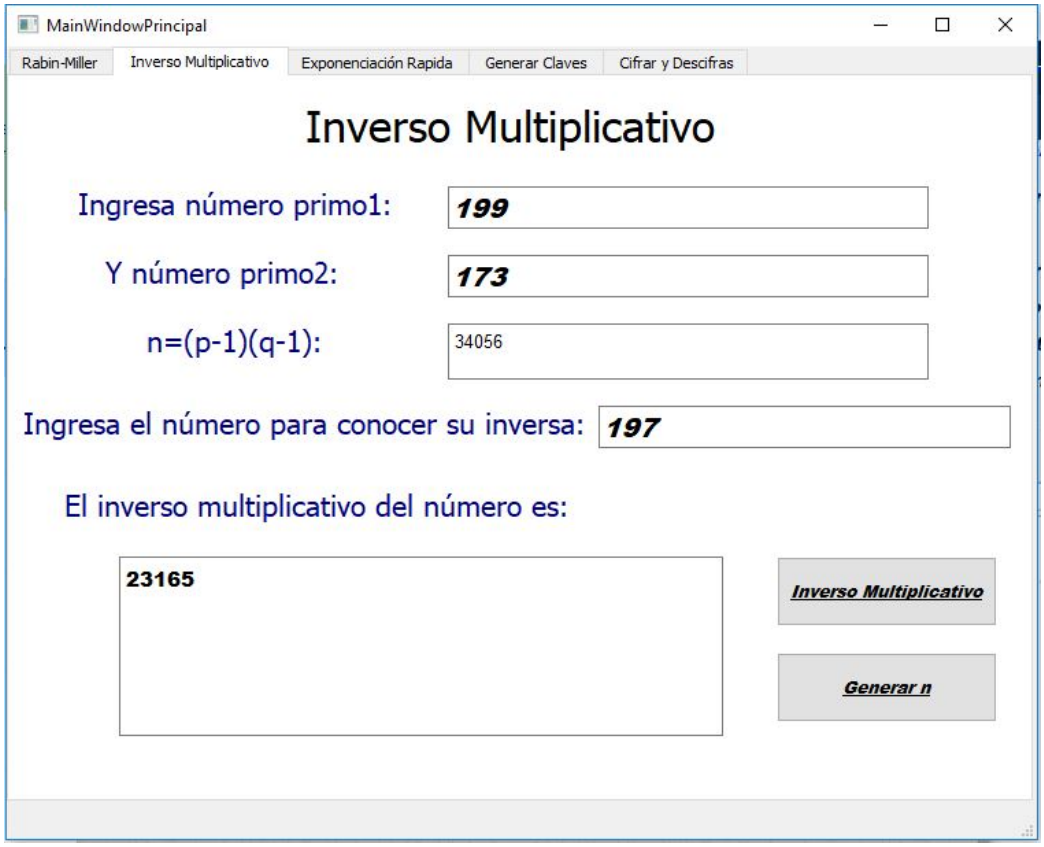


Figura 5.5: Pantalla principal Inverso Multiplicativo

En la figura 5.5 el usuario tiene que ingresar dos números primos puede obtenerlos de la pantalla de Rabin - Miller oprimiendo el botón **generar numero primo** dos veces. Entonces el usuario oprime el botón generar n y en la etiqueta de texto aparece el valor de n. Después el usuario tiene que ingresar el numero para conocer su inversa pero antes el sistema verifica que n y el número ingresado sean primos relativos. Si el número ingresado no es primo relativo a n en la etiqueta de texto no aparecerá nada en la pantalla, en otro caso esta desplegará el inverso multiplicativo.

La tercera pestaña de la pantalla principal fue desarrollada para ilustrar el algoritmo de exponenciación rápida como se ve a continuación.

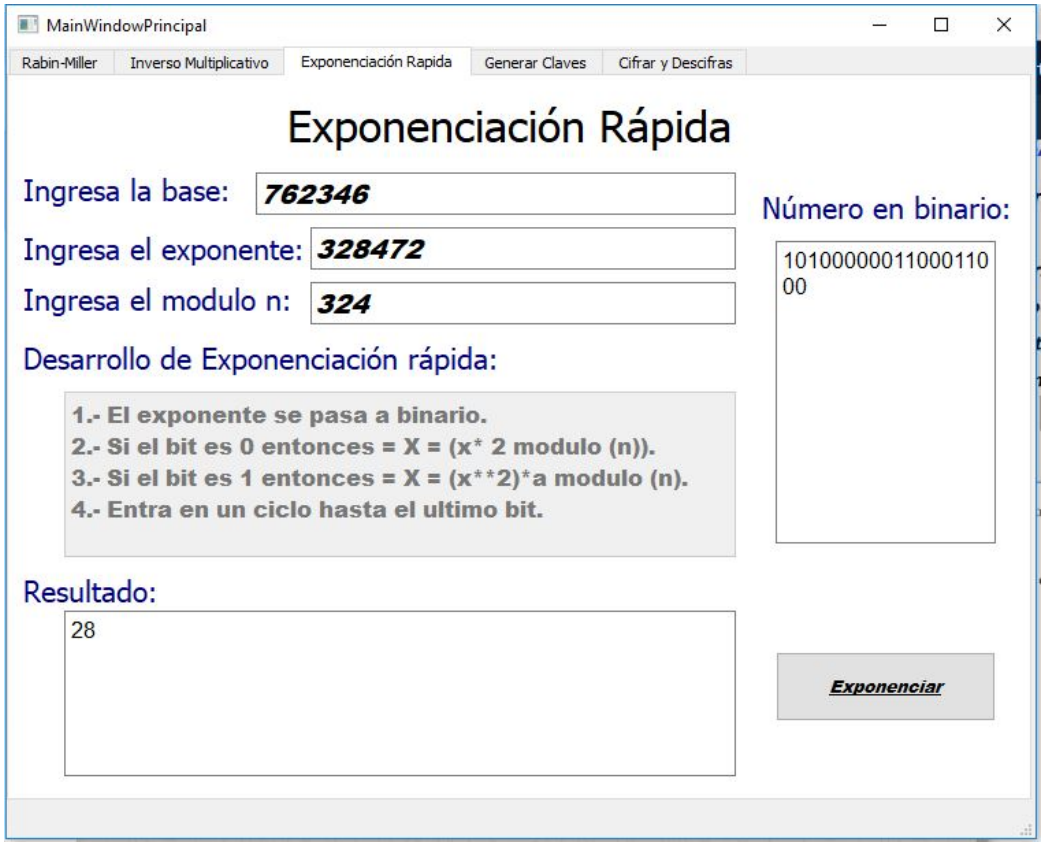


Figura 5.6: Pantalla principal exponenciación rápida

En la figura 5.6 el usuario tiene que ingresar el número base, el número exponente y el modulo, cuando el usuario presione exponenciar aparecerá como se desarrolla la exponenciación rápida, el numero exponente en binario y el resultado de la exponenciación.

La cuarta pestaña de la pantalla principal es para generar claves, el usuario tiene que ingresar el nombre de los archivos donde se guardaran las claves, así mismo el usuario elige el tamaño de las claves en bits como se puede observar en la figura 5.7.

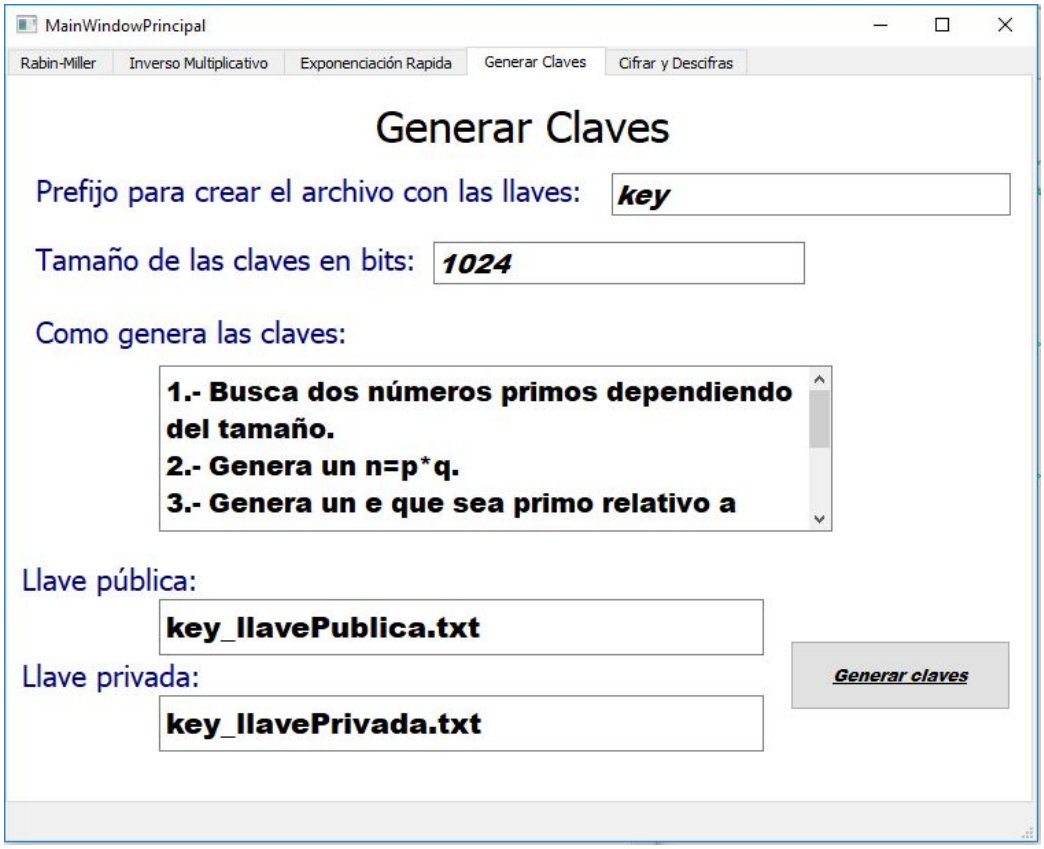


Figura 5.7: Pantalla principal Generador de claves

Cuando el usuario presione el botón generar claves, en la pantalla se puede observar como se generan las clave, también los nombres de los archivos donde fueron guardadas las claves.

La quinta pestaña es la recopilación de las otras cuatro, tiene dos secciones, la primera es para cifrar y la segunda para descifrar un mensaje. En la figura 5.8 se observa el primer apartado que es cifrar.

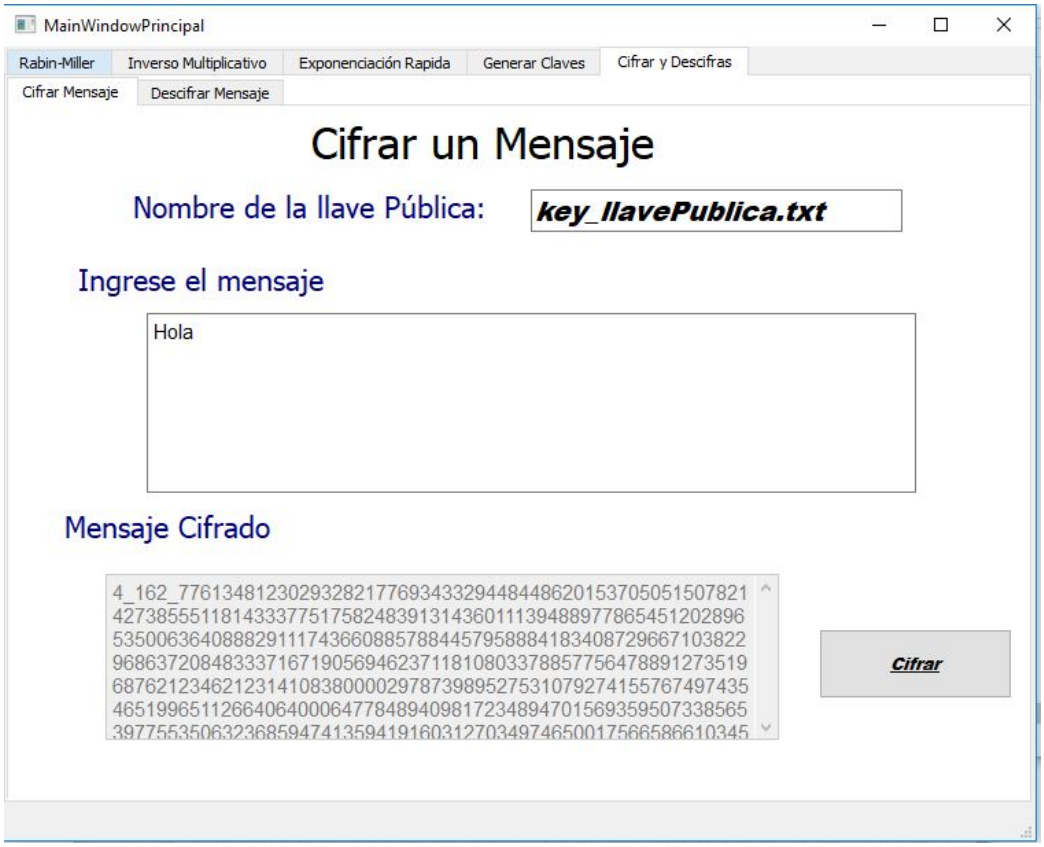


Figura 5.8: Pantalla principal para cifrar

Como se puede observar en la figura anterior el usuario ingresa el nombre de la llave publica, generada en la pestaña 4, el usuario ingresa el mensaje y cuando presione en botón **cifrar** aparecerá el mensaje cifrado.

En la segunda sección se podrá descifrar el mensaje obtenido en la sección anterior.

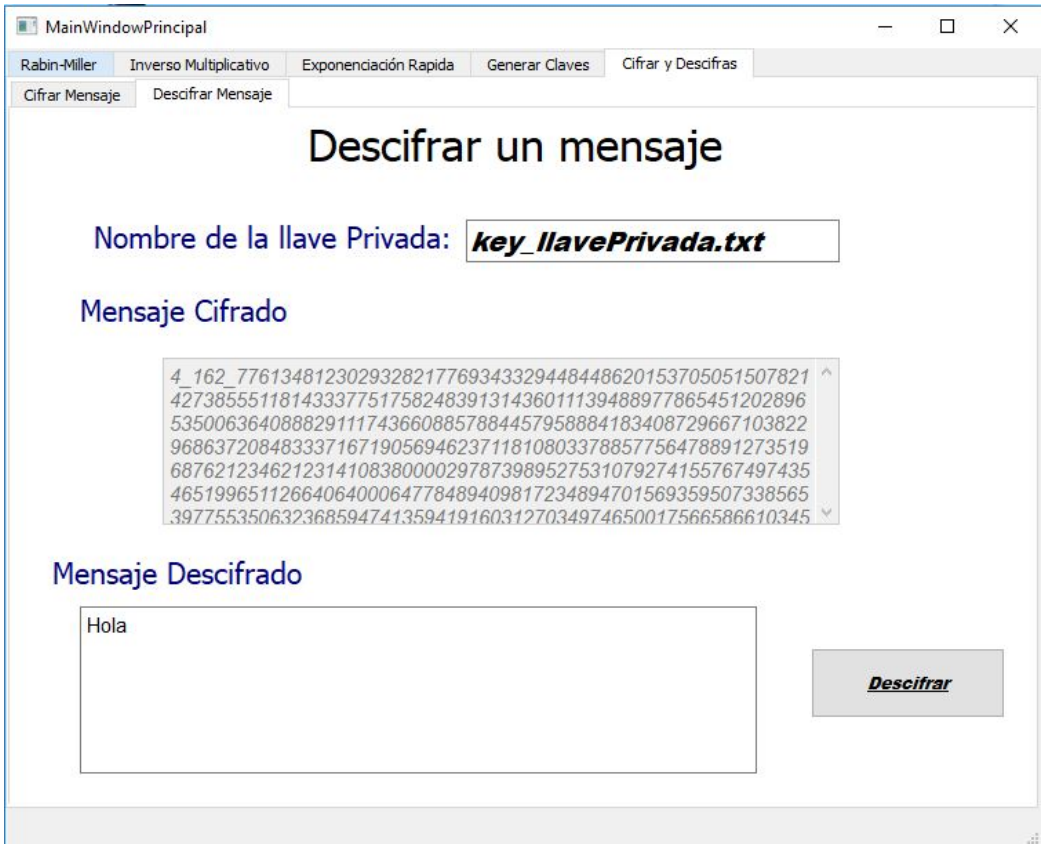


Figura 5.9: Pantalla principal para Descifrar

En la figura 5.9 se puede observar que para poder descifrar el mensaje solo necesitamos ingresar el nombre de la clave privada y cuando el usuario presione el botón descifrar aparecerá el mensaje antes cifrado.

5.2.5. Emergentes

En la figuras 5.10 y 5.11 se muestran algunas pantallas que aparecen en el sistema cuando existe un error.

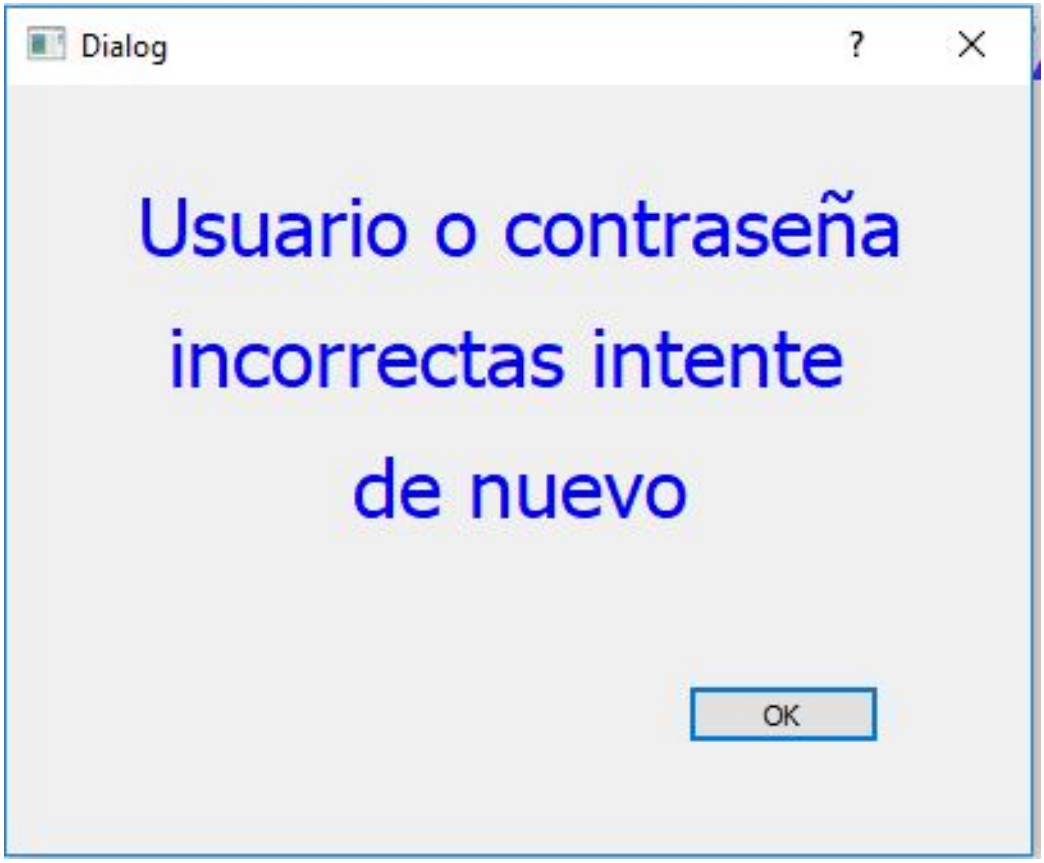


Figura 5.10: Error contraseña o usuario incorrectos

La pantalla anterior aparece cuando en el inicio se sesión el usuario quiere acceder al sistema e ingresa su usuario y contraseña pero cuando presiona el botón inicio sesión el sistema busca en la base de datos los valores y si no los encuentra sale el error de usuario o contraseña incorrecto.

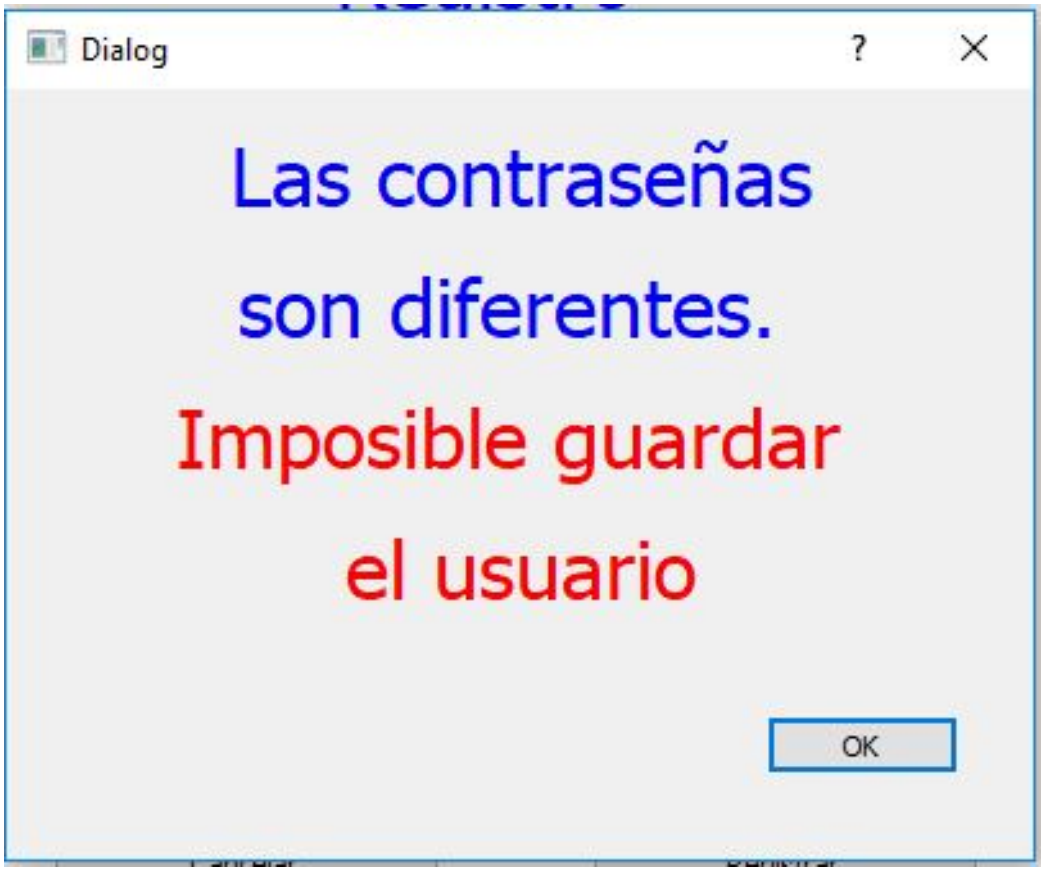


Figura 5.11: Error contraseñas diferentes

En la figura 5.11 es otro mensaje de error que aparece cuando un usuario es nuevo y quiere registrarte en el sistema entonces el usuario tiene que escribir dos veces la contraseña y después oprimir el botón guardar el sistema verifica que sean la contraseña si no son la misma aparece el error anterior.

Pero si son la misma contraseña se guarda en la base de datos y aparece un mensaje de que el usuario se ha registrado exitosamente como mostraremos a continuación.

Figura 5.12: Usuario registrado

El sistema de seguridad[1] implementado en este trabajo cifra y descifra un mensaje, posteriormente el mensaje es divido en bloques para poder cifrar y descifrar el mensaje completo, el tamaño de los bloques depende de tamaño de las claves.

[1]https://drive.google.com/drive/folders/1vS4dOsiGA4sWw6eN4XLs9JfXpdvtZyNK

Capítulo 6

Resultados

El sistema criptografico RSA es actualemnte el sistema de seguridad más utilizado para el envió de informacion por un canal inseguro.

En este trabajo se realizó una presentación didactica al sistema de seguridad RSA, para lo cual fue necesario abordar conceptos como teoría de números, aritmetica modular y se desarrollaron los algoritmos fundamentales en que se basa el RSA como fueron: generar numeros primos muy grandes, exponenciación rápida e inverso multiplicativo. Con ayuda de estos conceptos se desarrollo una interfaz con la finalidad de que el usuario pueda entender como funciona internamente el algoritmo RSA.

También se abordo el tema de la seguridad del algoritmo, en particular el problema de factorizar un numero n en primos. El sistema desarrollado para este trabajo, puede generar números primos de 2048 bits, a la fecha no hay un algoritmo tan eficiente para poder factorizar un número tan grande en sus factores primos en un tiempo razonable, pero hay que tener mucho cuidado que los factores de n no estén muy cercanos.

De la lectura de este trabajo se espera que el lector conozca como funciona internamente el sistema RSA, sus virtudes y defectos que le permitiría desarrollar mejoras en este sistema.

La interfaz presentada no representa un trabajo terminado, en un trabajo posterior se podrá agregar animaciones lo cual seria un aporte pedagógico.

Bibliografía

[1] Aguirre Ramió, J. (2016). Seguridad informática y criptografía. Madrid, España: OxWord.

[2] Aumasson, J. P.(2018). Serious cryptography a practical introduction to modern encryption. San Francisco: No Starch Press.

[3] Baig, M. (2001). Criptografía cuántica. Universitat Autònoma de Barcelona. Spain.

[4] Bakogiannis, C., Karapanos, N. (13 de agosto de 2009). KmGNFS - A General Number Field Sieve (GNFS) implementation. Recuperado el 13 de enero de 2019 de http://kmgnfs.cti.gr/kmGNFS/Home.html

[5] Beissinger, J., & Pless, V. (2018). The cryptoclub : using mathematics to make and break secret codes. Wellesley, Massachusetts. CRC Press.

[6] Bowne, S. (2018). Hands-on cryptography with Python. Birmingham: Packt publishing.

[7] Briggs, M. E. (1998). An introduction to the general number field sieve, Tesis Doctoral. Virginia Tech.

[8] Case, M. (2003). A beginner's guide to the general number field sieve. Oregon State University, ECE575 Data security and cryptography project.

[9] Delfs, H., & Knebl, H. (2015). Introduction to cryptography(Third Edition). Heidelberg: Board.

[10] Fernandez, S. (2004). La criptografia Clasica. Sigma: revista de matemáticas = matematika aldizkaria, 119-142.

[11] Gutierrez, P. (21 de junio de 2017). Genbeta. Recuperado el 27 de febrero de 2018, de https://www.genbetadev.com/seguridad-informatica/que-es-y-como-surge-la-criptografia-un-repaso-por-la-historia.

[12] Hoffstein, J., Pipher, J., & Silverman, J. (2008). An introduction to mathematical cryptography(Vol. 1). New York: springer.

[13] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thomé, E., Bos, J. W., ... Te Riele, H. (2010). Factorization of a 768-bit RSA modulus. In annual cryptology conference (pp. 333-350). Springer, Berlin, Heidelberg.

[14] Maubert, I. (1 de Diciembre de 2015). BeDetective. Recuperado el 20 de mayo de 2017, de https://www.bedetective.com/criptografia

[15] Sánchez Muñoz, J. M. S.(2013). Historias de matemática. criptologia nazi. Los códigos secretos de Hitler. Pensamiento matematico,3(1), 59-120

[16] Stein, W. (2009). Elementary number theory: primes, congruences,and Secrets. New York: Board.

[17] Sweigart , A. (2018).Cracking codes with Python: an introduction to building and breaking ciphers.. San Francisco: No Starch Press.

[18] Talbot, J., & Welsh, D. (2006). Complexity and cryptography. New York: an introduction (vol.13) Cambridge University Press.

[19] UNAM Facultad de ingeneria. (9 de Noviembre de 2017). Recuperado el 20 de Diciembre de 2017, de http://www.redyseguridad.fi-p.unam.mx/proyectos/criptografia/index.php/

[20] Van Rossum, G. (2001-2018). Python.Recuperado el 2 de diciembre de 2018 de https://www.python.org/.

[21] Y. Yan, S., Yung, M., & Rief, J. (2013). Computational number theory and modern cryptography. New Delh: Wiley.