



**POLITÉCNICA**

"Ingeniamos el futuro"

# **UNIVERSIDAD POLITÉCNICA DE MADRID**

Escuela Técnica Superior de Ingeniería de  
SISTEMAS INFORMÁTICOS

Grado en Ingeniería de Computadores

Curso 2016 – 2017

## **Software genRSA v2.1: Generación, cifrado y ataques a claves RSA**

**Autor: Rodrigo Díaz Arroyo**

Tutor: Jorge Ramío Aguirre



## **AGRADECIMIENTOS**

A mi madre Amelia y a mi padre Pedro por haber confiado en mí y haberme posibilitado acceder a la Universidad invirtiendo en mi educación.

A mi hermana Bárbara por apoyarme y aconsejarme durante toda mi vida.

A mis seres queridos por la fortaleza que me han otorgado entregándome su amor incondicional.

Y por último, a aquel que nos demostró que  
"Si se cree y se trabaja, se puede"



# Resumen

---



El programa genRSA v2.1 es una actualización de la versión 1.0. Este software incluye multitud de funcionalidades que tienen como fin mejorar el aprendizaje del algoritmo RSA y de las operaciones que se pueden realizar con él. Entre otras, las funcionalidades más destacadas son:

- Generación automática de claves RSA utilizando como clave pública la más usada en certificados digitales (65.537).
- Operaciones de Cifra, Firma, Descifrado y Validación de mensajes o números.
- Ataques a los componentes de la clave RSA por distintos métodos: cifrado cíclico, factorización y paradoja del cumpleaños.

Esta actualización pretende mejorar varias facetas de este software que han quedado obsoletas y cuyo diseño no facilitan que esta sea una aplicación didáctica. Las más relevantes son:

- Diseño gráfico: tomando como punto de partida que la versión 1.0 tiene una pantalla principal muy sencilla e intuitiva. En la nueva versión se añade lo básico para aumentar su funcionalidad y permitir maximizar las ventanas en todo momento. Particularmente, se mejoran todas las ventanas secundarias haciéndolas visualmente más atractivas con una interfaz más sencilla y didáctica.
- Aprendizaje: se han añadido botones de información los cuales facilitaran el aprendizaje acerca del algoritmo, los ataques a claves RSA y las operaciones de cifra y firma en criptografía asimétrica. Además siempre se mostrará en pantalla todos los datos necesarios para realizar los ataques o las operaciones.

Gracias a esta actualización el programa resulta muy sencillo de usar. En caso de cometer un error se obtendrá un mensaje donde se indica a que se debe y como solucionarlo.

- Computo: se han mejorado los algoritmos empleados en toda la aplicación. Lo que supone poder generar claves de mayor tamaño y poder ejecutar ataques sin restricción de longitud de clave.

Todos con estos cambio han permitido mejorar los tiempos de ejecución dando lugar a una aplicación más rápida con una interfaz gráfica que siempre responde.





# Abstract

---



GenRSA v2.1 program is an update of the version 1.0. This software includes multiple functionalities that aim to improve the learning of the RSA algorithm and also the Cipher and Sing operations. The most notable features of this software are:

- Automatic RSA key generation using the public key most used in digital certificates (65.537).
- Sign, Cipher, Decipher and Validation operations with messages or numbers.
- Attacks to RSA key components by different methods: cyclic encryption, factorization and birthday paradox.

This update have as an objective to improve several aspects that in the last version have become outdated. Also the design of that version does not make easier learn the task of learning. The most relevant are:

- Graphical design: genRSA version 1.0 has a very simple and intuitive main screen. Because of this, the new version only adds basic changes which increase the functionalities and allow to maximize windows in al situations. Also, all the secondary windows are improved making them visually more attractive with a simpler and didactic interface.
- Learning: information buttons have been added in order to facilitate learning about the algorithm, attacks on RSA keys, and encryption and signature operations on asymmetric cryptography. Also In addition, all data necessary to perform the attacks or operations will always be displayed on the screen.

Thanks to this update the program is more easy to use. In case of making a mistake you will get a message where it is indicated the reason of the error and how to solve it..

- Calculation: all the applications algorithms have been improved. This means being able to generate keys of greater size and to execute attacks without restriction of key length.

All this updates have allowed to improve the execution times. And this has led to a faster application with a graphical interface that never makes the application user interface unresponsive.



# Tabla de Contenidos

<b>Capítulo 1: Introducción .....</b>	<b>1</b>
<b>Capítulo 2: Establecimiento de Requisitos .....</b>	<b>5</b>
2.1 Requisitos Funcionales .....	7
2.2 Requisitos de Arquitectura .....	8
2.3 Requisitos de Interfaz Gráfica .....	9
<b>Capítulo 3: Software Relacionado .....</b>	<b>11</b>
3.1 RingRSA.....	13
3.2 LegionRSA.....	14
<b>Capítulo 4: Tecnologías y Herramientas.....</b>	<b>15</b>
4.1 Tecnologías .....	17
4.2 Entornos de programación.....	19
4.3 Generador de diagramas UML .....	20
4.4 Analizador de código.....	20
4.5 Repositorio de código y documentación .....	21
<b>Capítulo 5: Requisitos mínimos del sistema .....</b>	<b>23</b>
5.1 Espacio en Disco .....	25
5.2 Memoria RAM .....	25
5.3 Resolución gráfica .....	26
<b>Capítulo 6: Librerías .....</b>	<b>27</b>
6.1 Parte lógica .....	29
6.2 Parte gráfica .....	30
6.3 Concurrencia .....	31
<b>Capítulo 7: Desarrollo de la Aplicación.....</b>	<b>33</b>
7.1 Diseño Gráfico .....	35
7.2 Desarrollo Lógico.....	51
<b>Capítulo 8: Aspectos destacados de la programación.....</b>	<b>55</b>
8.1 Paquete PreloaderGenRSA.....	57
8.2 Paquete GenRSA .....	59
8.3 Paquete Methods.....	67
8.4 Paquete Model .....	75
8.5 Paquete Cyclic .....	76

8.6 Paquete Factorize .....	85
8.7 Paquete Paradox .....	88
8.8 Paquetes DeCipher y Sign .....	89
8.9 Paquete Imprimir .....	92
<b>Capítulo 9: Diagrama de Clases .....</b>	<b>99</b>
<b>Capítulo 10: Funcionalidades de genRSA v2.1 .....</b>	<b>103</b>
10.1 Generación Manual .....	106
10.2 Generación Automática .....	106
10.3 Test de Primalidad .....	107
10.4 Gestión de claves .....	107
10.5 Generar Log de Números No Cifrables .....	108
10.6 Cifrar-Descifrar .....	109
10.7 Firmar-Validar.....	109
10.8 Ataque Factorización.....	110
10.9 Ataque Cíclico .....	110
10.10 Ataque Paradoja del Cumpleaños.....	111
<b>Capítulo 11: Mejoras efectuadas.....</b>	<b>113</b>
11.1 Mejora de la parte gráfica.....	115
11.2 Mejora en facilidad de uso y mejor aprendizaje .....	115
11.3 Mejora de funcionalidad .....	116
11.4 Mejora en estabilidad .....	116
11.5 Mejora en velocidad.....	116
<b>Capítulo 12: Desarrollos Futuros.....</b>	<b>119</b>
12.1 Mejora del ataque por la paradoja del cumpleaños.....	121
12.2 Comprobación exhaustiva de los datos introducidos.....	121
12.3 Indicadores de progreso determinados .....	122
12.4 Contenido y aumento de los tooltips .....	122
12.5 Añadir hojas de estilo.....	122
12.6 Guardar estado de los ataques .....	123
<b>Capítulo 13: Valoración Económica .....</b>	<b>125</b>
<b>Capítulo 14: Conclusiones.....</b>	<b>129</b>

# Capítulo 1

## Introducción

---





El algoritmo RSA, y la criptografía asimétrica en general, es de uso cotidiano por todas las personas cuando navegan por Internet. Sus usos más destacados están relacionados con el intercambio de clave de sesión en el protocolo SSL/TLS y con la creación de Infraestructuras de Clave Pública, las cuales permiten generar confianza en las Autoridades Certificadoras (imprescindibles para navegar de forma segura por Internet).

La criptografía asimétrica es conocida como criptografía de clave pública. En este método criptográfico se usan un par de claves: la clave privada, que nunca debe ser revelada y la clave pública, que puede ser difundida a todo el mundo. La criptografía asimétrica se basa en la exponenciación modular para realizar las operaciones de cifra y de firma.

Para la operación de cifra, el emisor usa la clave pública del receptor para cifrar la información, de tal manera que solo el receptor puede descifrar la información con la clave privada. De este modo se garantiza la confidencialidad de la información.

Para la operación de firma, el emisor usa su clave privada para firmar la información. De este modo, el receptor usando la clave pública del emisor puede validar que este ha sido quien ha enviado la información y que esta está completa.

La aplicación genRSA v2.1 es una aplicación didáctica que permite adquirir y mejorar conocimientos acerca del algoritmo RSA. Los alumnos de la Escuela Técnica Superior de Ingeniería de Sistemas Informáticos (E.T.S.I.S.I.) han usado la versión 1.0 de esta aplicación para realizar prácticas de la asignatura Seguridad de la Información.

Partiendo de la funcionalidad de la versión 1.0, se pretende desarrollar desde cero la aplicación para ampliar funcionalidades, mejorar algoritmos y hacerla mucho más didáctica. Permitiendo de este modo que la nueva versión sea un complemento al estudio del algoritmo de criptografía asimétrica RSA.



# Capítulo 2

## Establecimiento de Requisitos

---



Para identificar las necesidades que debía cubrir esta software se concertaron varias reuniones con el tutor del proyecto, Dr. Jorge Ramió Aguirre. A lo largo de las primeras reuniones se determinaron los requisitos principales.

Pero dada la envergadura del proyecto, mientras se desarrollaba el software aparecieron nuevas necesidades a la par que otras se fueron acotando.

A continuación se exponen los requisitos funcionales acordados en la fase inicial del proyecto. Los cuales definen característicamente las necesidades del mismo. Estos requisitos se dividen en requisitos de arquitectura, requisitos funcionales y requisitos de interfaz gráfica.

## **2.1 Requisitos Funcionales**

### **2.1.1 Conservar las mismas Funcionalidades**

Dado que es una actualización, genRSA v2.1 tendrá las mismas funcionalidades que la versión v1.0. Estas funcionalidades se mejorarán en los siguientes puntos: mayor rendimiento en cálculo, interfaz gráfica más intuitiva, uso más sencillo, mejor comprobación de errores en las entradas y mayor rango de claves. Además se incluirán nuevas funcionalidades como visualización de ataques en tiempo real, firma y comprobación de firma entre otras.

### **2.1.2 Generación de claves de mayor cantidad de bits**

Se deberá permitir generar claves, tanto hexadecimales como decimales, cuya cantidad de bits sea de hasta 4096. Esto hará que el programa pueda ser usado para generar claves de longitud más segura y acorde con el año 2017.

### **2.1.3 Software didáctico**

El tipo de usuario principal de este software es un estudiante universitario. Por este motivo, genRSA deberá ser un software intuitivo y del que se pueda aprender fácilmente mientras se usa. Para poder aprender de él se dispondrá de ejemplos guiados, botones de información y diálogos de error dando soluciones.

#### **2.1.4 Generación de claves de manera automática**

Actualmente muchos certificados usados para servidores seguros utilizan una clave pública muy concreta: 65.537. El programa deberá permitir elegir si se quiere usar esta clave pública a la hora de generar una clave de manera automática.

## **2.2 Requisitos de Arquitectura**

### **2.2.1 Estabilidad del programa**

El software deberá ser estable. Para ello, los procesos pesados en cómputo deberán poder detenerse manteniendo una interfaz gráfica que responda siempre.

Todas las entradas deberán ser filtradas y procesadas antes de ser empleadas en cualquier ejecución. Evitando de este modo que la aplicación pueda entrar en estado de error.

Gracias a ello, genRSA v2.1 será un programa a prueba de fallos dando margen a la equivocación resultante del uso de la misma por usuarios no experimentados.

### **2.2.2 Uso de librerías que no caduquen**

Para evitar que ocurran fallos debido a que las librerías se queden obsoletas, para esta actualización se usaran librerías matemáticas. Estas librerías son menos susceptibles a cambios en su funcionalidad. Sin embargo, estas librerías sí que son actualizadas más frecuentemente para mejorar el rendimiento. De este modo, este software podrá funcionar correctamente durante más tiempo sin que se quede anticuado.

### **2.2.3 Mejorar rendimiento de todos los ataques**

Todos los ataques que se pueden realizar en genRSA v1.0 tienen dos puntos en común: solo funcionan para claves pequeñas y tienen una potencia de cálculo muy por debajo de lo normal en el año 2017. En la actualización se solucionarán ambos problemas: se podrán realizar ataques sin importar el número de bits de la clave y se usarán librerías y algoritmos con mucho mayor rendimiento y potencia de cálculo.

## **2.3 Requisitos de Interfaz Gráfica**

### **2.3.1 Diseño de pantalla principal**

Debido a que genRSA es un programa que es bastante conocido dentro del mundo universitario, la pantalla principal deberá guardar una apariencia muy similar a la de la versión ya existente. Solo se harán pequeñas modificaciones para incorporar funcionalidades.

### **2.3.2 Actualización de términos mostrados en pantalla**

Se añadirán los componentes de la clave que se usen al realizar las operaciones y ataques. Además, son varios los términos que necesitarán ser actualizados de forma que se facilite la comprensión de su significado.

Así mismo, deberá ser más sencillo leer los números decimales mostrados por pantalla. Por lo que estos se mostrarán separados por puntos cada tres cifras.

### **2.3.3 Información en tiempo real**

En todos los ataques se mostrará una pantalla que informe en tiempo real acerca de la ejecución de los mismos.

También resultará interesante que siempre que se genere una clave se muestre el número de bits de cada clave privada pareja calculada.

### **2.3.4 Mejorar ventanas de ataques**

Todas las ventanas de ataques deberán ser diseñadas de nuevo haciendo su uso mucho más sencillo e intuitivo. Estas mostrarán los parámetros necesarios para ejecutar cada tipo de ataque. De modo que el usuario sea consciente de que datos son necesarios para ejecutar cada ataque.

### **2.3.5 Rediseño de la ventana de Cifrado y Descifrado**

Actualmente el programa solo tiene una pantalla para cifrado y descifrado. Además dicha pantalla no indica con que clave (pública o privada) se realiza el cifrado y con cual el descifrado.

Se creará una ventana en la cual estén claramente diferenciadas la parte del cifrado y la parte del descifrado. No obstante, la ventana deberá aportar información suficiente para mejorar la comprensión del cifrado y descifrado en personas que estudien el algoritmo de cifrado RSA.

### 2.3.6 Creación de ventana de Firma y Validación de firma

Esta ventana aportará una funcionalidad totalmente nueva. Como en el caso anterior, estará dividida en una parte para firma y otra para validación de firma.

El fin último, será facilitar la comprensión de la firma y la validación de la firma a la par que complementar la información acerca del cifrado. Para ello esta ventana también añadirá botones de información donde se explica el proceso de Firma y de Validación de Firma.



# Capítulo 3

## Software Relacionado

---



Son varios los programas que se han desarrollado por alumnos de la Escuela que están relacionados con el algoritmo RSA. Sin duda el más conocido es genRSA v1.0 y esto es gracias a la cantidad de funcionalidades que implementa.

A pesar de ello genRSA es un software del año 2004 que necesita ser actualizado. Para ello, se tendrán en cuenta las bondades de otros programas recientemente desarrollados en el ámbito de las claves RSA.

### **3.1 RingRSA**

RingRSA es un software que haciendo uso de la generación de claves RSA, explota como ningún otro programa el cálculo de la formación de anillos. Al igual que genRSA, este software tiene un fin didáctico. Explica de una forma muy sencilla lo que son los anillos de cifrado, permitiendo su visualización por pantalla en tiempo real y en forma de gráfica circular.

Debido a su carácter eminentemente didáctico, algunas de sus características se han tomado como ejemplo para ser implementadas en la actualización de genRSA. Entre otras se encuentran las siguientes:

- Visualización en tiempo real de los resultados: gracias al uso de Threads (hilos de ejecución) se pueden usar varios procesadores del ordenador al mismo tiempo. Estos también permiten desarrollar una aplicación más rápida que muestre los resultados en tiempo real.
- Interfaz gráfica más actual: el uso de las librerías de JavaFX permiten desarrollar programas cuyas interfaces gráficas posean elementos gráficos muy actuales: control de progreso, cajas de selección, "tooltips" y elementos de menú entre otros.
- Algoritmo de ataque cíclico: es sabido que los ataques en genRSA son demasiado lentos y no permiten ser ejecutados con claves de tamaño elevado. Por ello, se toma como ejemplo el modo de actuar del ataque cíclico en RingRSA.

Este aporta botones para ejecutar el ataque hasta que prospere, un contador de tiempo empleado durante el ataque y cajas de texto que muestran la salida del ataque en tiempo real.

- Botones para personalizar la generación de claves: RingRSA da muchas facilidades al usuario para la generación automática de claves. Incluso permite configurar algunos parámetros.

Dado que genRSA es un programa didáctico orientado a la generación de claves, para su uso es vital que la generación sea lo más fácil y configurable posible. Por este motivo, se añadirán botones que permitan una mayor configuración en la generación automática y listas desplegables que faciliten la generación manual.

### 3.2 LegionRSA

LegionRSA es un software enfocado exclusivamente a realizar un ataque por la paradoja del cumpleaños de manera distribuida. Al contrario que genRSA el fin de este programa no es notoriamente didáctico. Pero aun así se puede aprender mucho de él, tanto a la hora de entender el ataque por la paradoja del cumpleaños como a la hora de desarrollar un producto software.

Las características de LegionRSA que marcaron el desarrollo de genRSA v2.1 son las que se exponen a continuación:

- Interfaz gráfica muy sencilla: LegionRSA muestra por pantalla los botones y la información elemental para realizar el ataque. De este modo no existen botones que se presten a confusión, como en ocasiones ocurren con el software ringRSA o la antigua versión de genRSA.
- Programa estable: es un software bien programado que es capaz de recuperarse a pesar de haber fallos en parte de su ejecución (por ejemplo un cliente que ha perdido conexión). Esta es una característica fundamental de la que se debe tomar conciencia a la hora de desarrollar el código de genRSA.
- Velocidad en cómputo: LegionRSA está altamente optimizado para realizar cálculos con potencias de dos. Además también optimiza en gran medida el código que se encuentra en el interior de aquellos bucles que se ejecutan miles de millones o incluso billones de veces.

# Capítulo 4

## Tecnologías y Herramientas

---



Desarrollar una aplicación como genRSA supone tener que investigar qué tecnologías y herramientas serán las más apropiadas para el desarrollo.

Lo primero es saber con qué tecnologías se va a trabajar, para ello se tendrá en cuenta características tan específicas de este proyecto como son trabajar en la parte lógica con números muy grandes (4.096 bits o más) y visualizar los resultados en una interfaz gráfica intuitiva.

A continuación se deberán elegir las herramientas de trabajo que mejor integren las tecnologías usadas. Evitando usar herramientas que resulten conocidas, pero cuya funcionalidad no sea la más adecuada para trabajar con cierta tecnología.

## **4.1 Tecnologías**

Son dos las tecnologías empleadas para desarrollar este software. Para toda la parte lógica se empleará el lenguaje compilado e interpretado Java, mientras que para la parte gráfica se usará JavaFX y su lenguaje declarativo de etiquetas FXML.

### **4.1.1 Java**

Se puede decir que este será el lenguaje utilizado para realizar toda la parte Back-End de la aplicación. Su uso para implementar toda la lógica de la aplicación vendrá motivado por tres razones.

La primera es que Java tiene librerías muy potentes para realizar cálculos (exponenciación modular, multiplicaciones, máximo común divisor) con números de gran tamaño. Concretamente, tiene la clase "BigInteger" la cual será eje central en el desarrollo de la parte lógica.

La segunda es que genRSA es una aplicación orientada a ser usada en ordenadores de sobremesa o portátiles, no en móviles ni en pequeños dispositivos del Internet de las Cosas (IoT – Internet of Things).

Para que cualquier usuario pueda utilizar la aplicación necesitará tener instalado la máquina virtual de Java (JVM – Java Virtual Machine). Y para esto se necesita bastante espacio en memoria y una buena capacidad de computo, lo cual los ordenadores actuales tienen capacidad de sobra pero los móviles no.

La tercera razón es que se desarrollará toda la parte gráfica con JavaFX. Y esta, es sin duda, el motivo que hará imprescindible usar del lenguaje de programación Java. Puesto que casi el 100% de la documentación de JavaFX está vinculada con la programación en Java.

#### 4.1.2 JavaFX

JavaFX en su versión 2.0 permite diseñar y desarrollar aplicaciones de escritorio. Con este lenguaje se puede diseñar interfaces gráficas tan complejas como se necesiten, a la par que visualmente atractivas y sencillas de utilizar.

A la hora de diseñar las aplicaciones JavaFX proporciona la herramienta SceneBuilder. Esta herramienta permite generar todo el código FXML de manera declarativa evitando tener que aprender el funcionamiento de cada elemento para crear la parte gráfica de nuestro programa. No obstante, JavaFX también permite diseñar la interfaz gráfica mediante la programación.

Además, JavaFX da la posibilidad de usar Threads (hilos que permiten ejecutar código de manera concurrente). De este modo se podrán usar dos o más hilos durante la ejecución de la aplicación: un Thread para la parte gráfica, más concretamente "JavaFX Application Thread", y otro Thread para la parte lógica, que será el que realice todos los cálculos y suministre trabajo al Thread gráfico.

#### 4.1.3 FXML

Este es un lenguaje declarativo de etiquetas basado en XML. Con él se puede definir toda la interfaz gráfica empleada por el usuario de la aplicación. Se puede decir que es el lenguaje empleado para realizar la parte Front-End de genRSA v2.1.

A pesar de que la mayor parte de este código se desarrollará de manera gráfica (gracias a la herramienta SceneBuilder), también será necesario tener bastantes conocimientos de los elementos de JavaFX. Lo cual permitirá implementar una interfaz amigable y efectuar aquellas modificaciones que la herramienta no permite realizar de manera gráfica.



## **4.2 Entornos de programación**

Desarrollar una aplicación gráfica hace que sean necesarios entornos de programación muy específicos. A continuación se detallan los entornos que se usarán durante todo el desarrollo de la aplicación.

### **4.2.1 Eclipse**

Este entorno de desarrollo integrado (IDE – Integrated Development Environment) será empleado inicialmente para poder desarrollar toda la parte lógica de la aplicación. Se utilizará debido a que es un entorno que posee gran cantidad de módulos. De este modo, en el caso que durante la implementación del proyecto se necesitase desarrollar parte de la aplicación en otro lenguaje, se podrán añadir nuevos módulos que doten al entorno de nuevos marcos de trabajo (Frameworks).

### **4.2.2 NetBeans IDE**

Una vez programado toda la parte lógica, a la hora de desarrollar la interfaz gráfica con JavaFX se hará necesario cambiar de entorno.

JavaFX y SceneBuilder se encuentran peor integrados en Eclipse que en NetBeans, además Eclipse no permite depurar fiablemente las aplicaciones desarrolladas. Por ello, se decidirá cambiar a la herramienta NetBeans, la cual por defecto trae instalada todo lo necesario para tener compatibilidad entre JavaFX y SceneBuilder.

De este modo, todo el desarrollo necesario para interconectar la parte lógica de la aplicación con la interfaz de usuario se desarrollará en este entorno de programación.

### **4.2.3 SceneBuilder Gluon**

Esta herramienta permite crear escenas (lo que a nivel de usuario como se conocen como ventanas de la aplicación) mediante lo que en inglés se denomina “drag and drop” (arrastrar y soltar). Para ello, esta aplicación dispone de todos los elementos necesarios para crear las escenas: botones, paneles, contenedores, etiquetas, cajas de texto, etc.

Esta herramienta es un entorno de programación gráfico. Esto quiere decir que mientras diseñas gráficamente la escena (haciendo uso del “drag and drop”), el código FXML se va generando automáticamente. En este sentido, es muy sencilla de utilizar pero requiere altos conocimientos en interfaces gráficas para desarrollar escenas con cierto nivel de complejidad.

### **4.3 Generador de diagramas UML**

Cuando se desarrolla un software de este calibre se deben realizar diagramas UML (Lenguaje de Modelado Unificado – Unified Modeling Language) por dos motivos.

El primero motivo es para facilitar el diseño lógico del programa y el desarrollo de un código legible y de calidad. El segundo motivo es para documentar todo el software desarrollado, de modo que si otra persona debe actualizarlo le resulte más sencillo.

Para generar el diagrama se usará la herramienta Yed. Esta es una herramienta para la creación de gráficos. Como tal no dispone de ninguna herramienta que tomando como entrada el código de genRSA sepa emplear la ingeniería inversa para generar de manera automática un diagrama UML. A pesar de ello, se ha decidido disponer de ella por su facilidad en el uso.

### **4.4 Analizador de código**

Desarrollar una aplicación con tantas funcionalidades como es genRSA supone que el desarrollo de su código pueda verse afectado, en ocasiones, por estilos de programación poco recomendados.

Por este motivo, se hará uso de la herramienta SonarQube. Este programa es un analizador de código que permite identificar fallos en la programación mediante el análisis estático del código. Con este software se puede evitar entre otros: líneas duplicadas en el código, clases con demasiada complejidad ciclomática y falta de documentación de alguna clase.

## **4.5 Repositorio de código y documentación**

GitHub será la plataforma elegida para usar como repositorio del proyecto. El motivo principal para elegir esta plataforma, y no cualquier otra nube donde subir el proyecto, será que esta está orientada al control de versiones de código.

Además toda la información almacenada en la cuenta de este servicio será totalmente pública. Pudiendo servir una parte o la totalidad del código de genRSA a cualquier persona que lo requiera. Otra ventaja de que el proyecto sea almacenado en un servidor de acceso público, es que cualquiera puede hacer una auditoría al código y mejorarlo o simplemente dar fe que no es un código malicioso.

Por otro lado, en GitHub se optará por tener dos versiones del proyecto. En la versión de Desarrollo, se almacenarán los desarrollos de las nuevas funcionalidades y se realizarán pruebas de implantación. Mientras que la versión de Producción, solo se almacenarán partes del proyecto que estén ampliamente probadas y no comprometan la estabilidad de lo almacenado con anterioridad.



# Capítulo 5

## Requisitos mínimos del sistema

---



La aplicación genRSA v2.1 no es una aplicación que se pueda considerar pesada. A pesar de ello, existen unos requisitos mínimos necesarios para la correcta ejecución del programa. A continuación los analizaremos dividiendo los principales en secciones.

## **5.1 Espacio en Disco**

El tamaño del archivo que contiene la aplicación no supera los 5 MB, por lo cual no es necesario tener una gran cantidad de espacio en disco para almacenarla. No obstante, para poder ejecutar la aplicación es necesario tener instalada la Máquina Virtual de Java (JVM).

En la versión 8 actualización 131, la máquina virtual de Java necesita alrededor de 300MB (MegaBytes) para instalarse. Esta cifra no supone un problema para ningún ordenador fabricado durante los últimos 4 años, puesto que el tamaño más normal de disco es de 500GB (GigaBytes).

## **5.2 Memoria RAM**

La ejecución de genRSA supondrá un alto consumo de memoria RAM en determinadas situaciones. Siendo 250 MB la cifra mínima de memoria RAM para ejecutar acciones sencillas tales como generación de claves, tests de primalidad u operaciones de cifrado y firma. Por otro lado, para ejecutar acciones más complejas como son los ataques se puede llegar a necesitar como máximo hasta 1.1 GB de RAM.

No obstante, si los ataques se realizan para claves pequeñas (alrededor de 40 bits) generalmente se necesitan aproximadamente 400 MB de RAM para su ejecución.

Si se analiza la cifra máxima de memoria RAM requerida 1.1GB, se puede sacar la conclusión de que la cifra un poco elevada.

Normalmente, los ordenadores portátiles suelen traer menos memoria RAM que los ordenadores de sobremesa y aun así la cifra más habitual son 4 GB de RAM. Si se necesitan puntualmente 1.1 GB de RAM puede ser necesario cerrar otros programas para aprovechar al máximo las prestaciones de genRSA v2.1.

No obstante, los casos en los que se necesite tanta memoria serán ocasiones puntuales en los que el mayor cuello de botella se encontrará en la potencia de cálculo del procesador.

### **5.3 Resolución gráfica**

GenRSA v2.1 tiene un tamaño predeterminado de ventana principal de 900x600 píxeles y puede adaptarse a cualquier tamaño superior de pantalla organizando los elementos de manera armoniosa.

Sin embargo, para adaptarse a tamaños inferiores de pantalla existe un límite (600x600 píxeles) por debajo del cual, los elementos no se pueden comprimir más en la ventana y se comienza a perder información de la misma.

En cuanto a las ventanas secundarias (ataques, información, errores, cifra y firma), cada una de ellas tiene un tamaño distinto pero todas ellas tendrán un tamaño menor que 600x600 píxeles. Y al igual que ocurre con la ventana principal, podrán adaptarse a distintos tamaños de pantalla respetando la posición de todos sus elementos.

Se puede concluir diciendo que la aplicación podrá ser visualizada de forma correcta en la gran mayoría de pantallas existentes en el mercado.



# Capítulo 6

## Librerías

---



Durante todas las etapas del desarrollo del proyecto, se adquirirán multitud de conocimientos. Y sobre los conocimientos que ya se tenían una base se profundizarán. Estos abarcarán aspectos teóricos como pueden ser los algoritmos de factorización, de generación de claves o de cálculo de números no cifrables.

También se adquirirán conocimientos lógicos y funcionales como es el funcionamiento de una interfaz gráfica y las partes de las que consta. Se aprenderá en gran medida de todas las tecnologías y librerías necesarias para la implementación de la aplicación.

Las librerías más destacadas en el desarrollo de genRSA serán empleadas en el desarrollo de las partes lógica y gráfica así como en la implementación de la concurrencia.

## **6.1 Parte lógica**

La parte lógica será la primera que se comenzará a desarrollar. Más concretamente, la parte de la generación de claves RSA. En este sentido, se buscarán librerías ya existentes que generen las claves automáticamente y muestren en claro (sin cifrar) todos los valores de los componentes de la clave.

En general, la mayoría de librerías son interfaces de programación de aplicaciones (API – Application Programming Interface) que estan más enfocadas a encriptar información o crear certificados que ha generar claves para su estudio y explotación (que es lo que realmente pretende genRSA v2.1). De acuerdo con lo dicho, caben destacar dos librerías no oficiales: Jasypt<sup>1</sup> y Legion of the Bouncy Castle<sup>2</sup>.

Dentro de las librerías oficiales de Java se pueden encontrar multitud de paquetes que podrán resultar útiles. El algoritmo RSA pertenece a la criptografía asimétrica y en Java existe una arquitectura criptográfica (JCA<sup>3</sup> – Java Cryptography Architecture) que permite integrar características de seguridad al desarrollo de aplicaciones. Los paquetes principales de esta arquitectura son “java.security<sup>4</sup>” y “javax.crypto<sup>5</sup>”. Pero como en el caso de las librerías no oficiales, la mayor parte de las clases de estos paquetes están orientados a la encriptación de información, a la generación de certificados y al almacenamiento seguro de claves. Dejando en un segundo plano la generación de claves.

Debido a que no se encontrará una librería que genere las claves de manera automática y con las características que la aplicación requiere, se implementará toda la parte lógica de forma manual. Haciendo uso únicamente de la clase BigInteger (paquete "java.math").

## **6.2 Parte gráfica**

Para desarrollar la parte gráfica resultará sencillo decidir entre todas las posibles librerías y tecnologías. JavaFX será la elegida principalmente por dos motivos.

El primero y más importante es que permite interconectar la parte gráfica con la parte lógica usando el lenguaje de programación Java. Esto facilitará mucho la interconexión incorporando pequeños fragmentos de código a la parte lógica.

Por tanto, si no se usa una herramienta que permita conectar ambas partes mediante el lenguaje Java habrá que adaptar todo el código de la parte lógica de forma que genere salidas que pueda comprender la parte gráfica (escrita en otro lenguaje). Este sería el caso si se utiliza una herramienta de tanto prestigio como Qt Creator<sup>6</sup>, la cual hace uso del lenguaje C++ .

El segundo motivo es que en Java solo existe una herramienta para crear de forma gráfica una interfaz de usuario. Y esta herramienta, SceneBuilder, utiliza JavaFX en lugar de otra tecnología como podría ser Swing.

El poder usar una herramienta que permita ver en tiempo real el aspecto de tu aplicación facilita bastante esta labor. Puesto que desarrollar una interfaz gráfica, en cualquier lenguaje de programación, es una tarea bastante compleja y más aún cuando se tiene que hacer escribiendo código.

## **6.3 Concurrencia**

El desarrollo en JavaFX de aplicaciones con procesos que requieren mucho tiempo de computo hacen que inevitablemente la interfaz gráfica deje de responder hasta que se termina dicho proceso.

Por esta razón, la concurrencia dentro de genRSA será necesaria para poder paralelizar la parte gráfica y la parte lógica. Pero implementarla en JavaFX no será un proceso tan sencillo como podría ser en Java.

Dado que JavaFX implementa su propio Thread se estudiará el funcionamiento de la concurrencia en JavaFX<sup>7</sup>, el cual está definido en el paquete "javafx.concurrent".

Dada la complejidad y poca flexibilidad que supondrá usar la "Worker Interface" de este paquete se optará por usar la clase Thread de Java (paquete "java.lang") en combinación con el Thread propio de JavaFX.

De este modo, toda parte paralelizada en genRSA utilizará un Thread para ejecutar procesos cuyo tiempo de computo sea elevado. Y este mismo será a su vez quien suministre tareas al Thread propio de JavaFX para que muestre gráficamente los resultados en tiempo real.



# Capítulo 7

## Desarrollo de la Aplicación

---





El desarrollo lógico de la aplicación duró desde la primera semana de Febrero hasta la primera de Junio, es decir, cuatro meses. Pero quitando los días dedicados al estudio por exámenes de otras asignaturas del grado y los días dedicados a la realización de otros trabajos, el tiempo real de desarrollo de la aplicación fue de aproximadamente tres meses. Para más detalle se recomienda ver el anexo 1.

Durante estos tres meses, se realizaron tareas de aprendizaje, diseño gráfico de la aplicación, desarrollo lógico y pruebas de funcionalidad. En este capítulo se van a detallar las tareas más importantes orientadas a hacer la aplicación más didáctica y fácil de usar.

## **7.1 Diseño Gráfico**

En esta sección se comentará el diseño de cada una de las ventanas que se muestran en la aplicación.

Las primeras seis ventanas se han desarrollado con la herramienta gráfica SceneBuilder usando el lenguaje FXML. En esta herramienta las ventanas son llamadas "Escenas" y tienen una estructura jerárquica. Todas ellas se han creado para que se pueda modificar su tamaño durante la ejecución de la aplicación y que los elementos se acomoden de manera armónica por la pantalla.

Las tres ventanas restantes se han desarrollado de manera programática, es decir, escribiendo el código para su creación. Estas tres ventanas tienen tres nombres específicos: preloader, diálogo de error y diálogo de información.

Por último, cabe destacar un tipo de ventana que se abrirá cuando se tenga que elegir una ubicación donde guardar o seleccionar un archivo. Esta ventana tendrá el aspecto propio de cada sistema operativo y será muy parecida a un explorador de archivos.

### 7.1.1 Ventana principal genRSA

Es la escena principal y representa de una manera unívoca al programa genRSA.

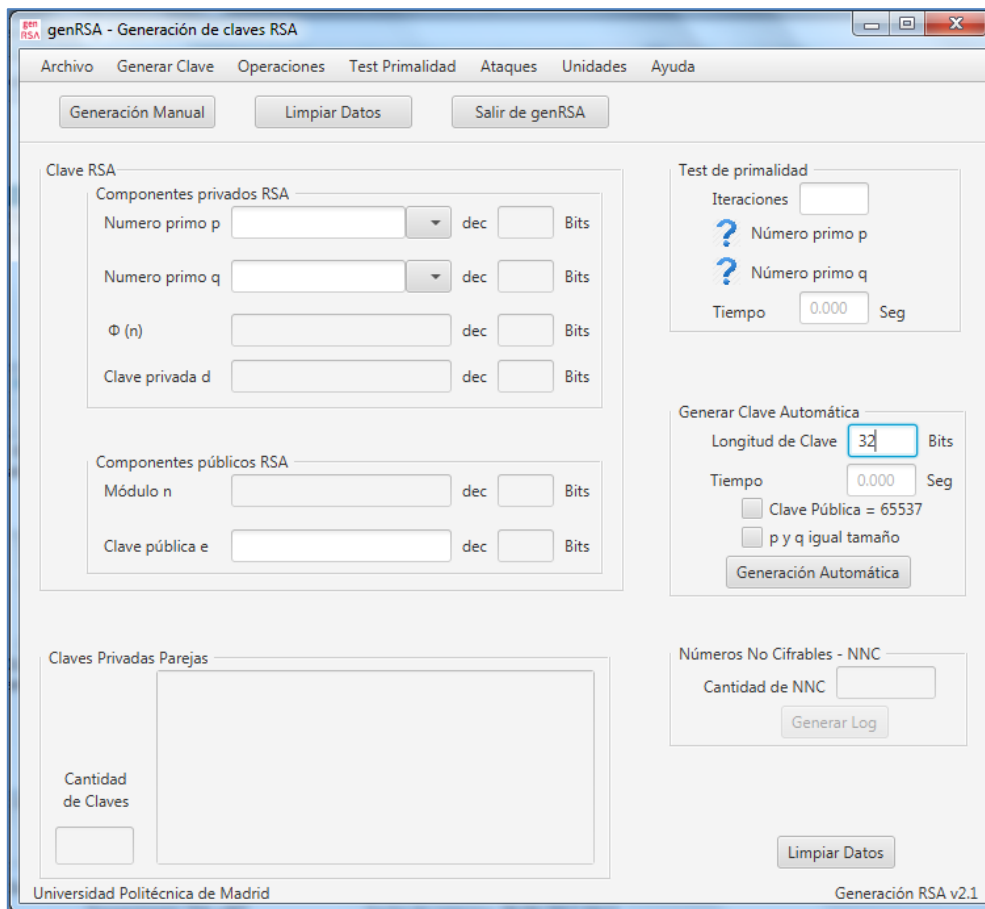


Imagen 1: Escena principal minimizada genRSA v2.1

Se van a analizar las mejoras realizadas durante la actualización con respecto a la antigua versión (ver imagen de genRSA v1.0 en anexo 2). Mostradas de arriba abajo y de izquierda a derecha sobre la imagen 1, las mejoras gráficas son las siguientes:

- El primer cambio que se puede apreciar es que se ha modificado el icono de aplicación. El nuevo icono tiene la misma tipografía que el icono original de la compañía RSA.



Imagen 2: Icono aplicación genRSA v2.1

- Se ha añadido información en todos los botones de la barra de menú acerca de los atajos de teclado para facilitar la ejecución del programa.

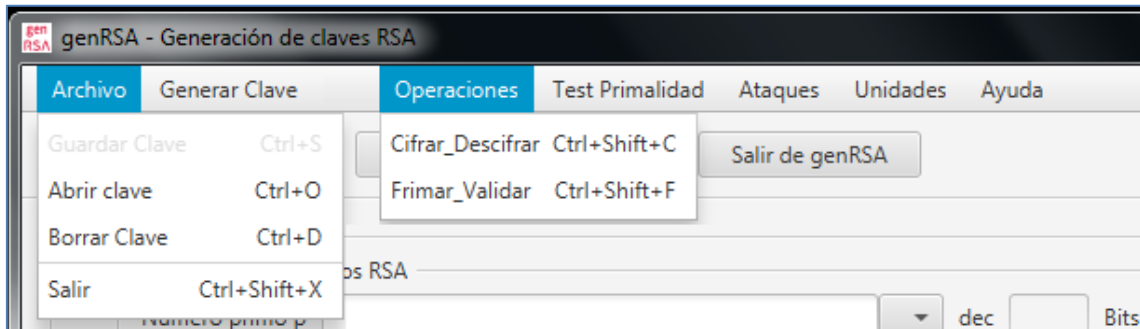


Imagen 3: Ampliación zona barra de menú de la ventana principal

- En la barra de menú en la pestaña de operaciones ahora se encuentran dos botones: "Cifrar\_Descifrar" y "Firmar\_Validar", como se puede observar en la Imagen 3.
- Justo debajo de la barra de menú se añade el botón "Limpiar datos" que complementa al mismo botón situado en la parte inferior derecha de la ventana principal.
- Se han añadido desplegados junto a las cajas de texto de los números primos. Estos permiten seleccionar números primos seguros para la generación manual de claves.
- En la zona inferior izquierda se han modificado etiquetas que facilitaran la comprensión de los resultados mostrados: la etiqueta "Claves Parejas" pasa a denominarse "Claves Privadas Parejas", evitando de este modo la confusión con las claves públicas parejas, y "Nº claves" pasa a denominarse "Cantidad de claves".
- En la zona superior derecha, se han modificado las imágenes mostradas cuando se ejecutan los test de primalidad. Estos cambios son para dar una estética más actual a la aplicación.

A la izquierda de la imagen 4 se puede observar la imagen de interrogación haciendo referencia a que el test no se ha ejecutado y por tanto no se sabe si los primos  $p$  y  $q$  son primos. A la derecha de la imagen, se puede ver una ejecución del test de primalidad donde el número  $p$  sí que es primo mientras que el número  $q$  no lo es.

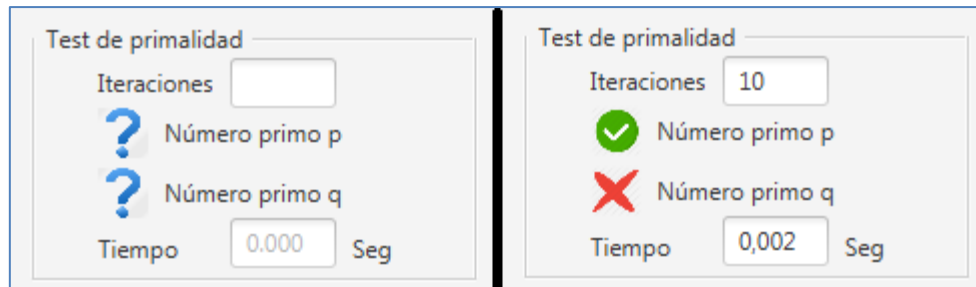


Imagen 4: Zona test de primalidad sin ejecutar y ejecutada

- En la zona de generación automática se ha añadido una caja (CheckBox) para seleccionar un nuevo modo de generación automática. Este nuevo modo genera una clave privada tomando como clave pública por defecto el valor 65.537.
- En la zona inferior derecha, también se han modificado las etiquetas para mejorar la comprensión. Las etiquetas modificadas son: "Mensajes no Cifrables" por "Números no Cifrables – NNC" y "Nº mensajes" por "Cantidad de NNC".

- Por toda la pantalla se ha mejorado la información de los llamados “tooltips”. Los tooltips son la información que se muestra cuando se deja unos segundos el ratón encima de algún elemento. En la imagen 5 podemos ver una composición de imágenes superpuestas de la pantalla principal en las que se muestran algunos de estos tooltips.

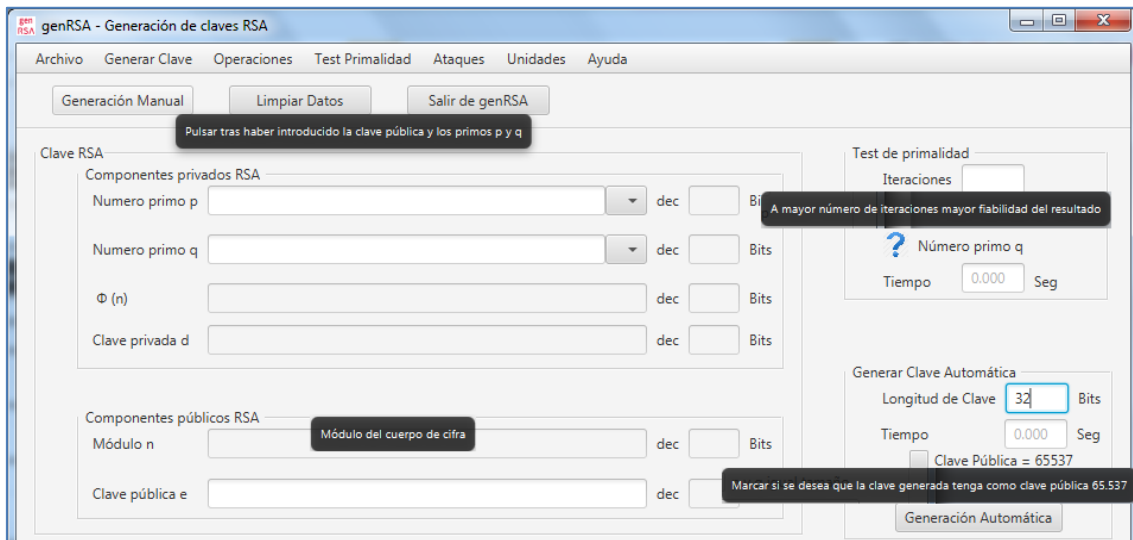


Imagen 5: Ventana principal con varios tooltips.

- Finalmente, cabe destacar la actualización más importante en el diseño de la ventana principal. Esta actualización no es otra que permitir que la ventana pueda ser modificada en tamaño para que se adapte a cualquier resolución de pantalla. Si comparamos la imagen 1 o la imagen 5 con la que se muestra a continuación se puede apreciar como varios de los elementos de la pantalla principal han aumentado su tamaño para adaptarse a la nueva dimensión de pantalla.

De este modo se facilita la visualización de claves de longitud elevada.

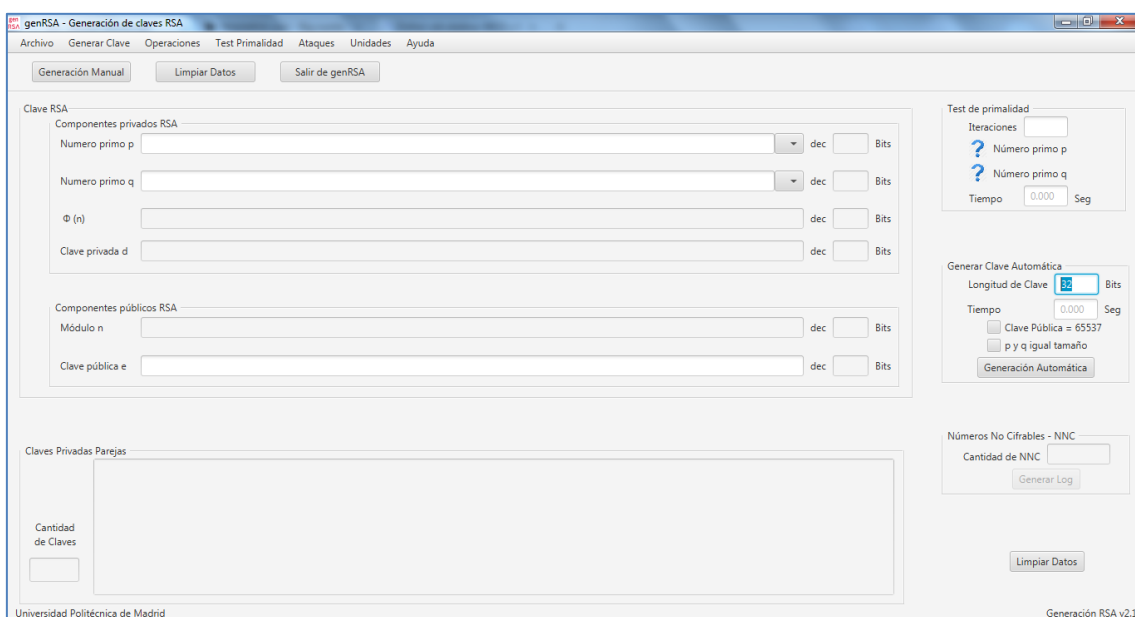


Imagen 6: Ventana principal mostrada en pantalla con resolución 1366x768.

### 7.1.2 Ventanas secundarias

Las ventanas secundarias son cinco: cifrado y descifrado, firma y validación, ataque cíclico, ataque por la paradoja del cumpleaños y ataque por factorización. Todas estas pantallas tienen en común que han sido rediseñadas desde cero.

Para su actualización se ha tenido en cuenta que genRSA es una aplicación didáctica que tiene que servir como complemento para el estudio del algoritmo RSA en la asignatura de Seguridad de la Información. Por esta razón en estas pantallas se muestran botones de información y todos los componentes de la clave necesarios para realizar el ataque, la firma o la cifra. Los botones de información muestran un diálogo explicando cómo se realiza la acción requerida.

Otra característica de la actualización que comparten todas estas pantallas es que la vista de la ventana está dividida en dos, pudiéndose modificar el tamaño de ambas gracias a un deslizador central. En el caso de los ataques nos permitirá mostrar en un lado la información necesaria para realizar el ataque y en el otro los resultados del mismo. Mientras que para el caso del cifrado-descifrado y firma-validación, nos permitirá mostrar la información necesaria para cada operación recalcando la clave que se usa en cada una de ellas.

A continuación se detallan algunos aspectos de diseño de cada una de las ventanas secundarias.

### Ataque Cíclico

Para diseñar esta ventana se combinó toda la funcionalidad que ya poseía la ventana en la versión antigua y se añadió la funcionalidad de poder realizar el ataque hasta que prospere, que es propia de RingRSA.

Una vez concretadas todas las funcionalidades, se rediseñó la ventana de tal manera que se mostrase toda la información necesaria de una forma sencilla y fuera visualmente fácil de utilizar. Para conseguir el fin didáctico, se añadieron los campos módulo y exponente y el botón de información. Permitiendo de esta manera, entender al momento como funciona el ataque cíclico.

The screenshot shows a software window titled "genRSA - Ataque Cíclico". The window is divided into two main sections. The left section, titled "Ataque Cíclico", contains input fields for "Nº de Cifrados" (set to 10), "Módulo n", "Exponente", "Mensaje Cifrado", and "Mensaje Original". There is a checkbox labeled "Hasta que prospere". Below these fields are four buttons: "Comenzar", "Continuar", "Información", and "Limpiar Datos". The right section, titled "Resultados", contains a large empty text area for output. At the bottom right of this section, there are two labels: "M. Original Recuperado" and "Tiempo Total" (set to 0.000 Seg).

Imagen 7: Ventana Ataque Cíclico



### Ataque por la Paradoja del Cumpleaños

Una vez que se diseñó la primera ventana secundaria, la de Ataque Cíclico, se mantuvieron la mismas propiedades para desarrollar el resto.

Dadas las características del ataque por la paradoja del cumpleaños se limitó las opciones de iniciar el ataque. En este caso, solo se puede iniciar el ataque de tal manera que no parará hasta que se le indique o encuentre el resultado del ataque.

Este ataque para una clave de 40 bits puede llegar a realizar más de 250 mil millones de cifrados. Para poder visualizar la rapidez de computo del equipo donde se ejecuta el programa, se incorporó una caja en la cual se muestra la información de la media de cifrados por segundos.

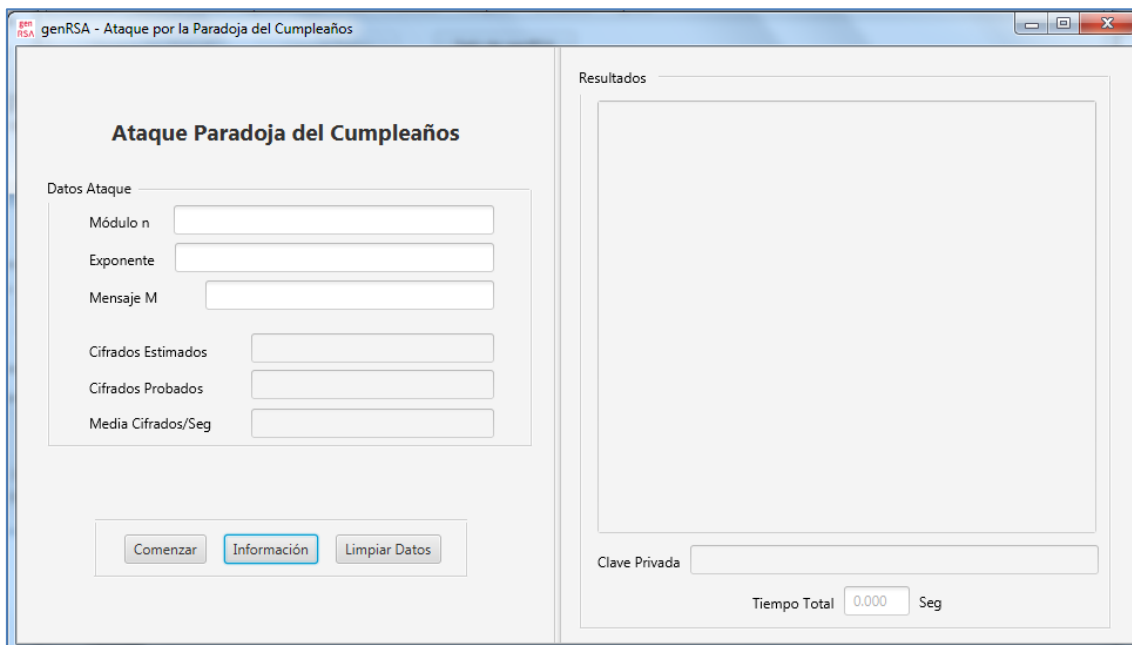


Imagen 8: Ataque por la Paradoja del Cumpleaños

### Ataque por Factorización

Esta ventana de ataque es de las que menos modificaciones ha sufrido con respecto a la versión 1.0. En rasgos generales, la ventana conserva la misma apariencia pero gana en funcionalidad.

Al igual que las ventanas anteriores, también puede ser maximizada haciendo que sus elementos se repartan de forma armónica por el resto de la pantalla manteniendo la simetría.

genRSA - Ataque por Factorización

**Factorización Pollar Rho**

Nº de Vueltas  ☐ Obtener p y q

Factorización

N = p \* q

Primo p

Primo q

Comenzar Continuar

Información Limpiar Datos

Resultados Parciales

Tiempo Total  Seg

Imagen 9: Ataque por Factorización

### Cifrado y Descifrado

Esta ha sido una de las ventanas que ha sufrido mayores modificaciones en el diseño. Esto ha sido debido a que a los alumnos que utilizaban este software no les quedaba claro que clave (pública o privada) es la que se usa para el cifrado y que clave se usa para el descifrado.

Gracias a este rediseño ya no habrá ninguna confusión. Ahora aparece con letras grandes el tipo de clave usada al lado de la operación a realizar. También muestra la clave a usar y en el caso de descifrado te deja elegir entre la clave privada o cualquiera de las claves parejas.

Cabe destacar que para ambas operaciones, cifrado y descifrado, permite la introducción de datos, característica que en la antigua versión solo ofrecía la parte de cifrado.

The image shows a software window titled "genRSA - Cifrado y Descifrado". It is divided into two main sections: "Cifrado - Clave Pública Destinatario" on the left and "Descifrado - Clave Privada Destinatario" on the right. Each section contains input fields for "Módulo", "Clave Pública/Privada", and a large text area for "Datos Originales/Cifrados". There are checkboxes for "Tienen texto los Datos Originales/Cifrados". Below the input fields are buttons for "Cifrar Datos", "Información", and "Limpiar Datos".

Imagen 10: Cifrado y Descifrado

## Firma y comprobación de firma

Esta ventana no tiene una correspondiente en la versión 1.0 de genRSA, por lo tanto se puede decir que es una nueva funcionalidad.

El diseño de esta ventana es el mismo que para la ventana de Cifrado y Descifrado pero cambiando de izquierda a derecha la posición de la clave privada y la clave pública.

En la imagen 11 podemos ver desplegado la caja ("ComboBox") de selección de clave privada para realizar la operación de firma.

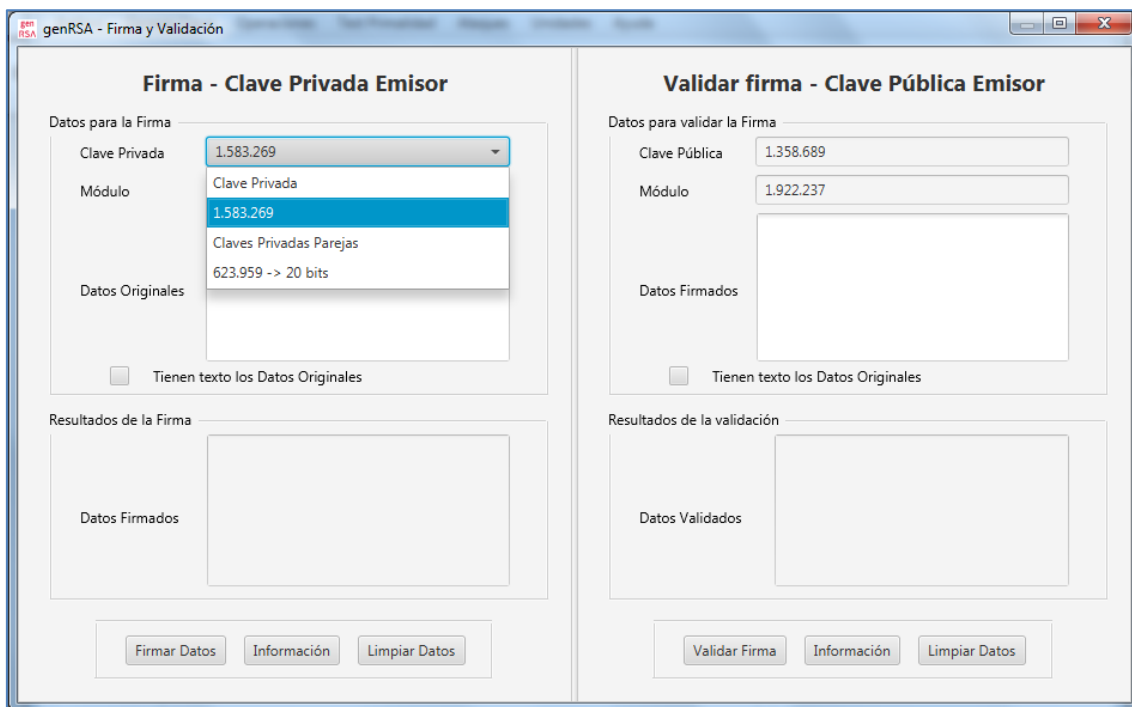


Imagen 11: Firma y Comprobación

### 7.1.3 Ventana Preloader

Es una ventana que sirve para modificar la experiencia de arranque de la aplicación. Esta ventana se mostrará mientras se carga la aplicación genRSA evitando que el usuario se irrite mientras espera a que aparezca la ventana principal. Más adelante, en el apartado 8.1 se desvelará una característica lógica muy importante de este Preloader.

En esta ventana se puede apreciar el logo de la aplicación, una breve descripción de la misma y una barra de progreso que indica el estado de la carga.

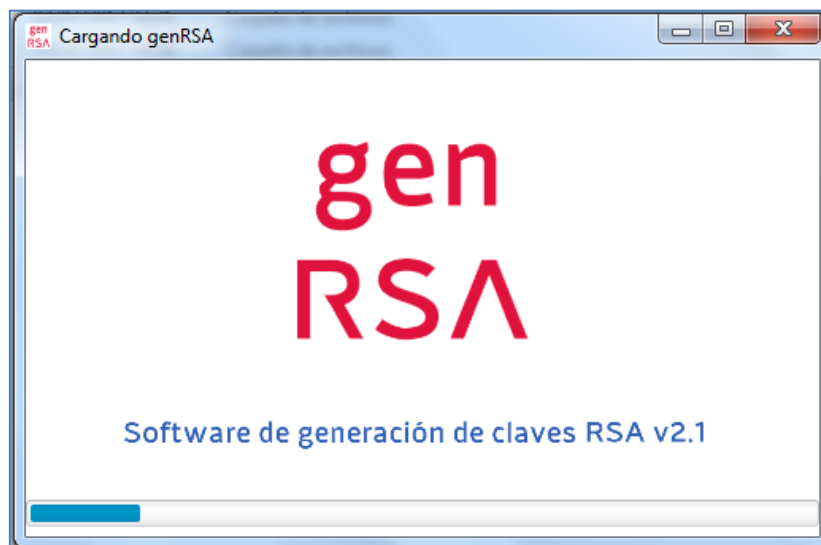


Imagen 12: Preloader genRSA v2.1

### 7.1.4 Diálogos de Información

Estos diálogos son ventanas que se muestran gráficamente de una manera similar que los “pop-ups”. El objetivo de estos es informar acerca de cómo se realiza una operación o de un ataque.

Estos diálogos se han programado con el objetivo de mostrar la información de un modo no intrusivo. Demasiados mensajes de información y muy repetitivos, en vez de ayudar molestan al usuario. En genRSA v2.1 la mayoría de ellos solo aparecerán cuando se pulse un botón de información, como es el caso del diálogo mostrado en la imagen 13.

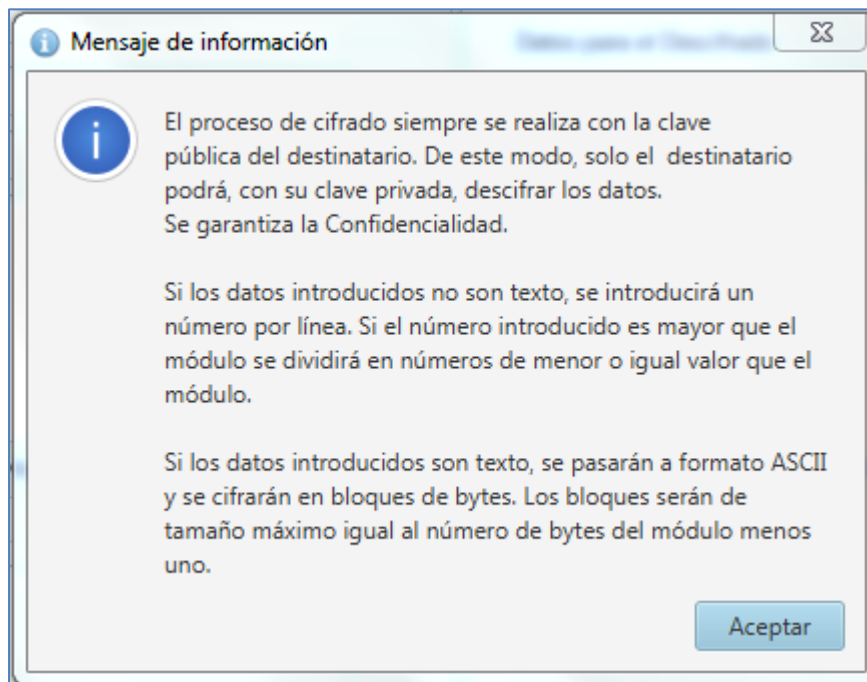


Imagen 13: Diálogo de información mostrado en la operación de Cifrado.

Los diálogos de información también se usan para informar cuando se realiza alguna modificación que pueda hacer que el programa no cumpla con su finalidad.

En la imagen 14 podemos observar el diálogo que aparecerá cuando modifiquemos el módulo o el exponente en el ataque cíclico. Como he comentado anteriormente, estos diálogos no son intrusivos. Ello implica que solo se mostrarán la primera vez que modifiquemos uno de los dos, el resto de veces que se modifiquen no volverán a aparecer.

De esta manera, evitamos irritar y molestar al usuario teniendo que cerrar ventanas continuamente, como en ocasiones ocurría con la versión 1.0 de la aplicación.

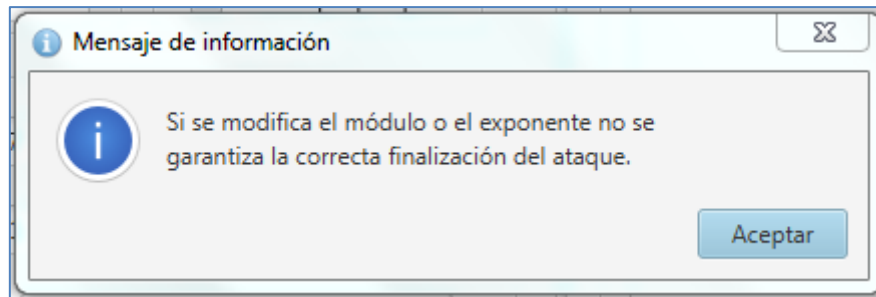


Imagen 14: Diálogo de información mostrado al modificar módulo o exponente en la ventana de ataque Cíclico

### 7.1.5 Diálogos de Error

Estos diálogos tienen un comportamiento muy similar a los diálogos de información. Ambos fueron desarrollados sin usar ninguna herramienta gráfica, es decir, se escribió su código<sup>8</sup>.

La diferencia entre los dos tipos de diálogos está en que los de este apartado tienen como objetivo informar acerca de errores producidos a la hora de introducir los datos. Es por esta razón que los diálogos de error sí que pueden resultar algo intrusivos para usuarios con poca experiencia en el algoritmo RSA.

Lejos de que esta intrusividad pueda resultar negativa, los diálogos de error se han programado para que tengan una cabecera que indique por qué se ha producido el error y un cuerpo en el que se ofrezca una solución a dicho error. De este modo, los diálogos podrán ayudar a recordar o incluso enseñar nociones básicas del uso del algoritmo RSA.

La imagen 15 se muestra un caso en el que aparece un Diálogo de Error. Este se produce cuando se pulsa el botón de guardar una clave, o de generar el log de los números no cifrables, y una vez se ha abierto la ventana para indicar el lugar donde guardar el fichero, esta se ha cerrado sin indicar el nombre del archivo ni la ubicación.

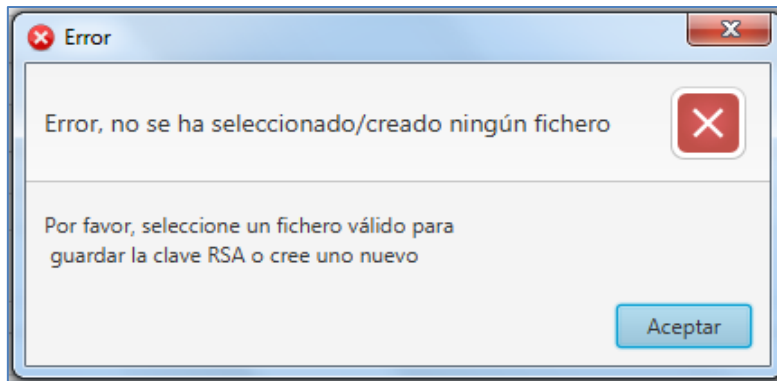


Imagen 15: Diálogo de error en creación de fichero

Otro ejemplo es la imagen 16. En ella se muestra el diálogo que aparece en el caso de haber introducido un valor par en la caja de texto del primo  $p$  o primo  $q$  y, a continuación, haber pulsado el botón de alguno de los dos test de primalidad (Fermat o Miller Rabin). Como no tiene sentido haber introducido un número par y comprobar su primalidad se entiende que se ha introducido un número de forma errónea o se ha olvidado algún dígito al final del mismo.

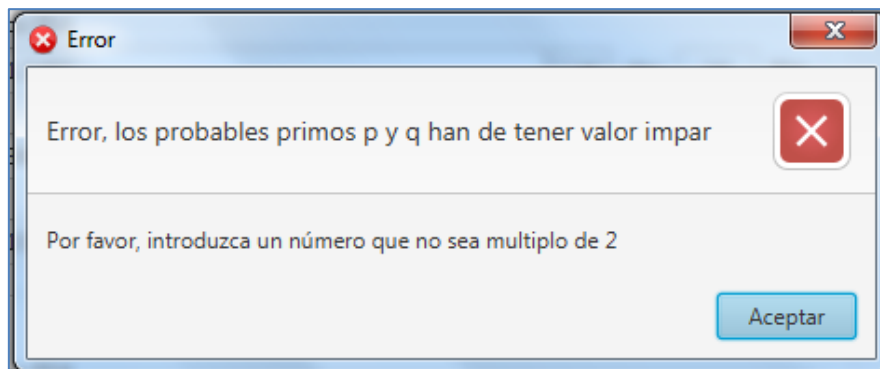


Imagen 16: Diálogo de error en Test de Primalidad.



## **7.2 Desarrollo Lógico**

Si bien el diseño y desarrollo de la parte gráfica de la aplicación se puede realizar con relativa independencia de la parte lógica, no se puede decir que suceda lo mismo con la relación inversa: el desarrollo de la parte lógica de la aplicación no se ha podido realizar con independencia de la parte gráfica. Esto es así debido a que la interconexión entre ambas partes se implementó mientras se desarrollaba el código de la parte lógica.

El desarrollo de toda la lógica e interconexión de la aplicación se llevó a cabo en cuatro fases.

### **7.2.1 Primera Fase: funcionalidades básicas**

La primera fase fue el desarrollo de la funcionalidad de generación de claves. En esta se incluye: el test de primalidad, la generación de claves privadas parejas, el cálculo de los números no cifrables, una pequeña aproximación a los ataques cíclico y por paradoja del cumpleaños y por último la generación manual y automática de claves RSA.

Para implementarlo, en primer lugar fue necesario recordar todos los conocimientos aprendidos acerca del algoritmo RSA en la asignatura Seguridad de la Información. En segundo lugar, fue necesario documentarse acerca del uso de todas las librerías expuestas en el capítulo 6. Una vez documentado se decidió usar la clase BigInteger del paquete "java.math", puesto que era la única clase que permitía desarrollar un código en el que se obtuvieran todos los componentes de la clave (particularmente, las claves privadas parejas y los primos  $p$  y  $q$ ). Otra de las razones fundamentales fue que esta clase de Java es de las pocas en todos los lenguajes de programación que permite generar claves de tamaño mayor que 8.192 bits. O lo que es lo mismo, claves de más de 2.466 dígitos decimales. Para poder hacerse una idea de lo que supone una clave de estas características, en el anexo 3 se expone el módulo de una clave de 8.192 bits.

Para finalizar esta fase, se realizaron todas las pruebas para comprobar el correcto funcionamiento del código desarrollado. Siendo el test de Miller Rabin el único método que presentaba ciertos fallos cuando se ejecutaba para claves de más de 40 bits. Este problema se solucionó en fases posteriores.

### 7.2.2 Segunda Fase: funcionalidades añadidas

Esta fase se desarrolló tras haber diseñado toda la parte gráfica de la aplicación. Básicamente, esta fase se puede dividir en dos etapas.

El esfuerzo de aprendizaje de la primera etapa se compartió en gran medida con el diseño de la parte gráfica. Esto es así, puesto que en la documentación de JavaFX<sup>9</sup> y SceneBuilder había información que era necesaria tanto para la parte gráfica como para la interconexión de la parte lógica con la gráfica.

Esta primera etapa supuso un gran cambio a la hora de programar. Ya no solo se usaba Java sino que también se usaba JavaFX, por ello había que tener dos nuevos aspectos en cuenta. El primero era que al paralelizar tareas había que ser consciente de que JavaFX usaba su propio Thread para la parte gráfica(más adelante, en la clase genRSAController del apartado 8.2.2 se explica con más claridad). El segundo aspecto a tener en cuenta era que ya no solo se trataba de desarrollar la parte lógica sino que a su vez se debería incorporar el código necesario para mostrar los resultados gráficamente en las ventanas diseñadas anteriormente.

La segunda etapa estuvo enfocada al desarrollo de la lógica de las ventanas secundarias. Se implementaron todas las funcionalidades correspondientes a: ataque cíclico, ataque por la paradoja del cumpleaños, ataque por factorización, operación de firma y operación de cifra. También se implementaron los diálogos de información y error de toda la aplicación.

En la segunda etapa fueron dos las tareas que cobraron especial importancia en la implementación:

- Desarrollo de un código que no hiciese que la parte gráfica no respondiese cuando se ejecutasen tareas cuyo tiempo de ejecución fuese prolongado. Para ello se hizo que el Thread de JavaFX (parte gráfica) estuviese coordinado con el Thread de la parte lógica. Lo cual permitía que la información mostrada fuese en tiempo real y sin fallos por falta o duplicación de resultados.
- Creación de ventanas secundarias manteniendo la ventana principal en segundo plano y conservando la información de esta última. Esto se consiguió creando otro escenario a parte del escenario que mostraba la ventana principal. Este nuevo escenario se creó siendo dependiente del escenario principal.

### 7.2.3 Tercera Fase: presentación y modificaciones.

La tercera fase consistió en depurar pequeños errores que aparecían con el uso de la aplicación. La mayoría de estos eran pequeños errores gráficos como palabras sin acentuación, mensajes de error cuyo contenido se podía mejorar para facilitar la comprensión del mismo y la información mostrada en los resultados de los ataques.

Además se hizo uso de la aplicación SonarQube para depurar pequeños errores en la programación. Gracias a ella, se disminuyó el número de líneas duplicadas y la complejidad de los métodos.

En esta fase, también se mantuvo una reunión con el tutor del proyecto. En ella el tutor aportó ideas notorias acerca de mejoras que podían realizarse en la aplicación para facilitar su uso. Estas mejoras fueron:

- Mejorar la información mostrada en los resultados de los ataques. Por ejemplo, en el ataque por la paradoja del cumpleaños el resultado en unas ocasiones mostraba números de más de tres cifras separados por un punto y en otras ocasiones esa separación no existía. Esto podía ocasionar dificultad en la comprensión del resultado.
- Proveer a las ventanas de todos los ataques con un botón para pausar la ejecución del mismo. Esta fue una mejora fundamental dado que si se comenzaba una tarea que durase días no podría pararse hasta que terminase o se matase al proceso.

Actualmente si se comienza un ataque este se podrá parar de dos maneras: cerrando esa ventana o pulsando el botón de parar. No obstante, esta función no solo se ha añadido a los ataques si no a todas aquellas tareas que puedan ser susceptibles de prolongarse demasiado tiempo en su ejecución.

- Mejorar la implementación del test de primalidad de Miller Rabin. Como comentaba en la primera fase, el test de primalidad no siempre funcionaba correctamente a pesar de haberse implementado de distintas formas el algoritmo. Finalmente, este test se decidió cambiar por el test de primalidad que ofrece la clase BigInteger el cual es una implementación del algoritmo de Miller Rabin y Lucas-Lehmer.

#### **7.2.4 Cuarta Fase: depuración.**

En una aplicación de esta escala es fácil que con el paso del tiempo se vayan encontrando pequeños fallos o acontezcan nuevas ideas para mejorar las funcionalidades de la aplicación.

En general, suelen ser modificaciones menores que pueden ayudar a mejorar aspectos como la rapidez de la aplicación, la facilidad de uso, el aspecto gráfico del programa o la comprensión del código.

# Capítulo 8

## Aspectos destacados de la programación

---



Este capítulo se dedicará a mostrar aquellos aspectos del código que deben ser recalcados. Ya sea debido a que su desarrollo ha supuesto un mayor esfuerzo o a que su implementación ha supuesto una mejora en las funcionalidades del programa.

Todo el código se ha desarrollado haciendo uso de la modularidad. Gracias a ello se ha podido reutilizar gran parte del código bien sea a la hora de compartir métodos o a la hora de desarrollar unos nuevos.

Esta modularidad también queda reflejada en los nombres de la paquetería empleada. Los siguientes apartados de este capítulo tendrán el mismo nombre que los paquetes donde se almacenan las clases Java.

## **8.1 Paquete PreloaderGenRSA**

El preloader es la ventana que se muestra al iniciar la aplicación. Esta ventana (ver imagen 12), se desarrolló programáticamente incluyendo y alineando imágenes dentro del contenedor de tipo "AnchorPane" que forma la ventana.

Lo más complejo fue añadir una barra de progreso y que se rellenase del 0 al 100 por ciento. Además, una vez que la barra llegase al 100% esta ventana debía desaparecer para dar paso a la ventana principal de la aplicación. Ambas funciones se lograron sobrescribiendo los métodos "handleProgressNotification" y "handleStateChangeNotification" respectivamente .

```

@Override
public void handleStateChangeNotification(StateChangeNotification scn) {
    if (scn.getType() == StateChangeNotification.Type.BEFORE_START) {
        stage.hide();
    }
}

@Override
public void handleApplicationNotification(PreloaderNotification preN) {
    if (preN instanceof ProgressNotification) {
        ProgressNotification pn= (ProgressNotification) preN;
        progressBar.setProgress(pn.getProgress());
    }
}
  
```

Imagen 17: Fragmento de código de la clase PreloaderGenRSA.java

No obstante, para rellenar la barra de progreso acorde con el estado de carga de la aplicación, se tiene que poner en contacto la clase GenRSA (del paquete genRSA) con el método "handleProgressNotification" del preloader. La clase genRSA también indicará al preloader cuando debe ocultar su ventana.

Para mandar estas notificaciones al preloader desde la clase genRSA se sobrescribió el método "init".

```
@Override
public void init() throws Exception {
    double progress =0.0d;

    while (progress<0.98){
        progress=progress+0.015d;
        notifyPreloader(new ProgressNotification(progress));
        Thread.sleep(50);
    }
    notifyPreloader(new ProgressNotification(1.0d));
    Thread.sleep(190);
    notifyPreloader(new StateChangeNotification(StateChangeNotification.Type.BEFORE_START));
}
```

Imagen 18: Método "init" clase GenRSA.java

Como se puede observar en la imagen 17, la barra de progreso de notificaciones es simplemente un bucle "while". Realmente esta aplicación no necesita un preloader puesto que el tiempo de carga de la aplicación es insignificante. Sin embargo, el preloader se ha decidido poner puesto que es un elemento que hace que la aplicación se perciba más profesional.



## 8.2 Paquete GenRSA

### 8.2.1 Clase GenRSA.java

Esta clase es la que extiende a la clase "Application". Lo cual implica que es en esta clase donde se carga la escena genRSA.fxml (ventana principal) y se inicia toda la aplicación.

```

@Override
public void start(Stage primaryStage) throws Exception {
    try {
        Parent root = FXMLLoader.load(getClass().getResource("genRSA.fxml"));

        Scene scene = new Scene(root);

        primaryStage.setScene(scene);
        primaryStage.setTitle("genRSA - Generación de claves RSA");
        primaryStage.getIcons().add(new Image(GenRSA.class.getResourceAsStream("/allImages/genRSA.png")));
        primaryStage.show();
    } catch (IOException ex) {
        Logger.getLogger(GenRSA.class.getName()).log(Level.SEVERE, null, ex);
    }
}
  
```

Imagen 19: Clase GenRSA.java fragmento método start

Como dato particular, es en esta clase donde se le añade un icono para ser mostrado en la barra de tareas del sistema operativo donde se ejecuta la aplicación.



Imagen 20: Icono en barra de tareas

### 8.2.2 Escena genRSA.fxml

Esta escena se corresponde con la ventana principal de la aplicación. Al ser la primera que se desarrolló tuvo varias versiones. Con cada una de ellas se aprendió a manejar mejor las características y funcionalidades propias de la herramienta SceneBuilder.

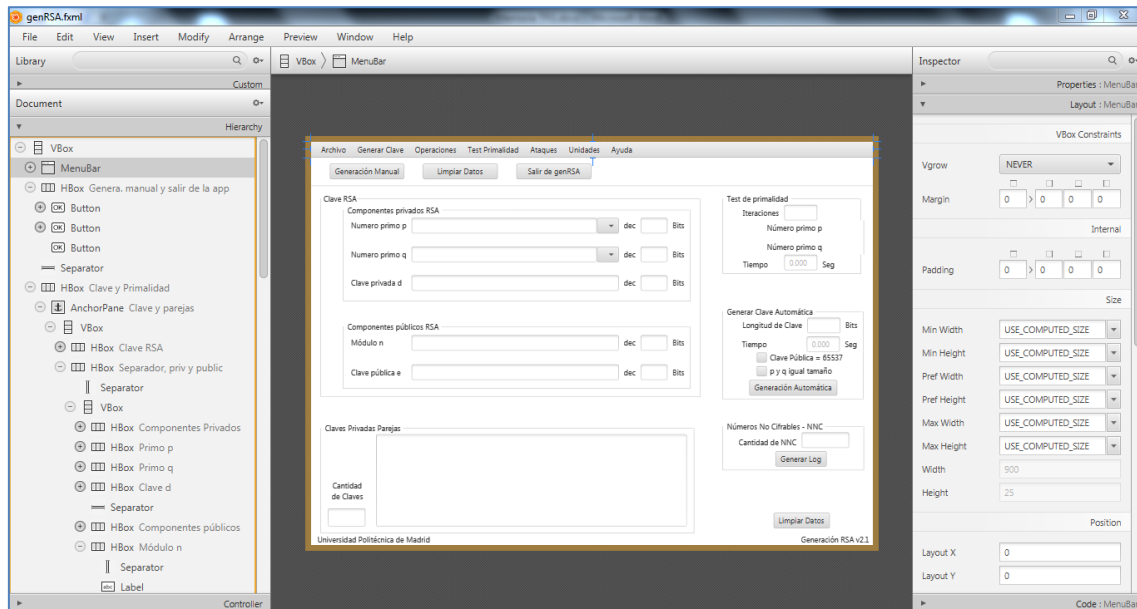


Imagen 21: Escena genRSA.fxml vista desde SceneBuilder

Con las últimas versiones se mejoró la escena creando pequeños módulos e integrándolos uno a uno en la escena principal. De tal manera que ante eventos que modificasen el tamaño de la escena, solo algunos de los módulos aumentasen o redujesen su tamaño proporcionalmente.

Estos módulos se crearon partiendo de los siguientes contenedores: "HBox", cajas donde se pueden incluir elementos horizontalmente; "VBox", cajas donde los elementos que se incluyen se posicionan verticalmente y "AnchorPane", los cuales permiten incluir elementos en cualquier posición del contenedor.

La ventaja de los "HBox" y "Vbox" frente a los "AnchorPane" es que se adaptan mucho mejor a los distintos tamaños que puede tomar la ventana facilitando el crecimiento de los elementos que contienen.

Como se aprecia en la imagen 22, en el diseño de genRSA solo se usaran "AnchorPane" cuando los contengan otros tipos de contenedores que permitan controlar su crecimiento.

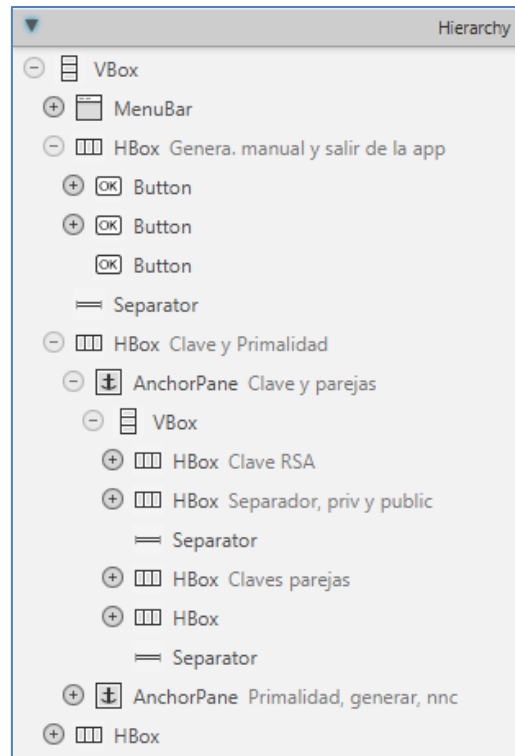


Imagen 22: Jerarquía de la escena genRSA.fxml

Con la herramienta SceneBuilder se añadieron atajos de teclado a esta escena para dar acceso rápido a todas las funcionalidades de la aplicación. Entre otros se añadieron los siguientes:

- Ctrl + S: Guardar clave.
- Ctrl + O: Abrir clave.
- Ctrl + Shift + X: Salir de la aplicación de forma segura.
- Ctrl + M: Ejecutar Test de Primalidad de Miller Rabin.
- Ctrl + F: Ejecutar Test de Primalidad de Fermat.
- Ctrl + 1: Ataque por la Paradoja del Cumpleaños.
- Ctrl + 2: Ataque Cíclico.
- Ctrl + 3: Ataque por Factorización.
- F1: Manual de Usuario
- F2: Banco de Pruebas
- F10: Acerca de

Como se puede observar en la lista superior, para la mayoría de atajos de teclado se tiene que pulsar la tecla Control. Sin embargo “Ayuda” y “Acerca de” se implementó simplemente pulsando las teclas de función F1 y F10 puesto que son las teclas predeterminadas en la mayoría de los programas.

Esto supuso tener que modificar el fichero FXML dado que la herramienta SceneBuilder no permite añadir atajos de teclado que solo se utilicen pulsando una tecla, como refleja la siguiente imagen.

```

</Menu>
<Menu mnemonicParsing="false" text="Ayuda">
  <items>
    <MenuItem mnemonicParsing="false" onAction="#help" text="Manual de Usuario">
      <accelerator>
        <KeyCodeCombination alt="UP" code="F1" control="UP" meta="UP" shift="UP" shortcut="UP" />
      </accelerator></MenuItem>
    <MenuItem mnemonicParsing="false" onAction="#testBench" text="Banco de pruebas">
      <accelerator>
        <KeyCodeCombination alt="UP" code="F2" control="UP" meta="UP" shift="UP" shortcut="UP" />
      </accelerator>
    </MenuItem>
    <MenuItem mnemonicParsing="false" onAction="#aboutGenRSA" text="Acerca de">
      <accelerator>
        <KeyCodeCombination alt="UP" code="F10" control="UP" meta="UP" shift="UP" shortcut="UP" />
      </accelerator></MenuItem>
  </items>
</Menu>
  
```

Imagen 23: Fragmento modificado escena genRSA.fxml

## Clase GenRSAController.java

Esta clase controla todos los eventos que ocurren en la ventana principal. Se puede decir que es la ventana cuya función principal es la interconexión de la parte gráfica con la parte lógica.

Es una clase compuesta de 1075 líneas de código. Estas se agrupan en métodos que serán ejecutados cuando se realice una acción de selección sobre elementos como botones, menús o cajas de selección.

En una clase de estas dimensiones son varios los puntos en los que se tiene que recurrir a la documentación de JavaFX o foros técnicos debido a la complejidad de la implementación de algunas funcionalidades. Los puntos más destacados son los siguientes:

- La asociación de un método a un botón de la ventana principal es una tarea tan sencilla como añadir dicho método al apartado "On Action" en la herramienta SceneBuilder.

Sin embargo, hacer que se ejecute un método cuando se pulsa una determinada tecla es una tarea mucho más complicada. Para lograrlo hay que crear otro método que maneje los eventos de teclado y filtre para activarse solo con la tecla elegida. Además este nuevo método deberá asociarse al apartado "On Key Pressed" en la herramienta SceneBuilder.

Un ejemplo de este caso se da cuando se pulsa la tecla “Enter” sobre las cajas primo p, primo q o clave pública para generar una clave de forma manual.

En este ejemplo, es el método “processManualGeneration2” asociado al apartado “On Key Pressed” el que generará una clave de forma manual cuando se pulsa la tecla “Enter”.

```

/**
 * Método usado cuando se pulsa enter al meter los primos p y q o la clave pública
 * @param keyEvent
 */
public void processManualGeneration2(KeyEvent keyEvent) {
    if (keyEvent.getCode() == KeyCode.ENTER) {
        this.processManualGeneration(new ActionEvent());
    }
}

```

Imagen 24: Método asociado a “On Key Pressed”

- Tampoco resultó sencillo hacer que se mostrase el número de bits a la vez que se introducía un número en las cajas anteriormente citadas.

En este caso la complejidad residía en que el número de bits se obtuviera “en tiempo real” y no con una pulsación de tecla de retardo. La solución pasó por asociar los métodos “bitsP”, “bitsQ” y “bitsPublicKey” al apartado “On Key Released” de cada una de las cajas de texto correspondientes.

```

/** Método usado cuando se introduce el primo P de forma manual, ...5 lines */
public void bitsP(KeyEvent keyEvent) {
    String primeP = this.primo_P.getText();

    this.generate.numberToBits(primeP, this.bits_primo_P);
}

/** Método usado cuando se introduce el primo Q de forma manual, ...5 lines */
public void bitsQ(KeyEvent keyEvent) {
    String primeQ = this.primo_Q.getText();

    this.generate.numberToBits(primeQ, this.bits_primo_Q);
}

/** Método usado cuando se introduce la clave publica de forma manual, ...5 lines */
public void bitsPublicKey(KeyEvent keyEvent) {
    String publicKey = this.clave_Publica.getText();

    this.generate.numberToBits(publicKey, this.bits_clave_Publica);
}

```

Imagen 25: Métodos asociados a “On Key Released”

- Otro de los aspectos que tuvo gran importancia en el desarrollo de todos los controladores fue la gestión de botones que tenían que estar habilitados durante el uso de la aplicación. Por ejemplo, antes de tener una clave generada no tiene sentido que se pueda pulsar el botón de generar log de los Números No Cifrables. Por este motivo, ese y otros botones deben estar deshabilitados cuando la clave no este generada. Sin embargo, cuando tengo la clave generada sí que cobra sentido pulsar dicho botón y por lo tanto debe estar habilitado.

El método "disableButtons" realiza esta gestión y puede ser ejecutado en combinación con el método "disableOnProgress", el cual deshabilita ciertos botones cuando se está generando una clave automática.

- Finalmente, la generación del log de Números No Cifrables supuso varios retos en su desarrollo.

El primero de ellos fue el crear una ventana para permitir al usuario elegir donde guardar el archivo de log en su sistema de ficheros. Para lo cual se hizo uso de la clase "FileChooser", la cual permitió elegir aspectos tan destacados como el título que tendría la ventana, el directorio inicial que se mostraría y la extensión en la que se guardaría el archivo (HTML). Además esta clase también permitía poner un nombre de archivo por defecto que se podría modificar.

El segundo de los retos fue hacer que durante la generación del log la interfaz gráfica respondiese en todo momento y no se quedase "colgada". En una primera versión, cuando se generaba el log para claves de más de 37 bits el programa dejaba de responder hasta que se terminaba de generar el archivo.

Como se puede ver en la imagen 26, en cierta medida se solucionó el problema creando un Thread de Java que ejecutase toda la lógica del método mientras que el Thread de JavaFX se encargaba de gestionar toda la interfaz gráfica.

El uso de otro Thread solventó el problema con la interfaz gráfica pero también acarreó dos nuevos desafíos.

El primero fue que dentro de un Thread de Java no se pueden modificar fiablemente elementos gráficos. Por este motivo, la ventana que debía mostrar donde guardar el archivo de log se situó en el Thread de JavaFX para que la mostrase desde un inicio. Y posterior a esta ventana, se lanzase el Thread de Java con toda lógica del método.

```

/**
 * Método usado cuando se pulsa el boton de generarLog de NNC
 * @param event
 */
public void generateNNC(ActionEvent event) {

    if (this.startLogNNC) {

        //todo se hace antes del thread porque si no nose podria manejar la ventana
        //para que se decidiera donde se guarda el archivo.
        FileChooser.ExtensionFilter extensionFilter = new FileChooser.ExtensionFilter("HTML files", "*.html");
        FileChooser fileChooser = new FileChooser();
        fileChooser.setInitialDirectory( new File(System.getProperty("user.home")));
        fileChooser.getExtensionFilters().add(extensionFilter);
        fileChooser.setTitle("Seleccionar directorio donde guardar el log");
        fileChooser.setInitialFileName("LogNNC genRSA");
        File logNNCFile = fileChooser.showSaveDialog(labelPubKey.getScene().getWindow());

        Task Cstart= new Task() {
            @Override
            protected Object call() throws Exception {
                startLogNNC = false;
                progress.setVisible(true);
                Platform.runLater(() ->{
                    disableOnProgress(true);
                    configureLogStop(true);
                });

                manageKey.saveLogNNC(unitsP, RSA, logNNCFile);

                Platform.runLater(() ->{
                    disableOnProgress(false);
                    configureLogStop(false);
                });
                progress.setVisible(false);
                startLogNNC = true;
                return null;
            }
        };

        new Thread(Cstart).start();

    } else {
        this.manageKey.setLogCancelled();
        this.startLogNNC = true;
    }
}

```

Imagen 26: Método generateNNC

El segundo desafío vino dado cuando se quería parar la generación del log. Debido al uso del Thread de Java, cuando se quería parar la generación se le debía avisar explícitamente.

Para ello, se modificó la ventana principal en tiempo de ejecución modificando el botón de "Generar Log" para que mostrase el texto "Parar Log". También se añadió una condición de parada a los bucles que calculaban los números no cifrables que se imprimirían en el log (ver imagen 27).

Esta solución se reutilizó para todas las funcionalidades que estaban implementadas con Threads (ataques y operaciones de cifra y firma).

```
/**
 * Método usado para cambiar el boton de pausa/generacion del log de NNC
 * @param stop
 */
public void configureLogStop(final boolean stop) {

    if (stop) {
        this.logNNCbtn.setDisable(false);
        this.logNNCbtn.setText(" Parar Log ");
    } else {
        this.logNNCbtn.setText("Generar Log");
    }
}
```

Imagen 27: Método cofigureLogStop



## **8.3 Package Methods**

Las clases de este paquete contienen toda la lógica necesaria para la ejecución de todas las funcionalidades de la ventana principal.

### **8.3.1 Clase CheckPrimes.java**

Es una clase que sigue el prototipo de modularidad empleado en todo el desarrollo de la aplicación:

- Constructor: método donde se inicializan todos aquellos objetos necesarios para la correcta ejecución de los métodos de la clase.
- Método de comprobación: en él se comprueban que los parámetros de entrada no contengan errores. En caso de contenerlos se mostrará una ventana de error. Si no contienen errores se llamará al método que realiza la acción específica.
- Métodos específicos: son los métodos que ejecutan la lógica necesaria para dotar de funcionalidades a la aplicación. Ellos se encargarán de realizar todos los cálculos y mostrar los resultados por pantalla.

Para esta aplicación se ha usado el mismo test de Fermat que ya se usaba en la versión 1.0. Pero este se ha programado de nuevo para optimizar su tiempo de ejecución.

También se ha implementado el test de Miller Rabin pero en combinación con el algoritmo de Lucas-Lehmer. Este último test lo provee la clase BigInteger del paquete "java.math".

### 8.3.2 Clase GenerateKeys.java

Es la clase encargada de la generación de las claves RSA. Existen dos formas de generación de claves: manual y automática.

Para la generación manual es necesario introducir los siguientes números: primo p, primo q y clave pública. Con ello el método "manualRSAkeys" comprobará si es posible generar una clave con esos valores y en caso afirmativo se obtendrá la clave.

Para la generación automática simplemente se introducirán el número de bits que se quiera que tenga la clave. Además gracias al método "autoRSAkeys" y a las cajas de selección "Clave Pública = 65.537" y "p y q de igual tamaño" que aparecen en la ventana principal, se podrán seleccionar diversos métodos de creación. Estos son:

- Generación automática con primos p y q de distinto tamaño.
- Generación automática con primos p y q de igual tamaño.
- Generación automática con primos p y q de distinto tamaño y clave pública igual a 65.537.
- Generación automática con primos p y q de igual tamaño y clave pública igual a 65.537.

En el caso de querer generar una clave con los primos p y q de distinto tamaño, la diferencia entre ambos variará a razón del tamaño de clave deseado. Para números de hasta 18 bits no habrá diferencia de bits. Para números de más de 18 bits pero menos de 25 la diferencia serán 2 bits. Para números de más de 25 bits pero menos de 30 la diferencia será de 4 bits. Para números de más de 30 bits pero menos de 40 bits la diferencia será de 6 bits. Y, finalmente, para números de más de 40 bits la diferencia de bits será igual a 8.

```

private int calculateDistanceBits(boolean isSameSize) {
    int keySize = this.RSA.getKeySize();
    int distance = 0;

    if (!isSameSize){
        if (keySize > 40){
            distance = 4;
        } else if (keySize > 30){
            distance = 3;
        } else if (keySize > 25){
            distance = 2;
        } else if (keySize > 18){
            distance = 1;
        }
    }

    return distance;
}
  
```

Imagen 28: Método para calcular la distancia entre el primo p y q

Por otro lado, en la generación automática la clave RSA se generará siguiendo 5 pasos.

- Paso 1: Seleccionar dos primos  $p$  y  $q$  aleatorios.
- Paso 2: Calcular el módulo  $n = p * q$ .
- Paso 3: Calcular phi de  $N$ .  $\Phi N = (p - 1) * (q - 1)$ .
- Paso 4: Encontrar la clave pública  $e$  tal que  $\text{mcd}(e, \Phi N) = 1$  y  $1 < e < \Phi N$ .
- Paso 5: Calcular la clave privada  $d$  tal que  $e * d = 1 \bmod \Phi N$ .

```

/**
 * Método para generar de manera automatica las claves RSA con una distancia de bits
 * entre p y q igual a distanceBits
 * @param keySize
 */
private void createRSAKeys(int distanceBits) {
    /* Step 1: Select the prime numbers (p and q) */
    this.RSA.setP( BigInteger.probablePrime((this.RSA.getKeySize()/2)+distanceBits, new SecureRandom()));
    this.RSA.setQ( BigInteger.probablePrime((this.RSA.getKeySize()/2)-distanceBits, new SecureRandom()));

    /* Step 2: n = p.q */
    this.RSA.setN( this.RSA.getP().multiply(this.RSA.getQ()));

    /* Step 3: phi N = (p - 1).(q - 1) */
    this.RSA.setpMinusOne( this.RSA.getP().subtract(Constants.ONE));
    this.RSA.setqMinusOne( this.RSA.getQ().subtract(Constants.ONE));
    this.RSA.setPhiN( this.RSA.getpMinusOne().multiply(this.RSA.getqMinusOne()));

    /* Step 4: Find e, gcd(e, φ(n)) = 1 ; 1 < e < φ(n) */
    do {
        this.RSA.setE( new BigInteger(this.RSA.getKeySize(), new SecureRandom()));
        // compareTo da 1 si es mayor que el valor entre parentesis
    } while ((this.RSA.getE().compareTo(this.RSA.getPhiN()) > -1) ||
        (this.RSA.getE().gcd(this.RSA.getPhiN()).compareTo(Constants.ONE) != 0);

    /* Step 5: Calculate d such that e.d = 1 (mod φ(n)) */
    this.RSA.setD( this.RSA.getE().modInverse(this.RSA.getPhiN()));
}

```

Imagen 28: Método para generar automáticamente la clave RSA

Una vez generadas la clave RSA, en esta clase también se calcularán las claves privadas parejas y la cantidad de números no cifrables asociadas a la clave recién generada. Para ello se usan los métodos "calculateNumNNC" y "calculateCKP".

### 8.3.3 Clase ManageKeys.java

Esta es la clase encargada de la gestión de las claves RSA. Permite guardar la clave RSA generada, abrir una clave que ha sido previamente guardada e incluso generar el log de números no cifrables .

Para permitir esta gestión se tiene que abrir una ventana en la que se permita seleccionar el fichero o directorio donde seleccionar o guardar la clave RSA. Como ya se ha comentado en el punto "GenRSAController.java", esta ventana se crea mediante la clase "FileChooser".

Con respecto a la clase "FileChooser", esta tiene un método que nos permite elegir que directorio se abrirá por defecto a la hora de pulsar el botón abrir o guardar clave, este método se llama "setInitialDirectory". En genRSA v2.1, en una primera versión se decidió poner la carpeta "user.home" como directorio por defecto.

Sin embargo, con el uso de la aplicación se comprobó que resultaba algo molesto tener que estar siempre navegando entre directorios cuando se va a guardar o abrir una clave. Para solucionar el inconveniente, siempre que se tenga abierto genRSA v2.1 solo se tendrá que elegir una vez el directorio donde se guardarán las claves, el resto de ocasiones la ventana por defecto será la última seleccionada.

Por otro lado, esta clase también genera el archivo de log de números no cifrables. En el archivo aparte de guardar dichos números también se guarda la clave RSA asociada. Por tanto, este log puede ser usado a la hora de abrir una clave desde un archivo.

El método "saveLogNNC" es el que gestiona la generación de este log. En él se elegirá entre dos implementaciones para generar el archivo. La diferencia entre ambas reside en la cantidad de números no cifrables precalculada en la clase "generateKeys.java".

Como se aprecia en la imagen 29, si la cantidad es menor que 10.000.000 entonces se usará el método "quickCalculate" de la clase "CalculateNNC.java", si la cantidad de números no cifrables es mayor se usará el método "calculate" de la misma clase.

```

/**
 * Método encargado de guardar el log de Número No Cifrables
 * @param label --> nodo cualquiera de la escena que se usa para ir escalando y obtener la ventana.
 * @param RSA
 * @param logNNCFile
 */
public void saveLogNNC (Label label, ComponentesRSA RSA, File logNNCFile) {
    CalculateNNC NNC;

    if (RSA != null){
        if (RSA.getD().bitLength() > Constantes.MAX_KeySize && logNNCFile != null) {
            Platform.runLater(() ->this.infoDialog.bigKeySize());
        }
        else {
            Platform.runLater(() ->this.errorDialog.RSAnotGenerated());
            return;
        }
    }

    if (logNNCFile != null){
        this.fileChooser.setInitialDirectory(logNNCFile.getParentFile());

        NNC = new CalculateNNC(this.radix, RSA, logNNCFile);

        if ((RSA.getNumNNC().compareTo(Constantes.MAX_NNC_BI) == -1)
            && !(RSA.getP().equals(RSA.getQ()))){
            NNC.quickCalculate();
        } else {
            NNC.calculate();
        }

        if (CalculateNNC.isCancelled){
            Platform.runLater(() ->this.infoDialog.LogNNCStopped());
        } else {
            Platform.runLater(() ->this.infoDialog.LogNNCSaved());
        }
    }
    else {
        Platform.runLater(() ->this.errorDialog.FileToSave());
    }
}

```

Imagen 29: Método saveLogNNC

### 8.3.4 Clase CalculateNNC.java

Antes de exponer los aspectos más destacados de programar esta clase, hay que definir que es un número no cifrable: un número no cifrable es aquel que al ser sometido a una operación de cifrado se obtiene el mismo número de partida. Por ejemplo, para la clave RSA con valores  $e = 29$ ,  $n = 49$  y  $d = 5$ , el número 48 es un número no cifrable, puesto que al realizar la cifra:  $48^{29} \bmod 49 = 48$ , se obtiene el mismo valor de partida 48, quedando el mensaje en claro.

Entendiendo NNC como Números No Cifrables, se puede asumir que esta clase se encarga de realizar los cálculos para obtener dichos números. Como se ha comentado en la clase "ManageKeys.java", en esta se diferenciará entre dos formas de calcular los números no cifrables en base a la cantidad que haya asociada a la clave generada.

Se usará el método "quickCalculate" para el cálculo de aquellas claves que solo tengan asociados menos de 10.000.000 de números no cifrables. Se ha elegido esta cantidad para que la aplicación no consuma demasiada memoria RAM y para que la lista donde se almacenan estos números sea fácil y rápidamente manejable.

Básicamente, "quickCalculate" recorre el espacio del primo  $p$  y el espacio del primo  $q$ . Y lo hace para calcular que números son no cifrables en cada uno de ellos y guardarlos en una lista para  $p$  y otra para  $q$ . Posteriormente combina ambas listas y obtiene los números no cifrables en el espacio del módulo de la clave.

El adjetivo "quick", rápido en español, se ha dado debido a que el otro método, llamado "calculate", que calcula los números no cifrables tiene un rendimiento más bajo. La diferencia entre ambos radica que en el método "quickCalculate" a pesar de que se recorren dos espacios (el de  $p$  y el de  $q$ ), estos son mucho menores que el espacio total del módulo que recorre el método "calculate".

Un ejemplo muy claro se da con la clave cuyos componentes son: primo  $p=6.047$ , primo  $q=428.863$  y clave pública  $=65.537$ . En esta clave la suma del espacio de " $p$ " y el espacio de " $q$ " da un valor de 434.910, que sería el número de cifrados a realizar para hallar los números no cifrables por el método "quickCalculate". Mientras que por el método "calculate" el espacio del módulo es 2.593.334.561. Esto implica que el método "calculate" tardará 5.296 veces más que el método "quickCalculate" para esta clave.

### 8.3.5 Clase Utilities.java

Esta es una clase que contiene pequeños métodos de utilidad que son empleados en gran cantidad de clases de la aplicación. Los métodos más destacados son los siguientes:

- FormatNumber: este método surge puesto que es muy común introducir números en la aplicación separados por comas, puntos, espacios o incluso saltos de tabulador. FormatNumber quitará estos elementos permitiendo el correcto funcionamiento de la aplicación y evitando que continuamente se muestre un mensaje de error indicando que se deben introducir solamente números en las cajas de texto.

```

/**
 * Metodo para quitar puntos, comas, espacios y tabuladores.
 * @param number
 * @return
 */
public String formatNumber (String number){

    number = number.replaceAll("[, . \\t]", "");

    return number;
}
  
```

Imagen 30: Método formatNumber

- IsNumber: este método se usará para comprobar que pequeños números se hayan introducido correctamente (sin caracteres extraños o letras). Estos pequeños números suelen ser el número de bits cuando la clave se genera de manera automática o el número de vueltas para cualquiera de los ataques o los tests de primalidad.

```

public boolean isNumber(String possibleNum) {
    boolean out = true;
    try{
        Integer.parseInt(possibleNum);
    } catch (NumberFormatException e){
        out = false;
    }

    return out;
}
  
```

Imagen 31: Método isNumber

- PutPoints: todo número mostrado en la aplicación pasa por este método. En él se pondrán los puntos correspondientes para agrupar de tres en tres los números decimales. En caso de que los números sean hexadecimales, el método no les realizará ninguna modificación. Este método tiene como finalidad facilitar a los usuarios la lectura de números muy grandes.

```
/**
 * Método que pone puntos al numero decimal pasado por parametro
 * @param number
 * @param radix
 * @return
 */
public String putPoints(String number, int radix){
    int length = number.length();
    if (radix == 10) {
        int processed= number.length()-3;

        while (processed>0){
            number = number.substring(0,processed ) +
                "." + number.substring(processed, length);
            processed = processed-3;
            length++;
        }
    }
    return number;
}
```

Imagen 32 : Método putPoints



## **8.4 Paquete Model**

Este paquete pretende almacenar todas aquellas clases que sirvan como modelado para la aplicación.

### **8.4.1 Clase Constantes.java**

Es una clase que todas sus propiedades son estáticas. En general las constantes almacenadas se pueden clasificar en dos grupos.

El primer grupo se corresponde con números de tipo BigInteger usados con mucha frecuencia en casi todas las clases de la aplicación. En segundo lugar, nos encontramos con constantes que indican el número máximo permitido para realizar una determinada acción. En otras palabras, gracias a estas constantes podremos discernir si elegir un método de ataque el cual consume mucha memoria o elegir otro menos agresivo con la memoria RAM. También nos permitirá elegir cada cuantas vueltas de un bucle mostramos los resultados por pantalla.

### **8.4.2 Clase ComponentesRSA.java**

Esta clase define el objeto el cual almacena todos los componentes de la clave RSA que ha sido generada.

Sus componentes básicos son los que más se utilizan: clave privada “d”, clave pública “e”, primo “p”, primo “q”, módulo “n” y tamaño de la clave.

Pero también almacena componentes más avanzados como son: cantidad de claves privadas parejas, cantidad de números no cifrables y el número Phi del módulo.

## 8.5 Paquete Cyclic

Este paquete contiene aquellas clases necesarias para crear la ventana del ataque cíclico, ejecutar la lógica del mismo y establecer la interconexión entre la parte lógica y la gráfica.

### 8.5.1 Escena Cyclic.fxml

Esta escena se va a comentar con gran nivel de detalle puesto que es una muestra representativa de todas las ventanas secundarias. De este modo, se entiende que todas las ventanas secundarias seguirán la misma jerarquía y combinarán los elementos sobre la ventana con un procedimiento muy similar.

A continuación se muestra una imagen de la escena creada a través de la herramienta SceneBuilder.

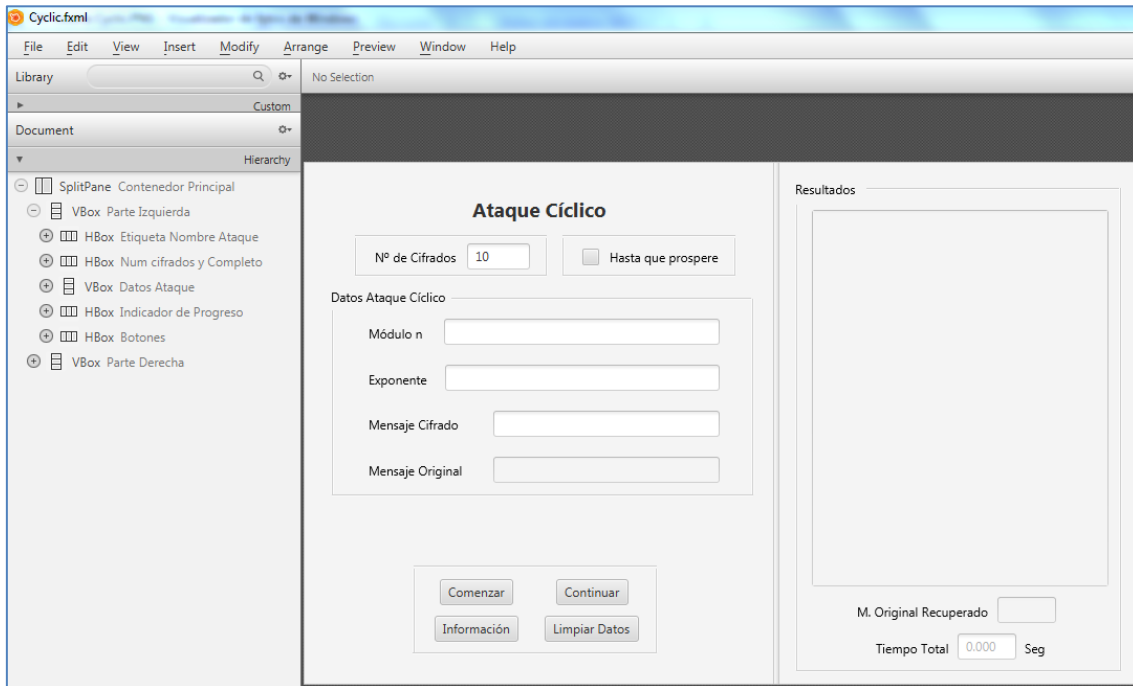


Imagen 33: Escena Cyclic.fxml

La herramienta SceneBuilder trabaja mediante contenedores y elementos gráficos. Como se muestra en el apartado Jerarquía (Hierarchy), situado en la parte izquierda de la imagen 33, el contenedor principal de la escena es un "SplitPane".

Un "SplitPane" es un contenedor que permite dividir la ventana en dos, pero además posibilita ampliar o reducir cada lado de la ventana para facilitar la lectura con claves de gran tamaño. Esta característica se puede apreciar comparando la imagen 34 la cual tiene ampliada la parte derecha dando prioridad a los resultados, con la imagen 7 la cual tiene el tamaño asignado por defecto.

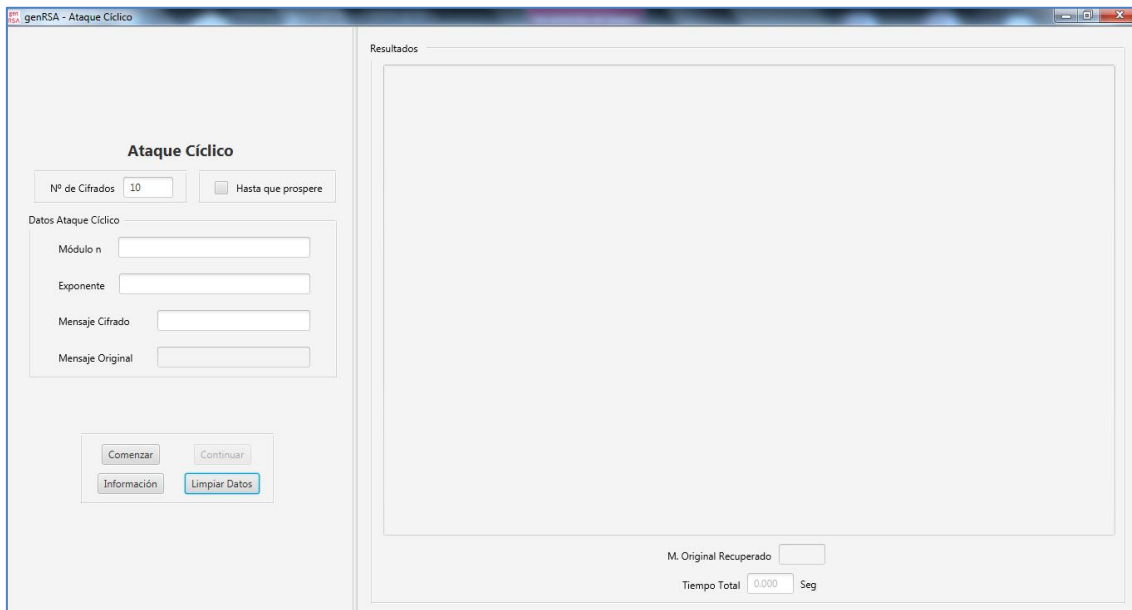


Imagen 34: Ventana Ataque Cíclico con prioridad en resultados.

Volviendo a la imagen 34, en ella se puede observar que el siguiente contenedor en jerarquía al "SplitPane" son las dos "VBox" (cajas verticales). Estas conforman la parte derecha y la parte izquierda de la ventana. Centrándonos en la parte izquierda se observa que está formada por más subcontenedores "VBox" y "HBox" (cajas horizontales).

Las "HBox" se pueden ver como bloques horizontales que contienen elementos gráficos tales como etiquetas, botones, cajas para introducir o mostrar texto y cajas de selección. Por otro lado, las "VBox", a pesar de que se pueden usar verticalmente del mismo modo que las "HBox", en el desarrollo de esta aplicación se han usado como subcontenedores que contendrán otros "HBox" o elementos de separación.

El motivo de usar tantos contenedores de tipo "HBox" y "VBox" es que estos permiten controlar de manera muy sencilla y eficaz eventos que modifiquen el tamaño de la pantalla.

Ambos contenedores tienen una propiedad que permite elegir en que ocasiones dicho elemento crecerá conforme a lo hace la ventana. La función de esta propiedad es permitir a los elementos que se ubiquen correctamente para distintos tamaños de ventana y que no se den situaciones en las que todos los elementos estén amontados en una esquina de la pantalla como ocurre con otros software.

Como se aprecia en el apartado de Diseño (Layout) de la imagen inferior, la propiedad que permite controlar el crecimiento de estos contenedores es "Vgrow" para el caso de las "VBox" y "Hgrow" para el caso de las "HBox". Estas propiedades tomaran valores como "Never" o "Always". Los cuales harán que, respectivamente, nunca o siempre crezca el contendor.

Además también existe la propiedad "Min Width" (Ancho Mínimo) la cual posibilita que los contenedores y elementos nunca se hagan más pequeños de un determinado tamaño. Esto es especialmente importante para asegurarnos que siempre estarán visibles ciertos datos del ataque. A diferencia de "Vgrow" y "Hgrow", esta propiedad si permite dar valores distintos a los valores por defecto. En la imagen inferior se puede observar que se ha dado el valor 400 a esta propiedad del contendor "Num cifrados y Completo".

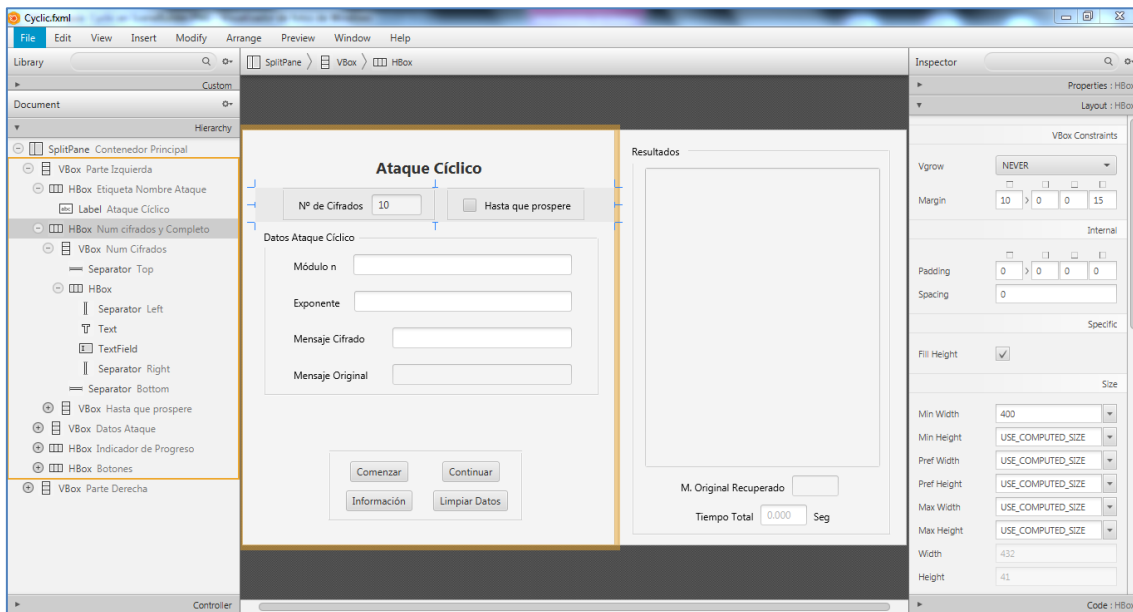


Imagen 35: Ampliación detalle estructura jerárquica y diseño.

Por otro lado, los elementos son componentes gráficos finales. Es decir, son los últimos nodos en la ramas de la jerarquía. Los elementos más empleados que componen las "HBox" de la aplicación son de seis tipos:

- Botones: los botones son elementos de control y representan la forma más habitual de interactuar con el usuario. Por ejemplo, en los ataques existen los botones "Comenzar", "Continuar", "Información" y "Limpiar Datos". Estos estarán habilitados o deshabilitados dependiendo del estado de la ejecución. Además se podrá editar el texto que contienen, particularmente el botón "Comenzar" se editará en plena ejecución para que se convierta en botón "Parar".
- Texto: permiten mostrar información acerca de la caja de texto que tienen a su derecha. También se usan para mostrar el nombre que se le da a un bloque o ventana.
- Cajas de Texto: sirven para introducir texto o números. Algunas de estas cajas permanecerán en todo momento ineditables a nivel de usuario, es decir, que no se puede introducir o modificar el texto que hay en ellas. Es el caso de las cajas asociadas a Mensaje Cifrado y M. Original Recuperado. No obstante, el resto de cajas podrán ser editables o no dependiendo del momento de ejecución.
- CheckBox: también llamadas cajas de selección, nos permiten marcar o desmarcar una opción. Es el caso de la opción "Hasta que prospere". A nivel de programación estas cajas suponen añadir una condición "if" en alguna parte del código.
- Áreas de Texto: tienen las mismas funcionalidades que las Cajas de Texto pero con la diferencia que en ellas se pueden escribir varias líneas y permiten hacer "scroll" (desplazarse hacia arriba y hacia abajo con el ratón). Son por tanto mucho más grandes verticalmente y se postulan como el elemento perfecto para mostrar los resultados del ataque.
- Separadores: son las líneas verticales y horizontales que agrupan los distintos conjuntos de elementos. Con ellas se facilita la lectura permitiendo la correcta distinción de funcionalidades dentro de una misma ventana.

### 8.5.2 Clase **CyclicController.java**

La clase CyclicController es una clase controlador que se encarga de interconectar la parte gráfica con la parte lógica. Todos las ventanas secundarias poseen una clase cuyo funcionamiento es similar a esta.

En esta clase se declaran todos los elementos de la ventana que se modifican durante la ejecución de la aplicación. A estos elementos se les da un identificador (fx:id) durante la creación de la escena y ese mismo identificador tiene que ser usado al declarar el elemento en esta clase.

Para evitar que se cometan errores, en el método "initialize" se comprueba que tanto en la escena como en esta clase los elementos tengan el mismo identificador. Este método también se usará para inicializar variables, crear objetos, deshabilitar el botón de continuar y poner el foco del teclado en la caja de texto de "Mensaje Original".

```

@FXML // This method is called by the FXMLLoader when initialization is complete
void initialize() {
    assert NumCiphers != null : "fx:id=\"NumCiphers\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert Complete != null : "fx:id=\"Complete\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert Message != null : "fx:id=\"Message\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert Modulus != null : "fx:id=\"Modulus\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert Exponent != null : "fx:id=\"Exponent\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert CypherMessage != null : "fx:id=\"CypherMessage\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert startBtn != null : "fx:id=\"startBtn\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert continueBtn != null : "fx:id=\"continueBtn\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert clearBtn != null : "fx:id=\"clearBtn\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert Results != null : "fx:id=\"Results\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert progress != null : "fx:id=\"progress\" was not injected: check your FXML file 'Factorizacion.fxml'.";
    assert mRecovered != null : "fx:id=\"mRecovered\" was not injected: check your FXML file 'Cyclic.fxml'.";
    assert Time != null : "fx:id=\"Time\" was not injected: check your FXML file 'Cyclic.fxml'.";

    firstTime = true;
    start = true;
    cyclicAttack = new CyclicAttack(new CyclicPrint(this));
    continueBtn.setDisable(true);

    Platform.runLater(Message::requestFocus);
}
  
```

Imagen 36: Método initialize

En esta clase, también existirán tantos métodos como botones haya en la ventana del ataque. En este caso tenemos cuatro botones y los métodos asociados son los siguientes:

- StartStop: este método se encarga de iniciar el ataque o de pararlo, por lo tanto se activa cuando se pulsa el botón de Comenzar/Parar. Para distinguir cuando se tienen que ejecutar las acciones de parar y cuando las de comenzar el ataque se usa una variable global que indica que botón de los dos está activo.

Este método permite iniciar el ataque de dos formas. La primera es indicando el número de vueltas que se quieren ejecutar. La segunda forma se da cuando se selecciona la opción "Hasta que prospere". Esta ejecuta el ataque hasta que encuentra la solución al mismo.

- Continue: en caso de haber iniciado el ataque indicando un número de vueltas y no se haya encontrado el resultado, el botón "Continuar" se habilitará permitiendo ejecutar este método.

Al igual que ocurre con el método "StartStop", este método cuando se ejecuta hace uso de la clase Thread de Java. En este Thread se ejecutará toda la parte lógica. Conforme se va ejecutando la lógica se van obteniendo resultados que se han de imprimir por pantalla.

Cuando se tiene que modificar la ventana el Thread de Java mandará ese trabajo al Thread de JavaFx mediante el método "runLater" de la clase Platform. Este método planifica el trabajo en una cola de eventos la cual se ejecutará en un tiempo cercano futuro pero indeterminado.

```

@FXML
/**
 * Metodo para continuar con el ataque por donde se quedo
 * cuando se pulsa el boton continue
 */
public void Continue(ActionEvent event) {

    Task CAcontinue = new Task() {
        @Override
        protected Object call() throws Exception {
            start = false;
            String numOfCyphers = NumCiphers.getText();
            Platform.runLater(() -> progress.setVisible(true));
            cyclicAttack.Continue(numOfCyphers);
            Platform.runLater(() -> progress.setVisible(false));
            start = true;
            cyclicAttack.setIsCancelled(false);
            return null;
        }
    };

    new Thread(CAcontinue).start();
}
  
```

Imagen 37: Método Continue

En la imagen 37 también podemos observar que, en el método Continue se edita la visibilidad de un indicador de progreso. Este indicador de progreso estará visible durante la ejecución del método.

Finalmente, el botón “Continuar” se deshabilitará si se encuentra la solución al ataque o se pulsa el botón de limpiar datos.

- Clear: método que elimina la información mostrada en pantalla relativa a los resultados y datos introducidos para ejecutar el ataque. Está asociado al botón “Limpiar Datos” de la ventana del ataque.

Entre otras acciones vuelve a poner como editable las cajas de texto necesarias para realizar el ataque, habilita el botón “Comenzar”, deshabilita el botón “continuar”, elimina toda la información del área de texto de resultados y pone el número de vueltas a su valor por defecto.

- Info: este método está asociado al botón “Información” y permite mostrar una ventana que muestra información acerca del algoritmo del ataque.

Vistos los métodos más importantes para la ejecución del ataque cíclico, caben destacar dos pequeños métodos más los cuales se encargan de modificar partes gráficas de la ventana. El primero “warningModify” muestra un mensaje de información cuando se modifica el módulo o el exponente del ataque por primera vez. El segundo método, “processStart”, llamará al método “start” una vez que se introduce el mensaje original y se pulsa la tecla Enter.



### 8.5.3 Clase **CyclicAttack.java**

Clase que contiene toda la lógica del ataque cíclico. En términos generales, dentro de genRSA v2.1 son estas clases las más complicadas de programa, sobre todo por la cantidad de formas distintas de ejecución que se pueden dar.

Como toda clase que contiene la lógica tendrá un método que, a parte del constructor, servirá para inicializar todos los valores necesarios para el ataque. También servirá para comprobar que los datos introducidos no contengan errores permitiendo que continúe el ataque o por el contrario mostrará una ventana con el error y su solución. Este método se puede ver en el anexo 4.

Una vez inicializado el ataque, la clase controlador decidirá si llamar al método que realiza el ataque completo (hasta que prospere) o por el contrario se realiza un inicio de ataque por unas vueltas determinadas.

En el caso que se haya optado por realizar un ataque hasta que prospere lo primero que se hará es elegir el método por el que se visualizarán los resultados parciales. Es decir, no todas las ejecuciones mostrarán la totalidad de los resultados parciales. Esto es así puesto que tras realizar varias ejecuciones del ataque cíclico para un rango entre 16 y 50 bits de clave se comprobó que podría resultar contraproducente mostrar los millones de resultados parciales que se generan para claves de más de 24 bits.

Un usuario que ejecute un ataque a una clave de más de 24 bits seguramente no esté interesado en poder seguir cada resultado parcial del ataque, más bien le interesará poder seguirlo de una forma sencilla viendo resultados parciales cada cierta cantidad de vueltas. Por ello, en el ataque hasta que prospere se darán las siguientes situaciones:

- Claves de menos de 24 bits: se mostrarán todos los resultados del ataque. Se usa el método "LMComplete".
- Claves de más de 24 y menos de 32 bits: se mostrarán 1 de cada 10.000 resultados parciales usando el método "BMComplete".
- Claves de más de 32 y menos de 40 bits: se mostrarán 1 de cada 100.000 resultados parciales usando el método "BMComplete".
- Claves de más de 40 bits: se mostrarán 1 de cada 1.000.000 de resultados parciales usando el método "BMComplete".

En el caso que se haya optado por realizar un ataque indicando el número de vueltas la complejidad aumenta. El procedimiento es el siguiente: se realizará una comprobación de que el número indicado de vueltas sea el correcto, después se guardarán los valores iniciales del ataque por si no se logrará finalizar en el número de vueltas indicado y posteriormente se decidirá qué tipo de visualización de resultados se lleva a cabo. En esta ocasión, se mostrarán todos los resultados si el número vueltas indicadas es menor que 10.000, en caso contrario, se mostrará el cifrado inicial y 1 resultado cada 10.000 vueltas. No obstante, siempre se mostrarán los dos últimos cifrados de las vueltas indicadas.

Cuando se realiza el ataque indicando el número de vueltas y no se encuentra la solución al ataque se puede continuar el ataque por donde se ha quedado. Al continuar el ataque se puede indicar un número de vueltas totalmente distinto. En este caso, ocurrirá igual que en la ocasión anterior, es decir, se deberá realizar una comprobación de que el número indicado de vueltas sea el correcto. Este número de vueltas se sumará al indicado en la fase inicial del ataque y con él decidiremos si mostramos todos los resultados del ataque o 1 cada 10.000 como en la ocasión anterior.

Para finalizar, el ataque debía ir mostrando los resultados parciales calculados en tiempo real. Pero existía un desfase de milisegundos entre el Thread de JavaFX y el Thread que realizaba los cálculos lógicos. Esto suponía que los resultados mostrados fueran erróneos (duplicación de algunos resultados y carencia en otros). La solución que se implantó fue agrupar miles de resultados para ser impresos por pantalla como si fueran uno solo.

## **8.6 Paquete Factorize**

Este paquete contiene todas las clases necesarias para crear la ventana del ataque por factorización, ejecutar la lógica del mismo y establecer la interconexión entre la parte lógica y la gráfica.

La escena Factorize.fxml y la clase FactorizeController se han implementado de una manera muy similar a la comentada en el paquete Cyclic.

La clase FactorizeAttack, como es evidente, se ha implementado con otra funcionalidad pero mantiene la misma estructura que en el ataque Cíclico.

### **8.6.1 Clase FactorizeAttack.java**

El ataque por factorización se basa en factorizar el módulo (componente público) para obtener los valores de los primos  $p$  y  $q$  (componentes privados), permitiéndonos obtener la clave privada realizando unos sencillos cálculos.

El cambio más significativo que ha sufrido la lógica de este ataque, con respecto a la versión 1.0, ha sido el cambio de algoritmo. El algoritmo elegido para factorizar cualquier número compuesto es el algoritmo Pollard rho.

La antigua versión era capaz de elegir entre 4 algoritmos a la hora de factorizar un número dependiendo del tamaño del mismo. Estos algoritmos son: método de ensayo de divisores pequeños, algoritmo de Pollard, método de curvas elípticas y método de propósito general.

A pesar de que esto puede resultar una gran ventaja, el algoritmo implementado en la versión 2.1 mejora los tiempos de los distintos algoritmos usados en la versión 1.0. Como podemos ver en la imagen 38, el algoritmo Pollard rho generalmente deberá realizar menos pasos frente a los algoritmos siguientes: método Pollard  $p-1$ , método de Fermat, método por factorización directa y método de Brent.

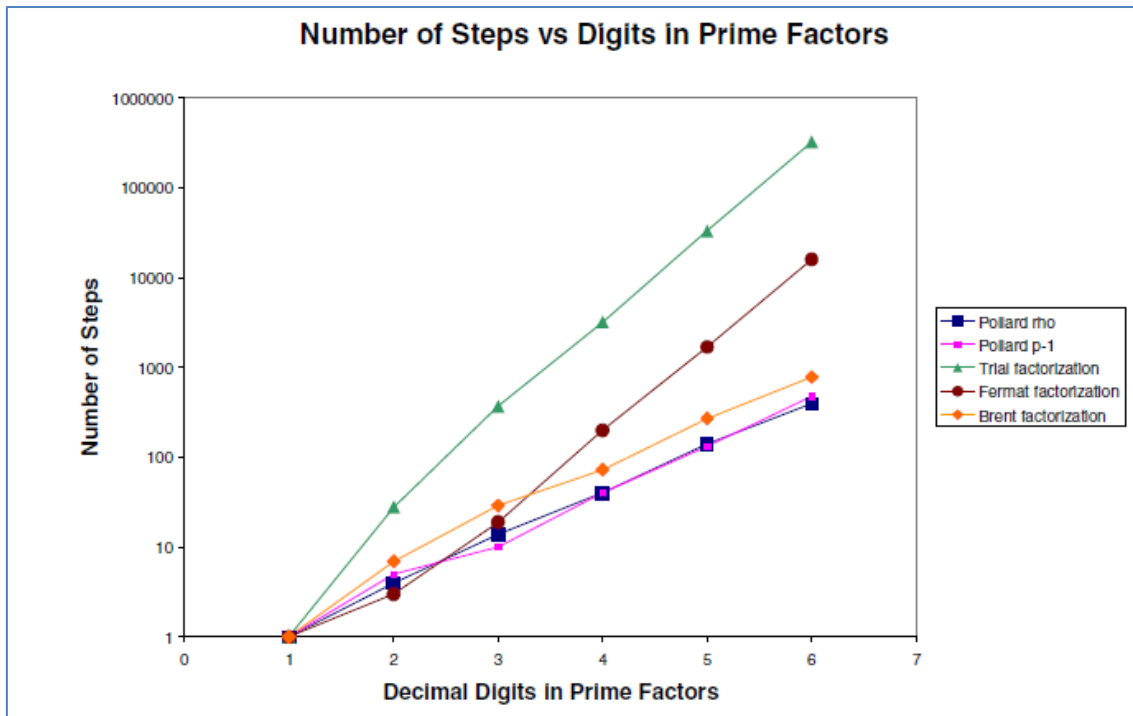


Imagen 38: Gráfica en el que se comparan distintos algoritmos de Factorización.  
 Fuente: Universidad de Oregón<sup>10</sup>

No obstante, es cierto que cuando se trata de factorizar un número que ha sido creado con primos muy cercanos el algoritmo Pollard rho se resiente en tiempo frente a los algoritmos usados en la antigua versión de GenRSA.

Por otro lado, como ocurre en el ataque cíclico no siempre es buena idea imprimir todos los resultados que se obtienen durante la realización del mismo. Por ello dependiendo del modo que se haya seleccionado se imprimirán más o menos resultados parciales:

- Iniciar ataque completo: cuando se selecciona la opción “Obtener p y q” se darán tres situaciones a la hora de obtener resultados.
  - Módulo menor que 50 bits: se imprimirán todos los resultados parciales del ataque.
  - Módulo mayor que 50 bits pero menor que 100 bits: se imprimirán 1 de cada 100.000 resultados parciales.
  - Módulo mayor que 100 bits: se imprimirán 1 de cada 1.000.000 de resultados parciales.

Estas tres situaciones están contempladas en el método “complete”.

```

/**
 * Método para decidir que tipo de visualización de los resultados se
 * lleva a cabo para el ataque completo.
 * No para hasta que prospera o se pulsa el boton de parar.
 */
public void complete () {

    if(this.modulus.bitLength() < 50){
        this.LMComplete();

    } else if(this.modulus.bitLength() < 100){
        this.BMComplete(Constants.BM_REFRESH);

    } else {
        this.BMComplete(Constants.MAX_REFRESH);
    }

}

```

Imagen 39: Método Complete

- Iniciar ataque indicando número de vueltas: en esta ocasión si el número de vueltas no supera las 3.000 se imprimirán todos los valores del ataque. En caso contrario se imprimirán 1 de cada 10.000 resultados parciales (siempre se imprimirán los dos últimos resultados parciales del número de vueltas indicado).

Si se compara esta funcionalidad con la misma que ofrece el ataque cíclico se observa que en el ataque cíclico se tienen que introducir más de 10.000 vueltas para que la aplicación no imprima todos los resultados. Para el ataque por factorización se han elegido 3.000 vueltas puesto que este ataque imprime tres líneas de resultados por cada vuelta en lugar de una sola línea como hace el ataque cíclico.

- Continuar ataque: al pulsar el botón de continuar, el ataque prosigue en el mismo punto donde se quedó. Y en este caso, ocurrirá igual que cuando se inicia el ataque indicando el número de vueltas. Con la única diferencia que ahora al número de vueltas indicado se le suman el número de vueltas realizadas anteriormente para comprobar si el número total es mayor que 3.000 o no.

## **8.7 Paquete Paradox**

Este paquete se corresponde a las clases del ataque por la paradoja del cumpleaños. Como en los paquetes de ataque anteriores se siguen la misma modalidad de programación.

### **8.7.1 Clase ParadoxAttack.java**

Este tipo de ataque es uno de los que más cálculos puede requerir para ejecutarse por completo. Por este motivo, se ha quitado la opción de ejecutar el ataque indicando cuantas vueltas se quieren realizar y se ha reforzado la opción de iniciar el ataque hasta que prospere añadiendo a la ventana una caja de texto en el que se indica la media de cifrados por segundo que se alcanzan en tiempo de ejecución.

A parte de añadir la media de cifrados por segundos y mejorar los tiempos de ejecución con respecto a la versión 1.0 de la aplicación, se ha añadido una comprobación del resultado con el cual podremos discernir si el resultado obtenido ha sido un falso positivo. Si no ha sido un falso positivo, se habrá encontrado la clave privada o alguna de las claves privadas parejas.

## **8.8 Paquetes DeCipher y Sign**

Estos paquetes se corresponden con las ventanas de las operaciones de Cifrado-Descifrado y Firma-Validación. Al ser ventanas secundarias se han configurado usando los mismos elementos gráficos que en el caso de los ataques pero con una distribución más compacta.

En lo referente al aspecto de programación se sigue la misma estructura que anteriormente, es decir, se tiene una clase controlador (para cada ventana) que conectará la parte gráfica con la lógica y una clase donde se definen las distintas funcionalidades lógicas de la cifra y de la firma.

Las clases que definen la lógica de la cifra y la firma son “DeCipherLogic” y “SignLogic” respectivamente. Ambas son prácticamente iguales, se diferencian en la clave usada en algunos métodos de la misma, por ello se comentarán los aspectos más destacados de solo una de ellas.

### **8.8.1 Clase DeCipherLogic.java**

Dado que la aplicación tiene una finalidad eminentemente didáctica, la implementación de esta clase supuso un gran reto. Sin duda, es una de las clases donde más tipos de errores se pueden dar, ya sea por parte del usuario al introducir los datos como por parte del programador al controlar los errores y convertir los datos.

Por parte del usuario el “error” principal que se va a cometer es tratar de usar la clase sin conocimiento técnico de cómo funciona el cifrado o el descifrado. En general, el usuario sabrá que tanto para cifrar como para descifrar se elevan los números a la clave pública o privada (respectivamente) y al resultado se le aplica el módulo, pero lo que no tendrán en cuenta será que no se deben cifrar números más grandes que el módulo y que en caso de querer cifrar texto el módulo de la clave ha de ser mayores que 11 bits.

Por parte del programador, deberá controlar multitud de errores para que en ningún momento la aplicación entre en error y pueda mostrar algún resultado incorrecto. Para ello tanto a la hora de descifrar como de cifrar, todos los datos introducidos por el usuario pasarán por un método que los procesará para que en caso de ser correctos o tener errores solucionables queden listos para la ejecución.

Existen dos métodos de procesado cuyo fin será controlar estos errores. Se ejecutarán en función de si los datos introducidos son texto o números:

- processNumbers: este método tiene como fin principal preparar los números introducidos para que puedan ser cifrados o descifrados correctamente. Para ello se realizan las siguientes comprobaciones en el orden que se encuentran:
  - Líneas en blanco: todas las líneas en blanco serán suprimidas.
  - Solo números: se comprueba que cada línea introducida solo contenga números. Los números pueden ser decimales o hexadecimales según se haya configurado en la ventana principal.
  - Números positivos: no se permiten números negativos ni tampoco el propio cero.
  - Números menores que el módulo: si el número es menor que el módulo directamente se inserta en una lista de números pendientes de cifrar o descifrar.

Si el número es mayor que el módulo se va partiendo dicho número en números lo más cercanos al módulo posible. Es decir, se parte el número introducido en números de igual cantidad de bits que el modulo o de un bit menos.

Para evitar confusiones una vez procesados todos los números introducidos se formatea el campo donde se han introducido y se muestran cada número procesado en una línea. De este modo se muestra claramente cómo se deberían haber introducido los datos y cuáles son los datos que realmente se van a cifrar o descifrar. Además se mostrará un mensaje indicando que se han modificado los valores introducidos.



- processText: como los datos introducidos también pueden contener texto, para el procesado se deben realizar más comprobaciones y una transformación de texto a código ASCII. El ASCII (American Standard Code for Information Interchange) es el Código Estándar Estadounidense para el Intercambio de Información.

Como el texto se codificará a números de 8 bits (ASCII) la primera comprobación será confirmar que el módulo cumpla con una longitud mínima de 12 bits. De este modo, con un módulo como mínimo 4 bits mayor, se evitará que la cantidad de números no cifrables sea tan alta que al cifrar el texto se obtenga el mensaje en claro.

Después se comprobará que no existan líneas en blanco en los datos introducidos. Y, a continuación, se codificará el texto a un número mediante la variante estandarizada "US" del código ASCII.

Una vez codificado se realizará la última comprobación. Esta es la misma que la definida anteriormente cuyo nombre es "Números menores que el módulo".

Si en alguno de los dos métodos hubiera algún fallo al realizar las comprobaciones, no se ejecutará ni el cifrado ni el descifrado. Y por pantalla aparecerá una ventana con un diálogo de error indicando el motivo del mismo y cuál es su solución.

## **8.9 Paquete Imprimir**

Todos los paquetes anteriores que hemos visto tienen clases dedicadas a la parte lógica y a la interconexión. Pero ninguno de ellos cuenta con clases especializadas que sean capaces de modificar la parte gráfica de la ventana para mostrar resultados o permitir nuevas funcionalidades. No obstante, es cierto que en los paquetes anteriores si realizan modificaciones menores a la ventana como son mostrar el indicador de progreso o hacer editable algunas cajas de texto.

Este paquete se encargará de realizar las modificaciones más importantes y también se encargará de crear ficheros HTML (HyperText Markup Language – lenguaje de marcas de hipertexto) en los cuales se guardaran las claves o el registro de log de los números no cifrables.

La complejidad real de este paquete se da a la hora de combinar los métodos que en él se implementan, debido a que en todo momento la aplicación debe guardar coherencia con los botones que se permiten pulsar en cada instante de la ejecución.

Para hacer más liviana esta complejidad el paquete está modularizado en clases. De tal modo que cada una de ellas corresponde a una funcionalidad, la relación entre clase y funcionalidad es la siguiente:

- GenRSAPrint: esta clase se encarga de imprimir los resultados de todas las funcionalidades existentes en la ventana principal: generación de claves, test de primalidad, cantidad de claves parejas y cantidad de números no cifrables.
- InitCbox: clase que se encarga de mostrar el desplegable de los números primos seguros.
- MainWindow: esta clase borra todos los resultados de la ventana principal y gestiona el cambio de unidades (decimal y hexadecimal) a nivel gráfico.
- CyclicPrint: clase encargada de la impresión de los resultados del ataque cíclico, así como de gestionar los botones y cajas de texto de la ventana durante la ejecución.
- ParadoxPrint: esta clase realiza la impresión de los resultados del ataque por paradoja del cumpleaños. También gestiona los botones y cajas de texto de la ventana del ataque.
- FactorizePrint: clase encargada de la impresión de los resultados del ataque por factorización, así como de gestionar los botones y cajas de texto de la ventana.

- DeCipherPrint: clase encargada de la impresión de los resultados al realizar las operaciones de cifrado y descifrado.
- SignPrint: clase que imprime los resultados en la ventana de firma y validación.
- SaveKey: clase que crea el archivo HTML donde se guarda la clave RSA generada.
- LogNNC: clase encargada de la generación del fichero HTML donde se guardan los Números No Cifrables correspondientes a la clave RSA generada. En el fichero también se guardan los componentes públicos y privados de la clave.

En este paquete también se incluyen las clases InfoDialog y ErrorDialog, las cuales su uso no está acotado a una funcionalidad específica. Se usarán en cualquier ventana de la aplicación cuando se pueda necesitar mostrar un mensaje de información o error.

### 8.9.1 Clase GenRSAPrint.java

Siguiendo la estética heredada de la antigua versión de la aplicación, los resultados de los test de Primalidad se muestran mediante imágenes. El poder introducir estas imágenes de manera programática ha sido la característica más destacada en el desarrollo de esta clase.

Para ello lo primero fue incluir el elemento "ImageView" (para la visualización de imágenes) en la escena "GenRSA.fxml". A continuación se realizó la carga de las imágenes en unas variables de la clase "GenRSAPrint", para ello se empleó el método "getResourceAsStream" contenido en la clase "Class<T>" del paquete "java.lang" (ver imagen 40).

```

/**
 * Constructor de la clase
 * @param sceneC
 */
public GenRSAPrint (GenRSAController sceneC){
    this.scene = sceneC;
    utilidades = new Utilities();
    cross = new Image(GenRSAPrint.class.getResourceAsStream("/allImages/cross.png"));
    tick = new Image(GenRSAPrint.class.getResourceAsStream("/allImages/tick.png"));
    interrogation = new Image(GenRSAPrint.class.getResourceAsStream("/allImages/interrogation.png"));
}
  
```

Imagen 40: Constructor de la clase GenRSAPrint.java

Una vez que ya se tenía cargada la imagen se crearon los métodos “primalityResults” y “flushIsPrime”. Estos métodos, dependiendo de los parámetros pasados, modifican la imagen mostrada permitiendo entender si el test de primalidad ratifica o no que el número es primo o si no se ha ejecutado ningún test de primalidad.

Pero surgió un problema a la hora de ejecutar la aplicación: las imágenes no se mostraban.

No es lo mismo ejecutar la aplicación dentro del entorno de desarrollo NetBeans que hacerlo usando el archivo JAR generado. Un archivo JAR (Java ARchive), como su nombre indica, es un tipo de archivo de JAVA el cual permite ejecutar la aplicación sin necesidad de usar un entorno de desarrollo.

Volviendo al problema, dentro de NetBeans la estructura de carpetas que contienen la aplicación es totalmente distinta a la estructura usada dentro del JAR. Debido a esto, la ruta de las imágenes usada para el método “getResourceAsStream” no era igual para ambos casos, por ello se decidió crear un paquete denominado “allImages” que contendría todas las imágenes del programa. De este modo, tanto en NetBeans como en el JAR la ruta sería la misma.

### 8.9.2 Clase InitCBox.java

Para facilitar las cosas al usuario de la aplicación se añadieron dos ComboBox. Estas permiten seleccionar primos seguros mediante un desplegable que se encuentra situado en la propia caja de los primos p y q. Lo cual facilita en gran medida la generación manual de una clave RSA a usuarios no experimentados.

Esta clase se implementó para facilitar la lectura del código debido a que las listas que contenían a los primos tenían un tamaño considerable. Estas listas contienen 474 primos seguros en formato decimal y hexadecimal.

Una característica que merece la pena destacar es el método empleado para asignar todos estos números a las ComboBox. En vez de asignar uno a uno mediante un bucle, como se puede observar en la imagen 41 se creó un objeto de tipo "ObservableList<String>" el cual contiene todos los números y puede ser asignado directamente a cada ComboBox.

```
/**
 * Método para inicializar las comboBox con primos seguros en formato decimal
 * @param primeP comboBox del primo P
 * @param primeQ comboBox del primo Q
 */
public void initCboxDec(ComboBox primeP, ComboBox primeQ){

    primeP.setDisable(true);
    primeQ.setDisable(true);

    ObservableList<String> obList = FXCollections.observableList(listDec);
    primeP.setItems(obList);
    primeQ.setItems(obList);

    primeP.setDisable(false);
    primeQ.setDisable(false);
}
```

Imagen 41: Método initCboxDec

### 8.9.3 Clase SaveKey.java

Esta clase genera un archivo HTML en el que se guardan todos los componentes de la clave RSA.

Para guardar los datos se empleó la forma más simple posible: escribir líneas en el fichero con las etiquetas correspondientes de HTML (ver imagen 42). Para la elección de estas etiquetas se consultó un editor online<sup>11</sup> de HTML que nos permitió mejorar el diseño del archivo. También se usaron distintos tipos de colores para resaltar la información importante<sup>12</sup>.

```

print.println("<center><h2><font color=\"navy\"> CLAVE GENERADA </font></h2></center>");
print.println("<center><h3><font color=\"grey\"> ( " + fechaStr + " ) </font></h3></center>");

if(radix==10){
    print.println("<B><font color=\"Black\">Unidades: Decimal</font></B>");
} else {
    print.println("<B><font color=\"Black\">Unidades: Hexadecimal</font></B>");
}

print.println("<B><font color=\"IndianRed\">Numero primo P generado:</font></B>");
print.println("<B>" + this.utilidades.putPoints(RSA.getP().toString(radix).toUpperCase(), radix) + "</B>");
print.println("<B><font color=\"IndianRed\">Numero primo Q generado:</font></B>");
print.println("<B>" + this.utilidades.putPoints(RSA.getQ().toString(radix).toUpperCase(), radix) + "</B>");
print.println("<b><font color=\"IndianRed\">Modulo N generado:</font></b>");
print.println("<B>" + this.utilidades.putPoints(RSA.getN().toString(radix).toUpperCase(), radix) + "</B>");
print.println("<B><font color=\"IndianRed\">Clave Publica e generada:</font></B>");
print.println("<B>" + this.utilidades.putPoints(RSA.getE().toString(radix).toUpperCase(), radix) + "</B>");
print.println("<B><font color=\"IndianRed\">Clave Privada d generada:</font></B>");
print.println("<B>" + this.utilidades.putPoints(RSA.getD().toString(radix).toUpperCase(), radix) + "</B>");

if (RSA.getN().bitLength()<26 && radix ==10){
    this.calculateAEE(RSA.getPhiN().intValue(), RSA.getE().intValue());
}
  
```

Imagen 42: Fragmento del método "generateHTML"

En la imagen superior también podemos observar que para claves decimales y de longitud menor que 26 bits el archivo HTML también mostrará paso a paso el Algoritmo Extendido de Euclides.

#### 8.9.4 Clase CyclicPrint.java

Todos los resultados y la mayor parte de los cambios gráficos del ataque cíclico se realizan gracias a esta clase. Como se ha comentado anteriormente, la mayor dificultad de gestionar los elementos gráficos es que toda la interfaz gráfica muestre un estado coherente con el instante de ejecución en el que se encuentra la aplicación. Un claro ejemplo de incoherencia sería que el botón de parar se mostrase en la interfaz gráfica cuando aún no se ha comenzado a ejecutar el ataque.

De nuevo, para tratar de facilitar la coherencia se han creado gran cantidad de métodos para que se pueda modularizar por funcionalidades y no tanto por elemento gráfico. Esto lo podemos ver reflejado en los métodos “enableStart” y “enableStop” de la imagen 43, los cuales permitirán gestionar varios elementos de la interfaz gráfica atendiendo a la funcionalidad de comenzar y parar el ataque cíclico.

```

/**
 * Método para permitir parar el ataque.
 * Activa el boton Parar
 */
public void enableStop() {
    this.scene.getStartBtnn().setText("Parar");
    this.scene.getStartBtnn().setDisable(false);
    this.scene.getClearBtnn().setDisable(true);
}

/**
 * Método para permitir comenzar el ataque.
 * Activa el botn Comenzar
 */
public void enableStart() {
    this.scene.getStartBtnn().setText("Comenzar");
    this.scene.getStartBtnn().setDisable(false);
    this.scene.getClearBtnn().setDisable(false);
    this.scene.getContinueBtnn().setDisable(true);
}

```

Imagen 43: Métodos “enableStop” y “enableStart”

### 8.9.5 Clases InfoDialog.java y ErrorDialog.java

Ambas clases son usadas para mostrar pequeñas ventanas de diálogo. Estas ventanas, llamadas diálogos, son posibles gracias a la clase Alert que pertenece del paquete "javafx.scene.control". Con ella se pueden crear pequeñas ventanas a las que añadir elementos como: un icono, una cabecera y un mensaje.

Para poder añadir el icono hay que acceder al escenario que genera la ventana de diálogo y es a dicho escenario al que se le añade el icono.

El icono para el diálogo de Información será distinto del de Error y ambos son añadidos dentro del constructor de la clase. La imagen 44 muestra el constructor de la clase ErrorDialog en el que se puede apreciar lo descrito anteriormente en este párrafo.

```
/**
 * Constructor de la clase
 */
public ErrorDialog() {
    this.alertError = new Alert(AlertType.ERROR);
    this.alertError.setTitle("Error");
    Stage stage = (Stage) this.alertError.getDialogPane().getScene().getWindow();
    stage.getIcons().add(new Image(ErrorDialog.class.getResourceAsStream("/allImages/error.png")));
}
```

Imagen 44: Constructor de la clase ErrorDialog



# Capítulo 9

## Diagrama de Clases



Una vez visto la estructura de paquetes y la mayor parte de las clases que componen genRSA v2.1 se puede comprender con mayor facilidad el diagrama de clases de este software.

En este diagrama se mostrarán las relaciones principales entre las clases más importantes. Es por ello que se han excluido aquellas que debido a su naturaleza se usen en muchas otras. Como ejemplo tenemos la clase "ComponentesRSA" que contiene toda la información de la clave RSA. Este precepto se ha aplicado para que el diagrama quede lo más limpio posible y sean las relación lógicas las que cobren más importancia.

La relación mostrada con flecha de línea continua y punta blanca indica que la clase origen contiene declarado un objeto del tipo de la clase destino. Por otro lado, la relación mostrada con flecha de línea discontinua y punta negra indica que la clase origen usa un objeto del tipo de la clase destino.

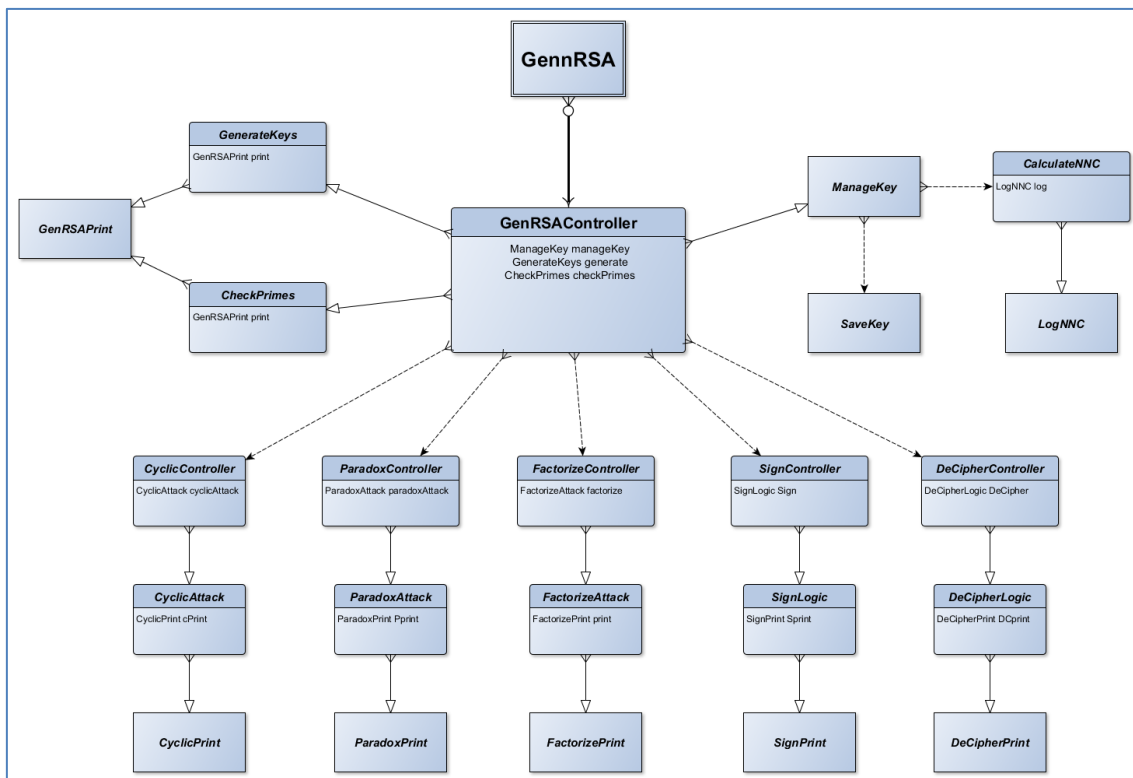


Imagen 45: Diagrama de Casos de Uso



# Capítulo 10

## Funcionalidades de genRSA v2.1

---



El desarrollo de la actualización de RSA supuso añadir y mejorar las funcionalidades que ya presentaba la versión anterior. A continuación se puede observar un diagrama de los Casos de Uso de la aplicación hecho mediante UML (Unified Modeling Language – Lenguaje Unificado de Modelado).

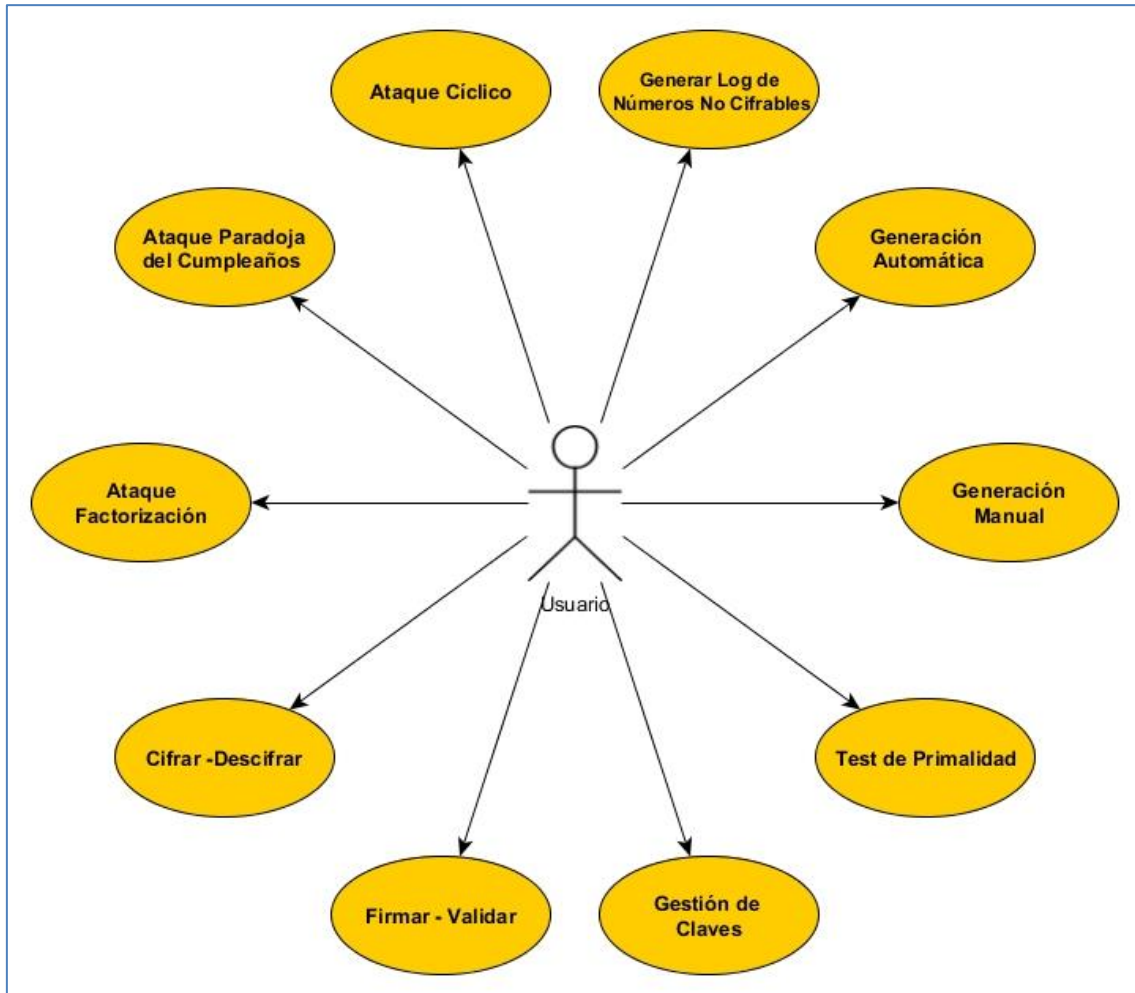


Imagen 46: Diagrama UML de Casos de Uso

En el diagrama se puede observar al usuario, el cual es el único actor de la aplicación. También se observan todos los casos de uso de genRSA. Todos ellos podrán ser ejecutados por el usuario.

## **10.1 Generación Manual**

La generación manual consiste en introducir ciertos componentes para obtener una clave RSA. Las pre-condiciones de este caso de uso son: introducir el primo  $p$ , el primo  $q$  y la clave pública.

A continuación se pulsará el botón de "Generación Manual" para obtener los resultados. Alternativamente, para ejecutar la generación manual se puede usar la tecla Enter en cualquiera de los parámetros anteriores después de haberlos introducido.

Como post-condición obtendremos una clave RSA. Esto se verá reflejado en pantalla al rellenarse los campos "Clave privada  $d$ ", "Módulo  $n$ ", "Claves privadas parejas" y "Cantidad de NNC". También desbloquearán los botones de "Cifrar\_Descifrar" y "Firmar\_Validar" situados en el menú de Operaciones junto con el botón "Generar Log" situado en la parte inferior derecha de la ventana principal.

## **10.2 Generación Automática**

La generación automática permite obtener una clave RSA sin necesidad de tener conocimientos acerca del algoritmo introduciendo la longitud en bits de la clave a generar. Las pre-condiciones de este caso de uso son: introducir el número de bits de la longitud de la clave y elegir las opciones de generación seleccionando o no las cajas de "Clave Pública  $e = 65.537$ " y " $p$  y  $q$  de igual tamaño".

Posteriormente se pulsará el botón de "Generación Automática" para obtener los resultados. Alternativamente, para ejecutar la generación automática se puede usar la tecla Enter al introducir la longitud.

Como post-condición obtendremos una clave RSA cuya longitud variará en función del número indicado. Al igual que en el caso de la generación manual, en la pantalla principal se rellenarán los campos "Clave privada  $d$ ", "Módulo  $n$ ", "Claves privadas parejas" y "Cantidad de NNC". También desbloquearán los botones de "Cifrar\_Descifrar" y "Firmar\_Validar" situados en el menú de Operaciones junto con el botón "Generar Log" situado en la parte inferior derecha de la ventana principal.



### **10.3 Test de Primalidad**

El test de Primalidad nos permite saber con una certeza determinada si los primos  $p$  y  $q$  son realmente primos.

Las pre-condiciones de este caso de uso son: introducir los números primos  $p$  y  $q$  (bien sea de forma manual o creando una clave automática) y el número de vueltas del test. A mayor número de vueltas mayor es la probabilidad de que el resultado del test sea correcto.

Posteriormente se pulsará el botón de “Miller Rabin” situado en el menú o se pulsaran las teclas “Ctrl + M” para obtener los resultados del test. Los resultados de este test tendrán una fiabilidad superior a  $1 - \left(\frac{1}{2}\right)^{\text{Número de Vueltas}}$ .

Alternativamente, se podrá ejecutar el test de “Fermat”. Para ello se pulsará el botón de “Fermat” situado en el menú o las teclas Ctrl + F. La complejidad de este test es de orden  $O(K \times (\log n)^{2+\varepsilon})$ .

Como post-condición, en ambos casos obtendremos el resultado del test de primalidad mediante imágenes. En la pantalla principal, en el apartado del test de primalidad se mostrarán las imágenes relativas al resultado. En caso de que el resultado del test haya sido la confirmación de la primalidad se mostrará un “check” en color verde, en caso contrario se mostrará un aspa en color rojo.

### **10.4 Gestión de claves**

Gracias a la Gestión de Claves se puede guardar una clave generada o abrir una clave previamente guardada.

La pre-condición para el caso de guardar una clave es que se tiene que haber generado una clave previamente. Para el caso de abrir una clave, la pre-condición es tener un archivo con una clave guardada.

Una vez vistas las pre-condiciones, se podrá pulsar el botón “Guardar Clave” situado en el menú o las teclas “Ctrl + S” para guardar una clave. Para el caso de querer abrir una clave, se pulsará el botón “Abrir Clave” o bien se pulsarán las teclas “Ctrl + O”.

En ambos caso se podrá navegar por el sistema de archivos para elegir la ruta donde guardar o abrir la clave.

Como post-condición en el caso de guardar una clave se obtendrá un mensaje en el que se indique que la clave se ha guardado correctamente. También se obtendrá un archivo HTML en la ruta seleccionada.

En el caso de abrir una clave obtendremos el mismo resultado que en la post-condición de Generación Automática.

## **10.5 Generar Log de Números No Cifrables**

Mediante la Generación del Log de Números No Cifrables se obtendrá en un fichero HTML todos aquellos números que se deberán evitar usar a la hora de compartir información con una clave determinada.

La pre-condición para este caso de uso es haber generado previamente una clave RSA.

Una vez que se tiene generada la clave, se podrá pulsar el botón “Generar Log” situado en la parte inferior derecha de la ventana principal. Esta acción abrirá una pantalla para seleccionar la ruta donde almacenar el fichero que se va a generar.

Como post-condición se obtendrá una ventana la cual informará que la generación del fichero ha finalizado. También se obtendrá un fichero HTML con los datos de la clave RSA y todos los números no cifrables asociados a la dicha clave.

## **10.6 Cifrar-Descifrar**

El usuario podrá cifrar y descifrar tanto texto como números a la par que podrá consultar la información acerca de estas operaciones.

La pre-condición para este caso de uso es haber generado previamente una clave RSA.

Una vez que tenemos generada la clave, se podrá pulsar el botón "Cifrar\_Descifrar" situado en la barra de menú o se podrán pulsar las teclas "Ctrl + Shift + C".

Como post-condición obtendremos una nueva ventana para realizar las operaciones de Cifrado y Descifrado. En ella se podrá introducir texto o números para el caso del cifrado y solo texto para el caso del descifrado. Para ambos casos se podrá consultar información acerca de estas operaciones.

## **10.7 Firmar-Validar**

Gracias a este caso de uso, el usuario podrá firmar y validar los datos firmados a la par que podrá consultar la información acerca de estas operaciones.

La pre-condición para este caso de uso es haber generado previamente una clave RSA.

Una vez que tenemos generada la clave, se podrá pulsar el botón "Firmar\_Validar" situado en la barra de menú o se podrán pulsar las teclas "Ctrl + Shift + F".

Como post-condición obtendremos una nueva ventana para realizar las operaciones de Firma y Validación. En ella se podrá introducir texto o números para el caso de la firma y solo texto para el caso de la validación. Para ambos casos se podrá consultar información acerca de estas operaciones.

## **10.8 Ataque Factorización**

El Ataque por Factorización permite al usuario factorizar el módulo de una clave RSA generada o un número cualquiera que se introduzca.

La pre-condición de este caso de uso es haber generado previamente una clave RSA o bien haber introducido un número a factorizar.

A continuación, se pulsará el botón "Factorizar n" situado en la barra de menú. Alternativamente se pueden pulsar las teclas "Ctrl + 3".

Como post-condición se obtendrá una nueva ventana para realizar el ataque por Factorización. En ella se podrá consultar información y ejecutar el ataque que factorizará el módulo de la clave o el número introducido.

## **10.9 Ataque Cíclico**

El Ataque Cíclico permite al usuario obtener el mensaje en claro de un mensaje cifrado partiendo de la clave pública, del módulo y del propio mensaje cifrado.

La pre-condición para este caso de uso es haber generado previamente una clave RSA o bien haber introducido manualmente los componentes públicos módulo y exponente (clave pública).

A continuación, se pulsará el botón "Cíclico" situado en la barra de menú. Alternativamente se pueden pulsar las teclas "Ctrl + 2".

Como post-condición se obtendrá una nueva ventana para realizar el ataque Cíclico. En esta ventana se podrá consultar información acerca del ataque y ejecutarlo.

## **10.10 Ataque Paradoja del Cumpleaños**

El Ataque por la Paradoja del Cumpleaños permite al usuario obtener la clave privada partiendo de un mensaje y los componentes públicos (exponente y módulo).

La pre-condición de este caso de uso es haber generado previamente una clave RSA o bien haber introducido manualmente un módulo y un exponente.

A continuación, se pulsará el botón “Paradoja del Cumpleaños” situado en la barra de menú. Alternativamente se pueden pulsar las teclas “Ctrl + 1” para ejecutar el ataque.

Como post-condición se obtendrá una nueva ventana para realizar el ataque por la Paradoja del Cumpleaños. En esta ventana se podrá ejecutar el ataque y consultar información acerca del mismo.



# Capítulo 11

## Mejoras efectuadas





La actualización de genRSA a la versión 2.1 ha supuesto mejoras en todas las facetas del programa. A continuación se van a detallar cada una de ellas.

### **11.1 Mejora de la parte gráfica**

A pesar de la restricción de mantener la pantalla principal muy similar a la de la versión anterior, se han podido añadir pequeñas modificaciones. La principal ha sido poder maximizar la ventana para adaptarse a cualquier tamaño de pantalla. También se han añadido dos desplegables para seleccionar primos seguros.

Sin duda, son las ventanas secundarias las que más mejoras han recibido. Todas ellas se han rediseñado de nuevo permitiendo ejecutar más acciones en ellas a la par que se ha mejorado la comprensión de la utilización de la misma.

### **11.2 Mejora en facilidad de uso y mejor aprendizaje**

La aplicación procesa todos los datos introducidos en busca de errores. Pero en esta versión, además de informarte del error te da una posible solución al mismo.

Además, con el rediseño de todas las ventanas secundarias la aplicación te permite observar que valores son necesarios para realizar ataques u operaciones de cifra o firma. En cada una de las ventanas existe un botón de información el cual te facilita la comprensión de la tarea que realiza dicha ventana.

### **11.3 Mejora de funcionalidad**

GenRSA v2.1 permite generar contraseñas de un número mayor que 8.192 bits, ya sea en Decimal o en Hexadecimal. Lo cual es un valor casi incomparable con los 2.048 bits máximos en Hexadecimal o con los 32 bits máximos en Decimal que permite la versión 1.0

Ahora la aplicación también permite ejecutar cualquier caso de uso para una clave sin importar el número de bits.

### **11.4 Mejora en estabilidad**

Como es bien sabido por los usuarios de la aplicación GenRSA v1.0, esta versión casi siempre que se ejecuta termina sin responder en algún momento, lo que causa que haya que forzar el cierre de la misma.

Con la actualización la estabilidad es una característica que está asegurada gracias a que no depende de librerías dinámicas. La versión 2.1 solo depende de que se tenga instalado la máquina virtual de Java, puesto que la librería principal utilizada es "java.math". Y al tratarse de una librería matemática los cambios en los métodos normalmente son debidos a cambios en el rendimiento de los mismos no a cambios en la forma de recibir los parámetros o en el número de ellos.

### **11.5 Mejora en velocidad**

La actualización de genRSA ha supuesto una enorme mejora en tiempos. Esto ha sido debido al cambio de algunos de los algoritmos y a la implementación que se ha realizado de los que eran iguales.

Para poder hacerse una idea de esta mejora con respecto a la versión 1.0 se van mostrar los resultados de unas pruebas comparativas. Cada una de las pruebas se ha ejecutado con la mayor longitud de clave soportada por la versión 1.0.

Además todas las pruebas se han ejecutado un mínimo de 10 veces y se han elegido unos valores de clave representativos.

- Generación de claves: para generar una clave Hexadecimal de 2.048 bits, la media obtenida muestra que el tiempo empleado es de 3.768

segundos para versión 1.0. Por otro lado, para la versión 2.1 la media es de tan solo 0.534 segundos.

- Ataque por la Paradoja del cumpleaños: para una clave de 20 bits, el tiempo empleado en la versión 1.0 ha sido de 20 segundos. Sin embargo, en la versión 2.1 el tiempo empleado ha sido de 0.082 segundos. Datos para reproducir el ataque: mensaje  $m=2$  y clave:  $p=1.597$ ,  $q=419$  y  $e=20.213$ .
- Ataque Cíclico: para una clave de 32 bits, el tiempo empleado por la versión antigua para obtener el resultado del ataque ha sido de 21 minutos y 50 segundos. Lo cual contrasta notablemente con los 0.726 segundos empleados por la versión 2.1. Datos para reproducir el ataque: mensaje  $m=2$  y clave:  $p=49.043$ ,  $q=45.707$  y  $e=46.313$ .
- Ataque por Factorización: para una clave de 32 bits, en la versión antigua se han alcanzado los 38 minutos y 46 segundos en la obtención del resultado. Sin embargo, para la versión actualizada el tiempo empleado ha sido de 0.001 segundos. Datos para reproducir el ataque:  $p=3.394.177$ ,  $q=739$  y  $e=29.239$ .

Vistos los resultados de la comparativa se puede concluir que el programa genRSA v1.0 ha sido superado notoriamente por su versión 2.1.

Por otro lado, también se ha comparado el ataque por cifrado cíclico de esta aplicación con el de la aplicación ringRSA. Para poder compararlos se ha elegido una clave de 42 bits y se han configurado ringRSA para que vaya mostrando resultados cada 1.000.000 de cifrados al igual que lo hace genRSA v2.1.

Con estas características de ataque, la aplicación ringRSA ha tardado 20 minutos y 7 segundos mientras que genRSA ha obtenido el resultado en 11 minutos y 46 segundos. Lo cual significa que ringRSA ha tardado un 70,96% más que la aplicación genRSA. Además la aplicación genRSA es más eficiente en el uso de la CPU usando tan solo un 26% mientras que ringRSA ha usado un 38%. Datos para reproducir el ataque: primo  $p=28.021.579$ , primo  $q=88.813$  y clave pública  $e=284.611.710.073$ .

Por tanto, se puede concluir que la implementación del ataque cíclico en genRSA v2.1 es bastante más eficiente en cómputo y en consumo de recursos que en la aplicación RingRSA.

El equipo usado para la realización de estas pruebas es un DELL Latitude E5470 con procesador Intel® Core™ i5-6300 con 4 CPUs a 2.40GHz y 8GB de memoria RAM.



# Capítulo 12

## Desarrollos futuros



GenRSA v2.1 es una aplicación muy completa, estable y con un gran componente didáctico. Su desarrollo se realizó durante cinco meses, pero dada la gran cantidad de funcionalidades este tiempo se hizo corto.

Por ello hay diversos apartados del desarrollo que se pueden mejorar e incluso hay nuevas funcionalidades que se pueden añadir. Acometer la mayoría de modificaciones no supondrá un esfuerzo excesivo gracias a la modularidad de las clases.

### **12.1 Mejora del ataque por la paradoja del cumpleaños**

Este ataque se puede mejorar tanto en velocidad como en funcionalidad. Para mejorar la velocidad de este ataque se puede utilizar más Threads que calculen en paralelo los cifrados del ataque, pudiendo multiplicar la velocidad actual por el número de procesadores de la maquina donde se ejecute.

La funcionalidad del ataque se podrá ampliar añadiendo un campo en el que se indique el número de vueltas del ataque para que este se pueda ejecutar poco a poco. Es decir, que tenga las mismas funcionalidades que el ataque cíclico y el ataque por factorización.

### **12.2 Comprobación exhaustiva de los datos introducidos**

La aplicación en su versión 2.1 ya comprueba los datos introducidos, pero se puede mejorar siendo más exhaustivo al comprobarlos.

Posibles comprobaciones futuras son: comprobar la longitud de los datos introducidos antes de preprocesarlos y comprobar la longitud de la clave antes de realizar ciertos ataques. Esta última permitirá mostrar una ventana en la que se informe que el ataque puede durar horas en realizarse, dotando al usuario del conocimiento necesario para poder parar el ataque.

### **12.3 Indicadores de progreso determinados**

Actualmente los indicadores de progreso, que se muestran al realizar acciones pesadas, son indicadores indeterminados. Es decir, en ellos no se muestra un porcentaje de ejecución realizada.

Para calcular el log de Números No Cifrables el indicador de progreso se puede hacer determinado de una manera bastante sencilla. En cambio, para todos los ataques o para la generación de claves se tendrían que hacer estimaciones para indicar el progreso de la ejecución.

A pesar de la posible complejidad, es una de las mejoras que aportarían más valor a la aplicación, permitiendo al usuario hacerse una idea aproximada del tiempo restante de ejecución.

### **12.4 Contenido y aumento de los tooltips**

Se han añadido “tooltips” por toda la ventana principal. El contenido de ellos se ha revisado varias veces, pero aun así habrá alguno cuyo contenido se pueda mejorar para hacerlo más entendible para usuarios no experimentados.

Además se podrán añadir tooltips a todas las ventanas secundarias para complementar la información mostrada por el botón “Información”.

### **12.5 Añadir hojas de estilo**

Las hojas de estilo CSS(Cascading Style Sheets) permiten modificar tanto en forma como en color los elementos de las ventanas. De este modo, aplicar una hoja de estilo a genRSA v2.1 dará como resultado una aplicación más vistosa.

Además, una vez creada una hoja de estilo se pueden modificar los colores de los elementos fácilmente para crear diversos estilos de interfaz.



## **12.6 Guardar estado de los ataques**

Una actualización de esta versión de genRSA podría contemplar la mejora de guardar el estado en el que se encuentra un ataque para poder retomarlo en otro momento. Esta es una característica que resulta muy interesante cuando los ataques se realizan a claves de tamaño tal, que el tiempo que conlleva la ejecución del mismo puede demorarse varias horas.

Además, el guardar el estado periódicamente mientras se realiza el ataque puede resultar muy interesante para aquellos casos en los que durante la ejecución haya un corte de luz o se acabe la batería del portátil.



# Capítulo 13

## Valoración Económica

---



Este capítulo pretende hacer una estimación del coste real de este proyecto.

En esta estimación se han tenido en valorado los siguientes aspectos: número de horas empleadas durante todo el proyecto, software utilizado y material necesario.

En cuanto al número de horas, durante todo el proyecto se han ido apuntando el tiempo empleado durante cada día para el desarrollo del mismo. Como se puede observar en el anexo 1 el número total de horas asciende a 356.

Asumiendo que este proyecto ha sido desarrollado por un becarios, se ha tomado como referencia el salario medio de un becario informático: 500 euros por 80 horas de trabajo. Por lo tanto el coste de desarrollo asciende a 2.225 euros.

En lo relativo al material empleado para desarrollar el proyecto, se ha necesitado un portátil y documentación. La documentación ha sido totalmente gratuita bien sea extraída de internet o de la biblioteca de la E.T.S.I.S.I.. Por el contrario, el portátil si representa un coste a añadir al proyecto. Concretamente, el equipo usado (DELL Latitude E5470) tiene un coste económico de 1.026 euros.

En cuanto al software empleado para el desarrollo, todos los programas han sido totalmente gratuitos. Así se deduce que este apartado no añade ningún coste al proyecto.

Finalmente podemos concluir que el coste final del proyecto se ha estimado en 3.251 euros.



# Capítulo 14

## Conclusiones

---





La realización de este Trabajo Fin de Grado me ha permitido aprender a desarrollar un verdadero proyecto de Ingeniería Informática. Cuando comencé el trabajo el único conocimiento que tenía acerca del mismo era un conocimiento teórico del algoritmo RSA. Pero conforme fueron pasando las semanas fui adquiriendo conocimientos teóricos, funcionales y técnicos que me permitieron desarrollar, no sin esfuerzo ni dedicación, la aplicación genRSA v2.1.

En lo referido a la implementación, hubo momentos en los que no se pudo avanzar nada a nivel de programación. Sobre todo durante las semanas que se estaban aprendiendo los conocimientos técnicos necesarios para desarrollar los objetivos. Esto resultó una pesada carga al ver que el tiempo pasaba y la aplicación no avanzaba en su desarrollo. Pero finalmente se comprendió que esta situación es normal en proyectos de este calibre donde el desarrollador tiene que aprender a usar diversas tecnologías.

Se puede afirmar que se cumplieron todos los objetivos acordados entre tutor y alumno. Siendo el más destacado la renovación de las ventanas secundarias para facilitar la comprensión de los ataques y de las operaciones de cifra y firma. Por este motivo, se espera que la actualización de la aplicación pueda facilitar aún más el aprendizaje del algoritmo RSA a todos los alumnos de la escuela así como a aquellos usuarios que la utilicen.

A nivel personal, lo que más valoro de este proyecto no son todos los conocimientos que he adquirido sobre interfaces gráficas, herramientas, tecnologías o criptografía. Lo que más valoro es que he aprendido a cómo gestionar mi tiempo, a confiar en los resultados y sobre todo que he aprendido a aprender.

Gracias a este trabajo sé que ante cualquier proyecto que se presente en mi vida tendré la posibilidad de enfrentarme a él con la certeza que podré abordarlo superando los obstáculos que se presenten durante el mismo.



# Bibliografía

---



- 1** Librería Jasypt - <http://www.jasypt.org/>
  - 2** Librería Legion of the Bouncy Castle - <http://www.bouncycastle.org/java.html>
  - 3** Arquitectura Criptográfica de Java - <http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html>
  - 4** API del paquete Security de Java - <http://docs.oracle.com/javase/7/docs/api/java/security/package-summary.html>
  - 5** API del paquete Crypto de Java - <http://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html>
  - 6** Herramienta para creación de interfaces gráficas - <https://www.qt.io/ide/>
  - 7** Documentación acerca de la concurrencia en JavaFX - <http://docs.oracle.com/javase/8/javafx/interoperability-tutorial/concurrency.htm#JFXIP546>
  - 8** Documentación y ejemplos prácticos de las ventanas de diálogo - <http://code.makery.ch/blog/javafx-dialogs-official/>
  - 9** Documentación acerca de JavaFX y SceneBuilder - <http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>
  - 10** Documento comparativo acerca de los algoritmos de factorización de enteros - <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.1230&rep=rep1&type=pdf>
- Es preciso recalcar que este documento fue obtenido de la siguiente página:  
<http://www.criptored.upm.es/crypt4you/temas/RSA/leccion8/leccion08.html>
- 11** Editor online de HTML - <https://html-online.com/editor/>
  - 12** Guía de códigos de colores para HTML - [https://www.w3schools.com/colors/colors\\_names.asp](https://www.w3schools.com/colors/colors_names.asp)



# Anexos

---





## Anexo 1

A continuación se muestran las tablas correspondientes a los meses de Febrero a Junio. Las horas mostradas en las tablas están redondeadas de 15 en 15 minutos.

<b><i>Febrero</i></b>							
	<b>Lunes</b>	<b>Martes</b>	<b>Miércoles</b>	<b>Jueves</b>	<b>Viernes</b>	<b>Sábado</b>	<b>Domingo</b>
<b>Día</b>	6	7	8	9	10	11	12
<b>Horas</b>	2h	1h 15m	1h	2h 15m	45m	-	2h 45m
<b>Día</b>	13	14	15	16	17	18	19
<b>Horas</b>	1h 30m	1h 15m	-	1h	2h 30m	3h	2h 30m
<b>Día</b>	20	21	22	23	24	25	26
<b>Horas</b>	1h	45m	2h 15m	-	1h	2h 30m	4h
<b>Día</b>	27	28					
<b>Horas</b>	2h 15 m	3h					

Total Horas Febrero: 37.

<b><u>Marzo</u></b>							
	<b>Lunes</b>	<b>Martes</b>	<b>Miércoles</b>	<b>Jueves</b>	<b>Viernes</b>	<b>Sábado</b>	<b>Domingo</b>
<b>Día</b>			1	2	3	4	5
<b>Horas</b>			-	1h	-	2h 15m	3h
<b>Día</b>	6	7	8	9	10	11	12
<b>Horas</b>	2h 30m	2h 15m	3h 30m	3h 45m		2h	4h
<b>Día</b>	13	14	15	16	17	18	19
<b>Horas</b>	1h 30m		2h 45m	3h 30m		3h 30m	2h
<b>Día</b>	20	21	22	23	24	25	26
<b>Horas</b>	4h	45m	3h 15m	-	1h	3h 15m	4h 30m
<b>Día</b>	27	28	29	30	31		
<b>Horas</b>	2h 45m	-	1h 30m	3h 45m	2h		

Total Horas Marzo: 64.

<b><u>Abril</u></b>							
	<b>Lunes</b>	<b>Martes</b>	<b>Miércoles</b>	<b>Jueves</b>	<b>Viernes</b>	<b>Sábado</b>	<b>Domingo</b>
<b>Día</b>						1	2
<b>Horas</b>						-	-
<b>Día</b>	3	4	5	6	7	8	9
<b>Horas</b>	E	X	A	ME	N	3h	3h 15m
<b>Día</b>	10	11	12	13	14	15	16
<b>Horas</b>	1h 30m	1h 15m	3h 30m	1h 15m	4h 30m	5h 30m	4h 15m
<b>Día</b>	17	18	19	20	21	22	23
<b>Horas</b>	4h 15m	3h 15m	3h 30m	2h	2h	5h 15m	8h 30m
<b>Día</b>	24	25	26	27	28	29	30
<b>Horas</b>	5h 30m		5h 30m	2h 45m	1h	6h 30m	6h 15m

Total Horas Abril: 84.

<b><u>Mayo</u></b>							
	<b>Lunes</b>	<b>Martes</b>	<b>Miércoles</b>	<b>Jueves</b>	<b>Viernes</b>	<b>Sábado</b>	<b>Domingo</b>
<b>Día</b>	1	2	3	4	5	6	7
<b>Horas</b>	8h 30m	8h 45m	1h	1h 30m	30m	8h 15m	7h 45m
<b>Día</b>	8	9	10	11	12	13	14
<b>Horas</b>	7h	T	R	A	B	A	JOS
<b>Día</b>	15	16	17	18	19	20	21
<b>Horas</b>	E	X	A	M	E	N	-
<b>Día</b>	22	23	24	25	26	27	28
<b>Horas</b>	E	X	A	M	E	N	-
<b>Día</b>	29	30	31				
<b>Horas</b>	1h	1h 30m	-				

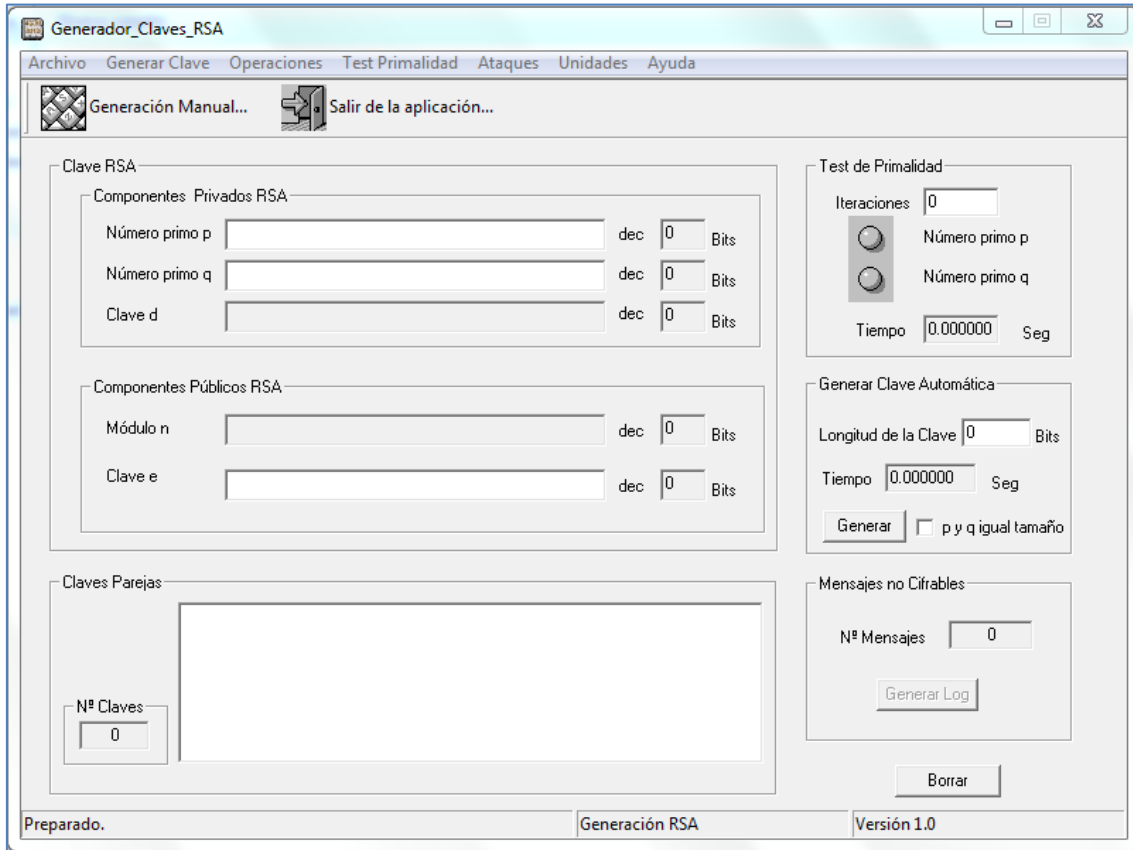
Total Horas Mayo: 45.

<b><u>Junio</u></b>							
	<b>Lunes</b>	<b>Martes</b>	<b>Miércoles</b>	<b>Jueves</b>	<b>Viernes</b>	<b>Sábado</b>	<b>Domingo</b>
<b>Día</b>				1	2	3	4
<b>Horas</b>				-	-	3h 45m	4h 15
<b>Día</b>	5	6	7	8	9	10	11
<b>Horas</b>	2h 15m	4h	4h 45m	4h 30m	2h 30m	4h 15m	4h 15m
<b>Día</b>	12	13	14	15	16	17	18
<b>Horas</b>	4h 15m	3h 15m	2h	3h 45m	3h	5h	5h 15m
<b>Día</b>	19	20	21	22	23	24	25
<b>Horas</b>	4h 45m	4h 30m	4h 45m	4h	1h 30m	7h 30m	8h
<b>Día</b>	26	27	28	29	30		
<b>Horas</b>	6h 30m	5h 30	5h	5h 15m	8h 45m		

Total Horas Junio: 126.

## Anexo 2

Imagen que muestra el aspecto de la ventana principal del programa genRSA v1.0.



## Anexo 3

Módulo obtenido con la aplicación genRSA v2.1 al generar una clave automáticamente de longitud 8.192 bits.

759.359.199.512.030.937.336.074.218.091.965.980.435.067.503.015.795.646.201.655.99  
9.921.502.612.590.886.768.181.341.710.061.678.495.276.605.896.338.465.249.233.282.0  
51.758.486.703.030.182.439.500.940.980.625.260.430.476.081.806.910.695.630.695.587.  
852.195.355.689.397.222.766.978.880.829.537.150.680.817.795.056.283.068.652.864.13  
7.027.647.333.809.044.272.788.766.413.827.769.079.055.144.237.202.768.720.855.875.3  
74.185.054.055.526.503.560.613.006.742.774.883.567.424.818.016.876.002.446.881.169.  
491.839.492.599.216.631.178.531.219.251.485.465.341.156.831.977.793.211.122.388.36  
9.264.400.330.103.391.040.448.309.236.121.812.328.677.879.568.774.397.866.992.721.4  
94.714.650.166.844.228.463.456.802.155.556.026.991.668.304.833.007.965.708.588.233.  
458.115.667.548.722.732.991.939.365.013.628.654.977.527.417.803.984.386.295.408.06  
1.894.517.629.990.858.654.782.192.350.011.708.149.832.556.645.847.751.540.658.272.4  
94.325.007.198.457.055.690.188.072.238.335.307.093.838.659.853.614.393.462.356.089.  
849.863.055.294.015.068.910.048.936.715.232.500.983.775.789.331.807.518.947.635.09  
6.429.101.450.991.766.975.493.567.682.669.889.028.984.335.310.651.082.077.458.157.7  
20.807.113.901.209.347.094.622.733.729.510.578.299.592.441.884.172.040.759.383.366.  
388.629.611.155.653.435.102.554.434.509.492.916.022.593.199.500.775.972.264.871.79  
4.120.576.612.830.000.005.650.826.830.520.719.117.486.544.994.682.717.198.522.646.6  
34.687.824.355.881.755.998.697.836.364.315.571.688.154.580.308.867.204.285.294.869.  
174.306.923.184.650.194.069.527.290.976.825.697.155.667.068.711.494.349.531.070.38  
4.128.667.840.365.318.067.062.953.092.979.601.290.905.775.841.951.251.309.210.879.7  
33.668.630.303.610.105.406.476.857.649.212.281.344.428.179.597.286.572.099.995.598.  
537.584.224.575.536.653.004.534.621.562.155.233.703.116.147.223.738.902.102.438.90  
1.457.749.580.098.804.730.839.926.038.190.898.785.335.158.732.188.217.508.027.657.4  
13.956.868.583.038.110.581.588.840.782.978.694.039.655.977.637.053.557.642.013.231.  
300.465.642.439.009.178.021.534.897.840.189.957.945.086.213.457.335.051.261.449.73  
4.603.833.019.854.129.632.951.712.028.440.564.955.289.971.186.854.078.863.836.094.7  
11.689.036.895.719.473.940.612.866.402.381.046.343.630.967.797.090.314.000.779.920.  
837.878.795.207.216.910.044.748.846.536.055.186.905.722.319.156.006.834.254.322.78  
8.871.651.967.884.441.828.947.220.961.915.087.375.799.815.101.123.818.591.047.042.1  
69.644.733.634.027.935.168.410.463.096.400.331.132.444.200.980.090.476.337.652.793.  
738.014.566.390.746.625.189.524.163.766.451.581.935.678.427.459.184.507.394.169.95  
9.343.232.172.626.564.271.833.353.478.373.073.488.605.343.709.231.200.793.751.319.8  
33.950.496.510.988.747.711.992.048.580.076.452.649.328.311.336.819.518.964.893.987.  
007.704.884.745.396.116.281.242.138.191.649.079.254.683.321.652.346.212.491.058.52  
6.417.556.529.807.909.909.761.574.014.416.632.678.681.473.259.645.065.614.133.852.7  
88.321.204.837.738.005.296.970.085.746.812.788.794.711.001.717.125.184.887.280.366.  
658.941.670.557.430.440.203.390.799.273.062.820.996.840.792.403.912.666.374.734.88  
6.076.797.153.905.122.365.552.103.180.116.676.243.042.333.253.747.678.412.153.754.2  
87.417.303.853.298.069.235.927.973.214.468.055.975.161.848.691.321.021.781.514.527.  
849.392.799.321.862.581.314.106.102.231.366.130.189.690.436.471

## Anexo 4

Método init de la clase CyclicAttack.java.

```
public boolean init(String message, String modulus, String exponent) {
    BigInteger messageBI;
    final String processedMessage;

    //CHECK MODULUS-----
    modulus = this.utilidades.formatNumber(modulus);

    try{
        this.modulus = new BigInteger(modulus, this.radix);
    } catch (NumberFormatException n){
        Platform.runLater(() -> errorDialog.Modulus(radix));
        return false;
    }

    //CHECK EXPONENT-----
    exponent = this.utilidades.formatNumber(exponent);

    try{
        this.exponent = new BigInteger(exponent, this.radix);
    } catch (NumberFormatException n){
        Platform.runLater(() -> errorDialog.Exponent(radix));
        return false;
    }

    if (this.exponent.compareTo(this.modulus)>-1){
        Platform.runLater(() -> errorDialog.bigExponent());
        return false;
    }

    //CHECK MESSAGE-----
    processedMessage = this.utilidades.formatNumber(message);

    try{
        messageBI = new BigInteger(processedMessage, this.radix);
    } catch (NumberFormatException n){
        Platform.runLater(() -> this.errorDialog.cyclicMessage(this.radix));
        return false;
    }

    if (messageBI.compareTo(Constantes.ONE) == -1){
        Platform.runLater(() -> this.errorDialog.cyclicMessage(radix));
        return false;
    }

    if (messageBI.compareTo(this.modulus) > -1){
        Platform.runLater(() -> errorDialog.bigMessage(radix));
        return false;
    }

    // INIT MESSAGE CIPHERED-----
    this.cypherMessage = messageBI.modPow(this.exponent, this.modulus);

    //PARTE GRAFICA-----
    Platform.runLater(() ->{
        this.Cprint.messages(this.cypherMessage.toString(this.radix).toUpperCase(),
            processedMessage.toUpperCase(),
            this.modulus.toString(this.radix).toUpperCase(),
            this.exponent.toString(this.radix).toUpperCase(),
            this.radix);
        this.Cprint.enableStop();
    });

    return true;
}
```

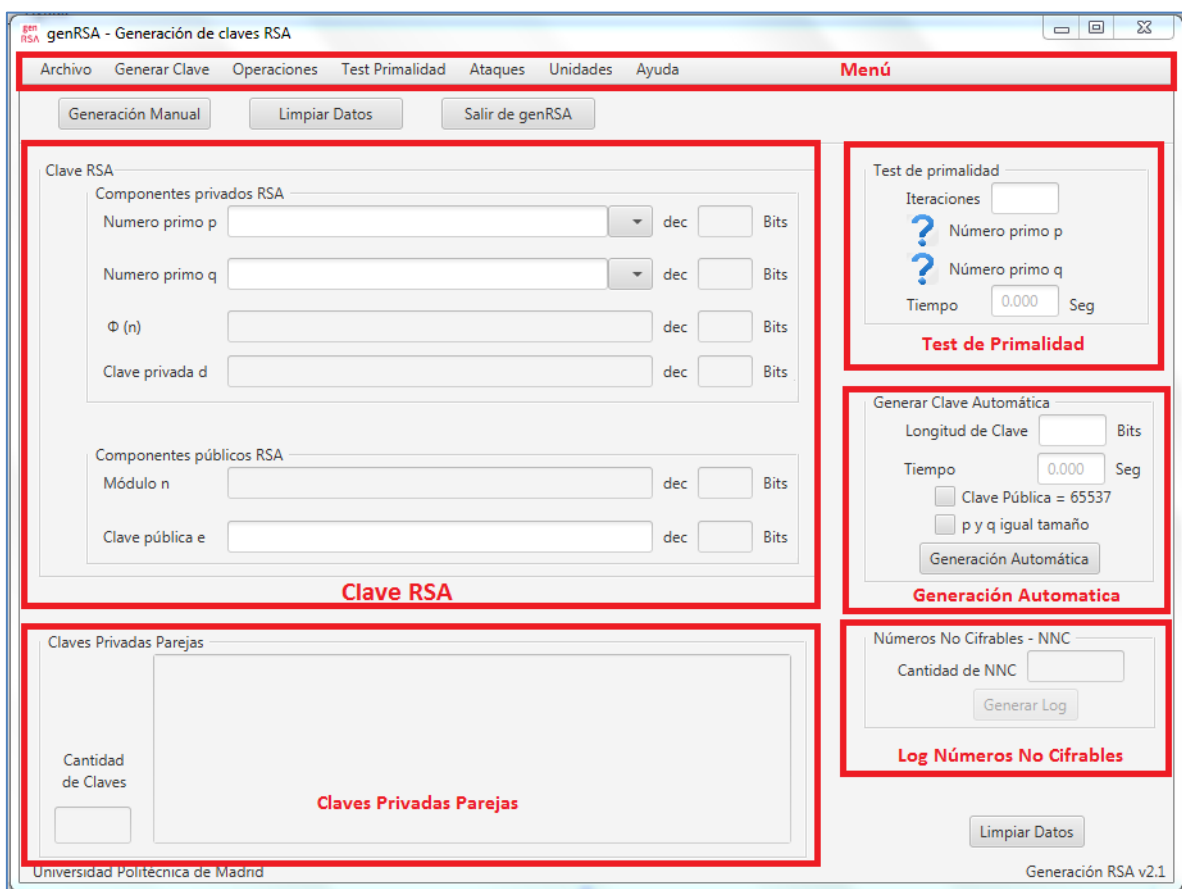
## Anexo 5

### Manual de Usuario genRSA v2.1

GenRSA v2.1 es una aplicación gráfica destinada a facilitar el aprendizaje del algoritmo RSA estudiado en el marco de la criptografía asimétrica.

### Funcionalidades Ventana Principal

Una vez ejecutado el archivo JAR aparecerá la ventana principal.



The screenshot displays the main window of the genRSA v2.1 application. The window has a title bar "genRSA - Generación de claves RSA" and a menu bar with options: Archivo, Generar Clave, Operaciones, Test Primalidad, Ataques, Unidades, Ayuda, and a "Menú" button. Below the menu bar are three buttons: "Generación Manual", "Limpiar Datos", and "Salir de genRSA".

The main interface is divided into several sections, each highlighted with a red border:

- Clave RSA:** This section contains two groups of input fields. The first group, "Componentes privados RSA", includes "Numero primo p", "Numero primo q", " $\Phi(n)$ ", and "Clave privada d". The second group, "Componentes públicos RSA", includes "Módulo n" and "Clave pública e". Each input field has a "dec" (decimal) button and a "Bits" (bits) button.
- Test de Primalidad:** This section includes "Iteraciones", "Número primo p", "Número primo q", and "Tiempo" (0.000 Seg). It features a red "Test de Primalidad" button.
- Generación Automática:** This section includes "Longitud de Clave", "Tiempo" (0.000 Seg), and checkboxes for "Clave Pública = 65537" and "p y q igual tamaño". It features a red "Generación Automática" button.
- Claves Privadas Parejas:** This section includes a "Cantidad de Claves" input field and a large text area. It features a red "Claves Privadas Parejas" button.
- Log Números No Cifrables:** This section includes "Números No Cifrables - NNC" and "Cantidad de NNC" input fields. It features a red "Log Números No Cifrables" button.

At the bottom of the window, there is a "Limpiar Datos" button and the text "Universidad Politécnica de Madrid" and "Generación RSA v2.1".

La ventana principal está dividida en seis secciones:

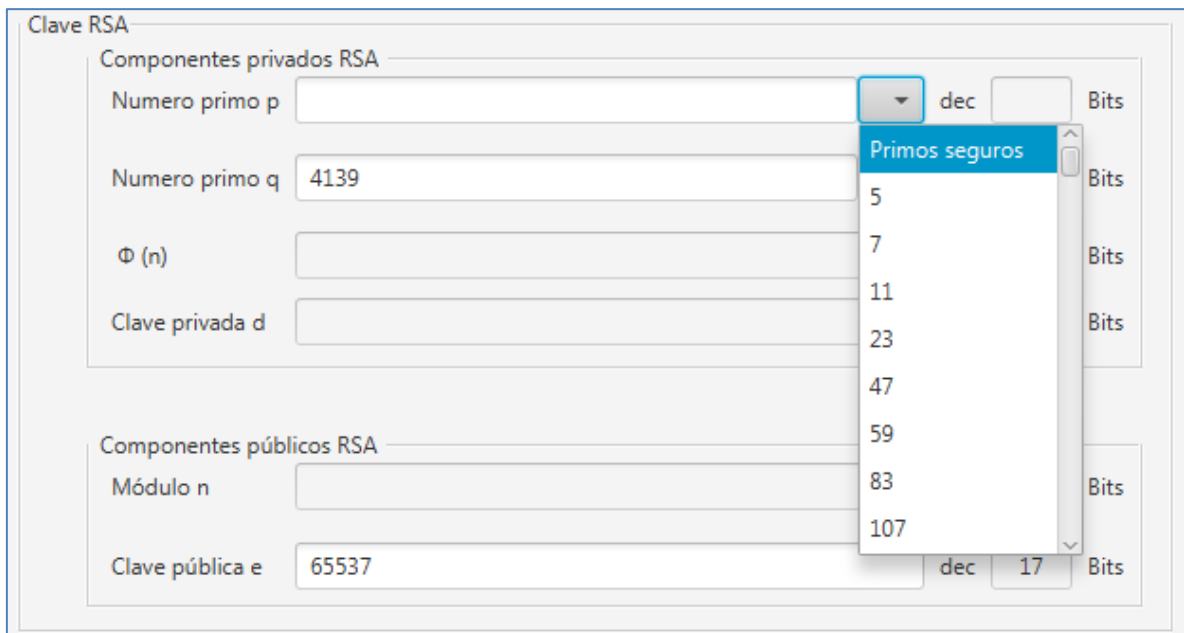
- Generación Automática: esta sección sirve para generar claves RSA de forma automática. Para ello solo se debe introducir la Longitud de la Clave que se quiere generar y pulsar el botón Generación Automática.
- Clave RSA: en ella se visualizarán los componentes de la clave generada, ya sea de forma manual o automática. En ella también se podrán generar claves de forma manual introduciendo los campos Número primo p, Número Primo q y Clave pública e y pulsando el botón "Generación Manual".
- Claves Privadas Parejas: una vez generada la clave se mostrarán la cantidad de claves privadas parejas asociadas a dicha clave. También se mostrarán hasta un máximo de 300 de estas claves. Cada una de ellas será mostrada informando de su longitud en bits.
- Log Números No Cifrables: en esta sección se mostrará la cantidad de Números No Cifrables(NNC) asociados a la clave. Además se podrá pulsar el botón "Generar Log" para crear un archivo HTML donde se visualicen los componentes de la clave y todos los NNC asociados.
- Test de Primalidad: muestra el resultado de la ejecución de los test de primalidad sobre el primo p y el primo q.
- Menú: la barra de menú permitirá ejecutar el resto de funcionalidades del programa genRSA. Muchas de estas funcionalidades abrirán pantallas secundarias que se comentarán más adelante.



Las funcionalidades que se pueden ejecutar sin necesidad de abandonar la ventana principal son las siguientes (todas ellas podrán ser ejecutadas para claves decimales y hexadecimales).

### Generación Manual

Para generar una clave RSA se han de rellenar los campos “Número primo p”, “Número primo q” y “Clave pública e”. En los campos de los números primos se pueden introducir los valores directamente o hacer uso de los desplegables en los que se pueden seleccionar números primos seguros.



Clave RSA

Componentes privados RSA

Número primo p  dec

Número primo q

$\Phi(n)$

Clave privada d

Componentes públicos RSA

Módulo n

Clave pública e  dec

Primos seguros

5

7

11

23

47

59

83

107

A continuación se pulsará el botón “Generación Manual” o bien se podrán pulsar la combinación de teclas “Ctrl+Shift+M”.

Como resultado de estas acciones se obtendrá una clave RSA, es decir, se rellenarán los campos “ $\Phi(n)$ ”, “Clave privada d”, “Módulo N”. Además, también se mostrarán las claves privadas parejas y la cantidad de NNC asociados a la clave recién generada.

No obstante, la generación de una clave, tanto de forma manual como automática, supondrá desbloquear las siguientes funcionalidades del programa:

- Guardar la clave recién generada en un archivo HTML.
- Realizar operaciones de Cifrado-Descifrado y de Firma-Validación.
- Generar el Log de Números No Cifrables.
- Realizar los tres tipos de ataque con los datos de la clave.

### Generación Automática

Para generar una clave de forma automática simplemente se introducirá la longitud que se quiere que tenga la clave (entre 6 y 8.192 bits). Y se pulsará el botón “Generación Automática” o la combinación de teclas “Ctrl+Shift+A”.

Alternativamente, se pueden realizar otros tipos de generación automática. Estos tipos son los resultantes de seleccionar o no las opciones “Clave Pública=65.537” y “p y q igual tamaño”.

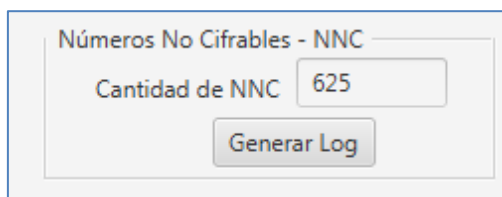
- La primera opción, generará una clave cuya clave pública será el valor más usado en certificados digitales(65.537). Para ello la longitud de clave mínima es 19. Si no se selecciona esta opción, la clave pública tomará el valor más pequeño posible.
- La segunda opción generará una clave cuyos primos p y q sean de la misma longitud de bits. Si no se selecciona esta opción la diferencia máxima de bits se dará en claves mayores a 40 bits donde la diferencia serán 8 bits.

No obstante, las claves inferiores a 19 bits y de longitud par tendrán los primos p y q de igual cantidad de bits.

Finalmente, una vez ejecutado cualquiera de las diferentes combinaciones de generación automática se obtendrá como resultado una clave RSA cuya longitud en bits del módulo n será igual al número introducido. De este modo, al igual que en la generación manual se rellenarán las secciones “Clave RSA”, “Claves Privadas Parejas” y “Log Números No Cifrables”. También se desbloquearán las mismas funcionalidades que en la generación manual.

### Generar Log Números No Cifrables

Una vez generada una clave se obtendrá la cantidad de NNC y se habilitará el botón “Generar Log”. Este botón se tendrá que pulsar para generar estos números no cifrables.



Una vez pulsado el botón se abrirá una ventana donde se podrá seleccionar donde guardar el fichero de log y el nombre que tendrá.

El log de NNC se generará en formato HTML. En él se guardarán los componentes de la clave y una lista con todos los NNC asociados a dicha clave. Dependiendo del tamaño de la clave el fichero puede tardar más o menos en generarse (a partir de los 50 bits de longitud de clave la generación puede alargarse bastantes minutos).

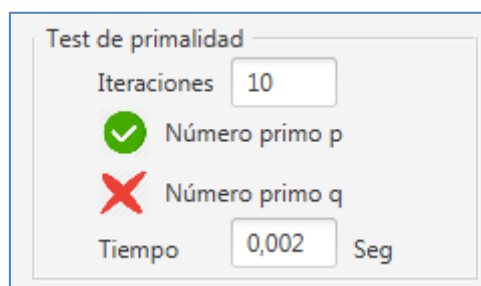
### Test de Primalidad

En esta aplicación se pueden realizar dos tests de primalidad, el de Fermat y el de Miller Rabin y Lucas-Lehmer. Para ejecutar cualquiera de ellos se han de introducir con anterioridad dos números a comprobar su primalidad en las cajas de "Número Primo p" y "Número primo q". Además se indicará el número de iteraciones a ejecutar el test, valor que oscila entre 1 y 300. A mayor número de vueltas mayor certeza de que el resultado sea correcto.

Para realizar el test de Fermat se pulsará el botón "Test de Primalidad > Fermat" situado en la barra de menú. De forma alternativa, una vez introducidos los datos se puede pulsar la combinación de teclas "Ctrl+F".

Para ejecutar el test de Miller Rabin y Lucas-Lehmer se pulsará el botón "Test de Primalidad>Miller Rabin" situado en la barra de menú. De forma alternativa, una vez introducidos los datos se puede pulsar la combinación de teclas "Ctrl+M".

En ambos casos se obtendrá el resultado de la ejecución y el tiempo empleado para la misma. Como se muestra en la imagen inferior, el resultado se mostrará mediante imágenes.



### Guardar una clave generada

Para guardar una clave generada simplemente se pulsará el botón “Archivo > Guardar Clave” situado en la barra de menú. Esto abrirá una ventana con un explorador de archivos donde se tendrá que indicar donde se guardará la clave.

### Abrir una clave generada

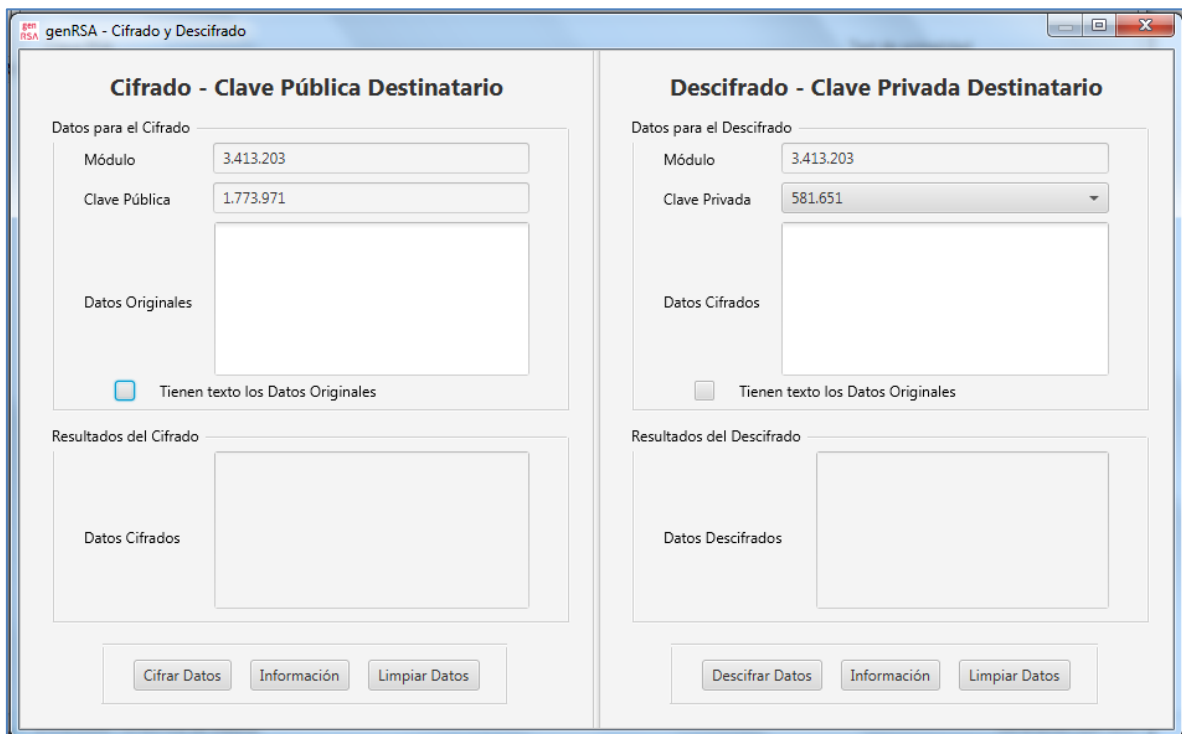
Para abrir una clave guardada simplemente se pulsará el botón “Archivo > Abrir Clave” situado en la barra de menú. Esto abrirá una ventana con un explorador de archivos donde se tendrá que indicar donde se encuentra el archivo guardado con la clave. Una vez abierto el resultado será el mismo que al generar una clave de forma manual o automática.

## Funcionalidades Secundarias

Las funcionalidades secundarias son cinco. Todas ellas se acceden desde la barra de menú.

### Operación Cifrar – Descifrar

Para poder acceder a esta funcionalidad previamente se debe haber generado una clave. Una vez generada se pulsará el botón “Operaciones < Cifrar\_Descifrar” situado en la barra de menú. Entonces se abrirá la siguiente ventana.



Esta ventana está dividida en dos: la parte de cifrado y la parte de descifrado.

- **Cifrado:** en la caja “Datos Originales” se pueden introducir números decimales o hexadecimales, según sea la clave generada. Estos números deben tener un valor positivo y menor que el módulo. En el caso de introducir un número mayor que el módulo, el número introducido se va partiendo en números lo más cercanos al módulo posible. Es decir, se parte el en números de igual cantidad de bits que el modulo o de un bit menos. Un ejemplo: si el módulo es 65.537 y se introduce el número 5883364551 para cifrar, este se partirá en los números 58833 y 64551.

Para evitar confusiones una vez procesados todos los números introducidos se formatea el campo donde se han introducido y se muestran cada número procesado en una línea. De este modo se muestra claramente cómo se deberían haber introducido los datos y cuáles son los datos que realmente se van a cifrar descifrar. Además se mostrará un mensaje indicando que se han modificado los valores introducidos.

En el caso de introducir texto a cifrar se debe marcar la opción "Tienen texto los Datos Originales". El texto se codificará a números usando el código ASCII, en el cual cada carácter se convierte en números de 8 bits. En este caso será necesario que la clave generada tenga una longitud mínima de 12 bits. Pero al igual que en el caso anterior, si al codificar los datos se obtienen números mayores que el módulo estos se partirán.

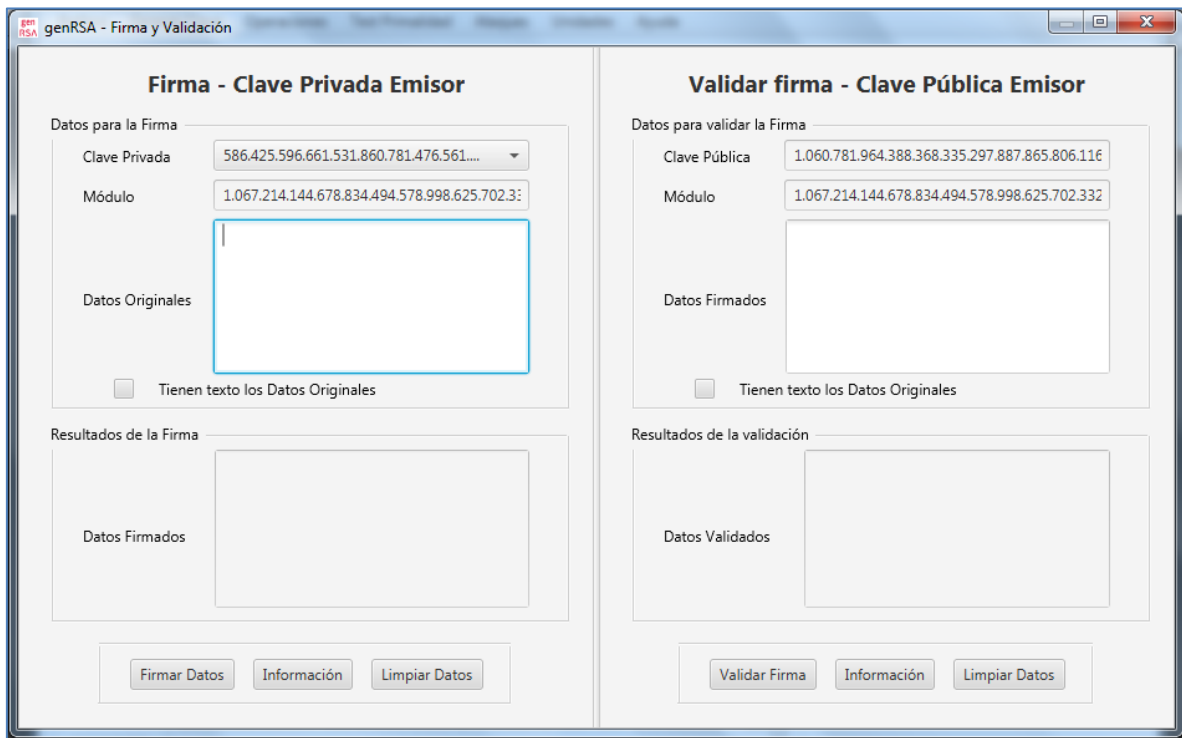
Una vez introducidos los números o el texto se pulsará el botón "Cifrar" y se obtendrán los datos cifrados en la caja de nombre "Datos Cifrados".

- Descifrado: se introducirán datos cifrados en la caja con el mismo nombre. Si los datos cifrados eran originalmente texto se ha de marcar la opción "Tienen texto los Datos Originales". Además se puede usar cualquiera de las claves privadas parejas aparte de la clave privada para descifrar los datos.

A continuación se pulsará el botón "Descifrar" y se obtendrán los datos originales en la caja "Datos Descifrados".

### Operación Firmar – Validar

Para poder acceder a esta funcionalidad previamente se debe haber generado una clave. Una vez generada se pulsará el botón “Operaciones < Firmar\_Validar” situado en la barra de menú. Entonces se abrirá la siguiente ventana.



Esta ventana está dividida en dos: la parte de Firma y la parte de Validación.

- **Firma:** en la caja “Datos Originales” se pueden introducir números decimales o hexadecimales, según sea la clave generada. Estos números deben tener un valor positivo y menor que el módulo. En el caso de introducir un número mayor que el módulo, el número introducido se va partiendo en números lo más cercanos al módulo posible. Es decir, se parte el en números de igual cantidad de bits que el módulo o de un bit menos. Un ejemplo: si el módulo es 65.537 y se introduce el número 5883364551 para firmar, este se partirá en los números 58833 y 64551.

Para evitar confusiones una vez procesados todos los números introducidos se formatea el campo donde se han introducido y se muestran cada número procesado en una línea. De este modo se muestra claramente cómo se deberían haber introducido los datos y cuáles son los datos que realmente se van a firmar. Además se mostrará un mensaje indicando que se han modificado los valores introducidos.

En el caso de introducir texto a firmar se debe marcar la opción “Tienen texto los Datos Originales”. El texto se codificará a números usando el código ASCII, en el cual cada carácter se convierte en números de 8 bits. En este caso será necesario que la clave generada tenga una longitud mínima de 12 bits. Pero al igual que en el caso anterior, si al codificar los datos se obtienen números mayores que el módulo estos se partirán.

Una vez introducidos los números o el texto se pulsará el botón “Firmar” y se obtendrán los datos firmados en la caja de nombre “Datos Firmados”.

- Descifrado: se introducirán datos firmados en la caja con el mismo nombre. Si los datos cifrados eran originalmente texto se ha de marcar la opción “Tienen texto los Datos Originales”.

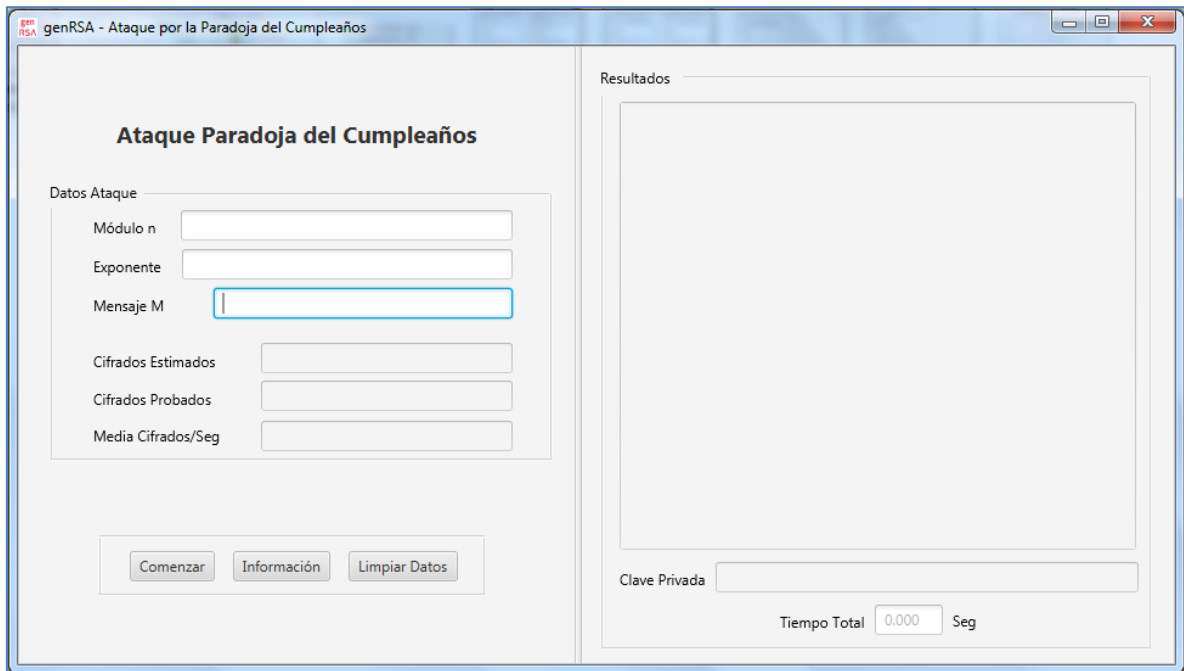
A continuación se pulsará el botón “Descifrar” y se obtendrán los datos originales para que se validen.



### ***Ataque por la Paradoja del Cumpleaños***

Para acceder a la ventana de este ataque no es necesario generar una clave, simplemente se pulsará el botón “Ataques > Paradoja del Cumpleaños” de la barra de menú.

Se debe introducir un mensaje M para realizar el ataque y en caso de no haber generado una clave será necesario rellenar los datos “Modulo n” y “Exponente”.



The screenshot shows a software window titled "genRSA - Ataque por la Paradoja del Cumpleaños". The window is divided into two main sections. The left section, titled "Ataque Paradoja del Cumpleaños", contains a "Datos Ataque" group box with input fields for "Módulo n", "Exponente", "Mensaje M", "Cifrados Estimados", "Cifrados Probados", and "Media Cifrados/Seg". Below these fields are three buttons: "Comenzar", "Información", and "Limpiar Datos". The right section, titled "Resultados", contains a large empty text area for displaying results. At the bottom right of the window, there is a "Clave Privada" input field and a "Tiempo Total" display showing "0.000 Seg".

A continuación se pulsará el botón “Comenzar” y el ataque dará inicio. Durante el ataque se mostrará la media de cifrados realizados por segundo en la caja “Media Cifrados/Seg” y se irán mostrando los resultados del ataque en la parte derecha de la ventana.

No obstante también se mostrará una estimación de los cifrados necesarios para resolver el ataque y el número de cifrados probados que se están realizando durante el ataque.

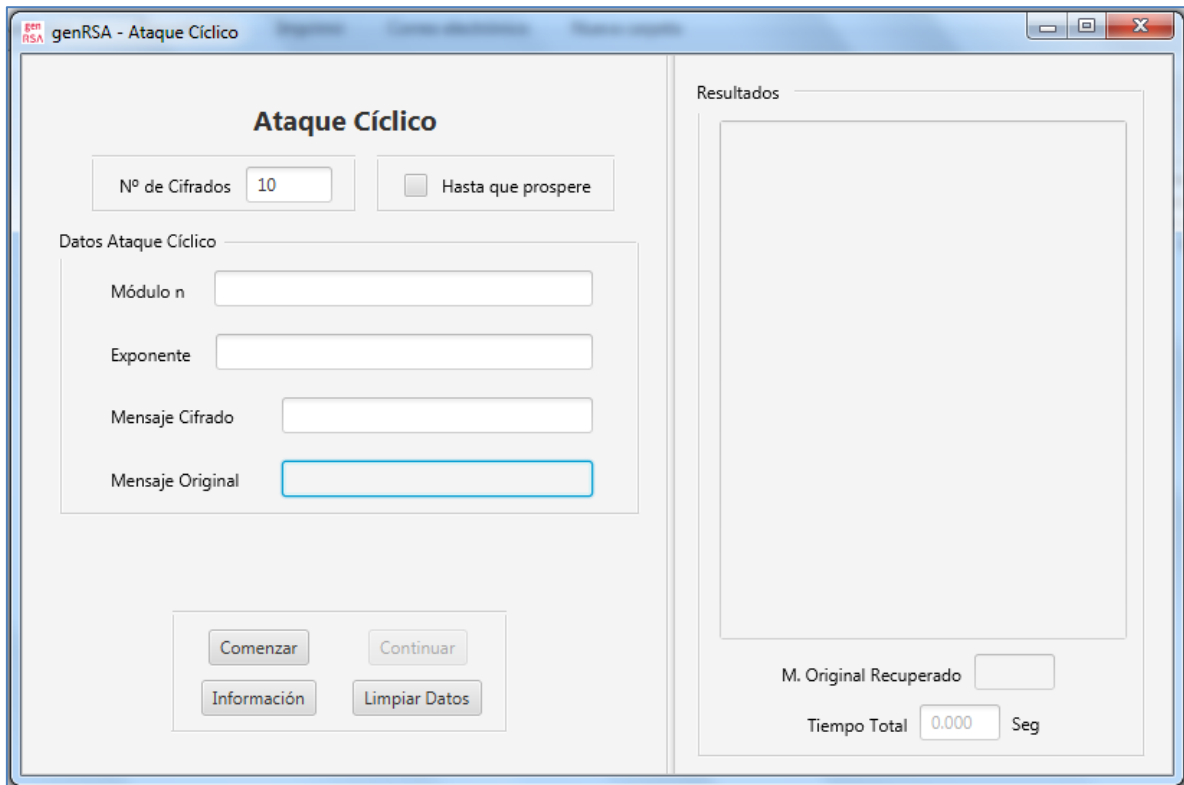
Durante el ataque, se habilitará un botón para que pueda ser parado.

Finalmente, cuando termina el ataque se mostrará la clave privada y el tiempo total empleado para calcularla.

## Ataque Cíclico

Para acceder a la ventana de este ataque no es necesario generar una clave, simplemente se pulsará el botón “Ataques > Cíclico” situado en la barra de menú.

Se debe introducir un mensaje cifrado para realizar el ataque y en caso de no haber generado una clave será necesario rellenar los datos “Modulo n” y “Exponente”.



El ataque se puede ejecutar de dos formas: indicando el número de cifrados o marcando la opción de hasta que prospere. En el caso de seleccionar la opción hasta que prospere el ataque no parará hasta que encuentre el mensaje original o se pulse el botón de parar.

A continuación se pulsará el botón “Comenzar”. El ataque dará inicio realizando cifrados cíclicos al mensaje cifrado con los componentes públicos de la clave. Una vez comenzado el ataque se habilitará el botón de parar el ataque.

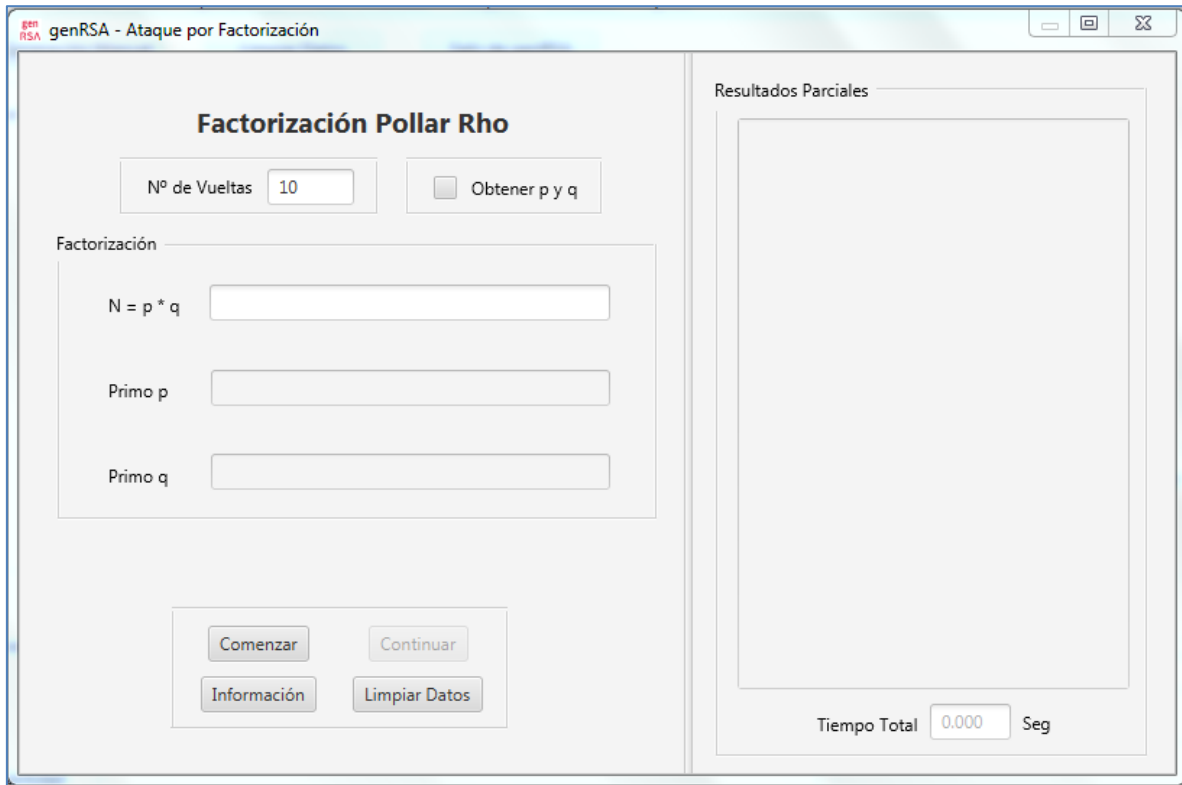
En el caso de no haber recuperado el mensaje original en el número de vueltas indicado se puede continuar el ataque en el mismo punto donde se dejó.

Finalmente, cuando termina el ataque se mostrará el mensaje original recuperado y el tiempo total empleado para calcularla.

### Ataque por Factorización

Para acceder a la ventana de este ataque no es necesario generar una clave, simplemente se pulsará el botón “Ataques > Factorizar n” situado en la barra de menú.

En caso de no haber generado una clave previamente se debe introducir el número que se quiere factorizar en el campo “Módulo n”.



El ataque se puede ejecutar de dos formas: indicando el número de vueltas o marcando la opción “Obtener p y q”. En el caso de seleccionar esta opción, el ataque no parará hasta que se factorice el módulo o se pulse el botón de parar.

A continuación se pulsará el botón “Comenzar” y se comenzarán a obtener resultados parciales del ataque. Una vez comenzado el ataque se habilitará el botón de parar el ataque.

En el caso de no haber factorizado el módulo en el número de vueltas indicado se puede continuar el ataque en el mismo punto donde se dejó pulsando el botón “Continuar”.

Finalmente, cuando termina el ataque se mostrarán los primos p y q y el tiempo total empleado para ejecutar el ataque.

## Anexo 6

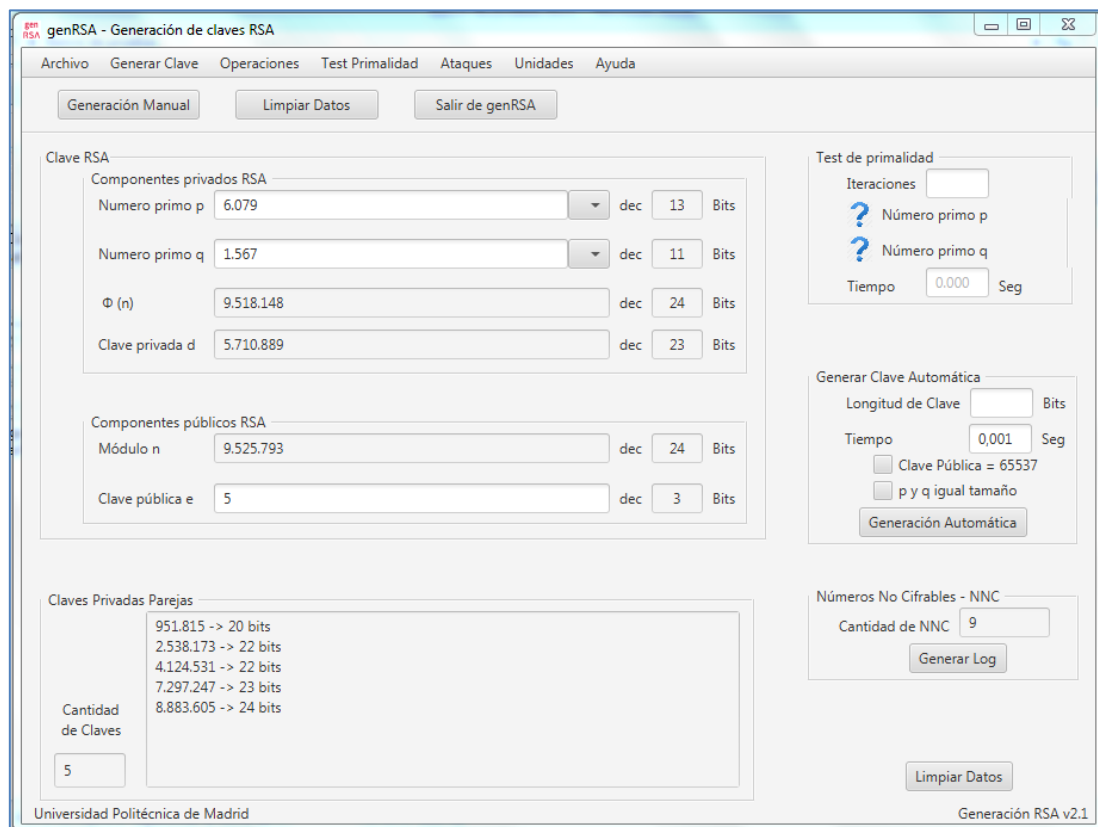
### Banco de Pruebas

Todas las pruebas aquí indicadas se han realizado con un ordenador portátil marca Dell modelo Latitude E5470 con procesador Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz y 8GB de memoria RAM. El sistema operativo instalado es un Windows 7 Enterprise 64-bit.

### Generación Manual

La generación manual permite generar claves RSA decimales o hexadecimales sin importar la longitud en bits de la misma. A continuación se puede ver el ejemplo de una clave decimal de 24 bits:

- Primo p: 6.709
- Primo q: 1.567
- Clave pública e: 5



genRSA - Generación de claves RSA

Archivo Generar Clave Operaciones Test Primalidad Ataques Unidades Ayuda

Generación Manual Limpiar Datos Salir de genRSA

**Clave RSA**

Componentes privados RSA

Numero primo p: 6.079 dec 13 Bits

Numero primo q: 1.567 dec 11 Bits

$\Phi(n)$ : 9.518.148 dec 24 Bits

Clave privada d: 5.710.889 dec 23 Bits

Componentes públicos RSA

Módulo n: 9.525.793 dec 24 Bits

Clave pública e: 5 dec 3 Bits

**Test de primalidad**

Iteraciones: [ ]

? Número primo p

? Número primo q

Tiempo: 0.000 Seg

**Generar Clave Automática**

Longitud de Clave: [ ] Bits

Tiempo: 0.001 Seg

☐ Clave Pública = 65537

☐ p y q igual tamaño

Generación Automática

**Números No Cifrables - NNC**

Cantidad de NNC: 9

Generar Log

**Claves Privadas Parejas**

951.815 -> 20 bits

2.538.173 -> 22 bits

4.124.531 -> 22 bits

7.297.247 -> 23 bits

8.883.605 -> 24 bits

Cantidad de Claves: 5

Limpiar Datos

Universidad Politécnica de Madrid

Generación RSA v2.1

Se han comprobado los resultados de los componentes de la clave obtenidos en genRSA v2.1 (ver imagen 1) realizando los cálculos con ayuda de la web MobileFish:

- Módulo  $n = p \times q = 9.525.793$   

$$n = 6.079 \times 1.567 = 9.525.793$$
- Máximo común divisor (clave pública  $e$ ,  $\Phi(n)$ ) = 1  

$$\Phi(n) = (p-1) \times (q-1) = 6.078 \times 1.566 = 9.518.148$$

$$\text{mcd}(5, 9.518.148) = 1$$
- $\Phi(n)$  mayor que la clave pública y la clave pública mayor que 1.  

$$9.518.148 > 5 > 1$$
- $d = \text{inv}[e, \Phi(n)] = 5.710.889$   

$$d = \text{inv}[5, 9.518.148] = 5.710.889$$

No obstante, también se van a comprobar las claves privadas parejas y los números no cifrables (NNC) asociados a la clave.

- Claves privadas parejas: la clave privada  $d$  es única en  $\Phi(n)$  pero no es el único valor que descifra en  $n$ .

Para comprobar que la cantidad es realmente el número indicado en la aplicación, primero se calcula la primera clave pareja  $d_\gamma$ .

$$d_\gamma = \text{inv}(e, \gamma), \text{ siendo } \gamma = \text{mcm}[(p-1), (q-1)]$$

$$\gamma = \text{mcm}(6.078, 1.566) = 1.586.358$$

$$d_\gamma = \text{inv}(5, 1.586.358) = 951.815$$

A continuación se calcula la cantidad de claves privadas parejas.

$$\lambda = \text{parte entera de } [(n - d_\gamma) / \gamma]$$

$$\lambda = ((9.525.793 - 951.815) / 1.586.358) =$$

$$= (8.573.978 / 1.586.358) = 5$$

Una vez comprobado que la cantidad de claves y la primera coinciden se comprueba el resto de claves, tal que  $d_i = d_\gamma + i \times \gamma$ , donde  $i = 0, 1, \dots, \lambda$

$$\text{Clave 2} = d_1 = 951.815 + 1 \times 1.586.358 = 2.538.173$$

$$\text{Clave 3} = d_2 = 951.815 + 2 \times 1.586.358 = 4.124.531$$

$$\text{Clave Privada} = 951.815 + 3 \times 1.586.358 = 5.710.889$$

$$\text{Clave 5} = d_4 = 951.815 + 4 \times 1.586.358 = 7.297.247$$

$$\text{Clave 6} = d_5 = 951.815 + 5 \times 1.586.358 = 8.883.605$$

- **Números No Cifrables:** la aplicación muestra los números no cifrables asociados y permite generar el fichero de log.

En cuanto a la cantidad, se comprueba que es correcta calculando  $\sigma_n$   
 $= [1 + \text{mcd}(e-1, p-1)] \times [1 + \text{mcd}(e-1, q-1)]$ .

$$\begin{aligned}\sigma_n &= [1 + \text{mcd}(4, 6.078)] \times [1 + \text{mcd}(4, 1.566)] = \\ &= [1 + 2] \times [1 + 2] = 9\end{aligned}$$

Comprobado que la cantidad es correcta se genera el fichero de Log. Este indica que los NNC son: 0, 1, 2.772.023, 2.772.024, 3.981.746, 5.544.047, 6.753.769, 6.753.770 y 9.525.792. Se pueden comprobar que son correctos utilizando la operación de cifrar que ofrece la propia aplicación (más adelante se comprobará el correcto funcionamiento de la funcionalidad de cifrado).

Una vez comprobado que los datos mostrados al generar la clave de forma manual son correctos, se procede a comprobar que se han habilitado todas las funcionalidades de la aplicación que se encontraban deshabilitadas:

- Generación del fichero de Log de NNC.
- Guardar la clave en un fichero HTML.
- Realizar operaciones de Cifrado-Descifrado y Firma-Validación.
- Realizar los tres tipos de ataque con los datos de la clave.

## Test de Primalidad

Los test de primalidad se han de poder ejecutar se haya creado o no una clave. Además deben dejar la aplicación en un estado coherente después de ejecutarse: si se ejecuta sobre los primos de una clave después se debe permitir realizar operaciones de cifra y firma, si se ejecuta sobre números introducidos sin existir una clave estas operaciones no deben permitirse (al igual que ninguna de las funcionalidades que habilita la generación de una clave).

El número de iteraciones de ambos tipos de tests deben estar comprendidas entre 1 y 300. Pudiendo tomarse el valor 50 como número de iteraciones cuyo resultado tiene una fiabilidad sobrada.

Los dos tipos de implementaciones para ejecutar el test de primalidad son: el algoritmo de Fermat y el algoritmo de Miller Rabin combinado con el de Luchas-Lehmer.

El algoritmo de Miller Rabin y Lucas-Lehmer es muy rápido en ejecución. Sin embargo, el algoritmo de Fermat puede llegar a demorarse bastante más cuando el número de iteraciones es alto o la clave tiene una longitud mayor que 2.048 bits.

Para comprobar que los resultados son correctos se han hecho dos grupos de pruebas.

El primer grupo de pruebas consiste en probar números que ya se sabe que son primos. Se comprobará que el resultado indique que el número es primo para ambos algoritmos y con un número de iteraciones igual a 1, 50 y 300.

- Número 1, 20 bits: 963.097
- Número 2, 40 bits: 802.531.780.577
- Número 3, 128 bits:  
206.022.852.512.717.161.155.800.602.892.466.466.027
- Número 4, 512 bits:  
13.302.234.470.614.217.204.793.688.246.321.864.759.812.857.969  
.866.885.120.212.272.507.338.410.462.753.235.484.338.062.389.0  
26.258.357.078.305.228.326.176.576.532.983.080.807.609.059.660  
.259.192.952.449.763

El segundo grupo de pruebas consiste en probar números que se sabe que son compuestos. Estos números serán los módulos de diversas claves que se generarán. La prueba se basará en pasar ambos test de primalidad a números de distinta longitud de bits y comprobar que el resultado indica que no son números primos. Además todos los números se comprobarán para 1, 50 y 300 iteraciones.

- Número 1, 20 bits: 641.737
- Número 2, 40 bits: 737.304.039.083
- Número 3, 128 bits:  
211.279.548.627.169.082.596.241.022.037.888.000.997
- Número 4, 512 bits:  
11.123.175.549.804.993.308.746.373.791.189.175.807.007.447.831  
.932.598.943.746.092.445.552.757.926.599.044.109.456.406.791.6  
12.934.381.949.242.184.333.527.585.839.484.989.304.177.336.008  
.496.437.129.967.769

El resultado de ambas pruebas ha sido satisfactorio. Con lo cual se puede afirmar que los Tests de Primalidad funcionan correctamente.

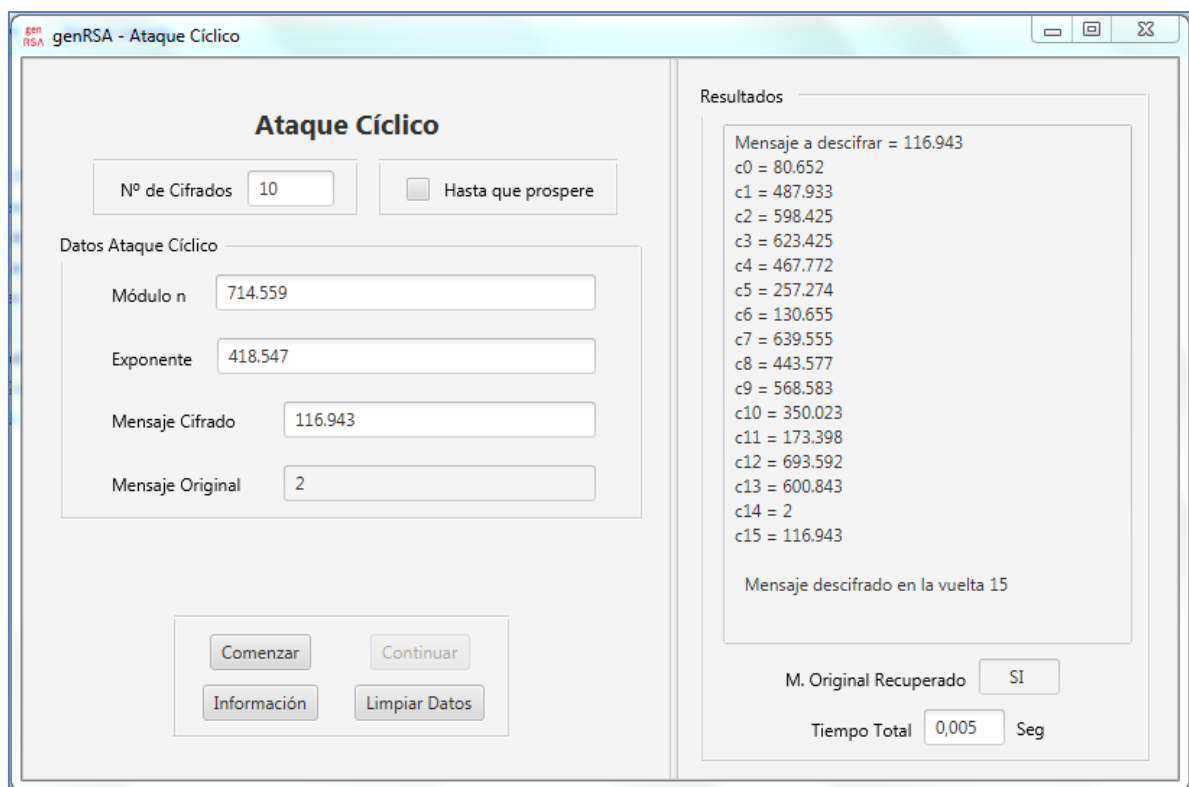


## Ataque Cíclico

El ataque por cifrado cíclico pretende romper el secreto de un mensaje cifrado, es decir, obtener el mensaje en claro. Para ello solo se utilizarán componentes públicos de la clave.

Para obtener el mensaje en claro se van a realizar cifrados sucesivos del mensaje cifrado con la clave pública  $e$  y módulo  $n$ . Cuando se obtenga de nuevo el mismo mensaje cifrado significará que el valor del cifrado anterior es el secreto que da solución al ataque.

A continuación se va a realizar el seguimiento del ataque para el mensaje cifrado 116.943 y la clave cuyo módulo  $n$  es 714.559 y la clave pública es 418.547. Los resultados de este ataque con el software genRSA v2.1 se pueden ver en la siguiente imagen.



**genRSA - Ataque Cíclico**

**Ataque Cíclico**

Nº de Cifrados:  ☐ Hasta que prospere

Datos Ataque Cíclico

Módulo n:

Exponente:

Mensaje Cifrado:

Mensaje Original:

Comenzar Continuar

Información Limpiar Datos

**Resultados**

Mensaje a descifrar = 116.943

$c_0 = 80.652$   
 $c_1 = 487.933$   
 $c_2 = 598.425$   
 $c_3 = 623.425$   
 $c_4 = 467.772$   
 $c_5 = 257.274$   
 $c_6 = 130.655$   
 $c_7 = 639.555$   
 $c_8 = 443.577$   
 $c_9 = 568.583$   
 $c_{10} = 350.023$   
 $c_{11} = 173.398$   
 $c_{12} = 693.592$   
 $c_{13} = 600.843$   
 $c_{14} = 2$   
 $c_{15} = 116.943$

Mensaje descifrado en la vuelta 15

M. Original Recuperado

Tiempo Total  Seg

El mensaje recuperado da como resultado el mensaje 2, pero es el valor cifrado 116.943 el que se ataque:

- $c_0 = 116.943^{418.547} \bmod 714.559 = 80.652$
- $c_1 = 80.652^{418.547} \bmod 714.559 = 487.933$
- $c_2 = 487.933^{418.547} \bmod 714.559 = 598.425$
- $c_3 = 598.425^{418.547} \bmod 714.559 = 623.425$
- $c_4 = 623.425^{418.547} \bmod 714.559 = 467.772$
- $c_5 = 467.772^{418.547} \bmod 714.559 = 257.274$
- $c_6 = 257.274^{418.547} \bmod 714.559 = 130.655$
- $c_7 = 130.655^{418.547} \bmod 714.559 = 639.555$
- $c_8 = 639.555^{418.547} \bmod 714.559 = 443.577$
- $c_9 = 443.577^{418.547} \bmod 714.559 = 568.583$
- $c_{10} = 568.583^{418.547} \bmod 714.559 = 350.023$
- $c_{11} = 350.023^{418.547} \bmod 714.559 = 173.398$
- $c_{12} = 173.398^{418.547} \bmod 714.559 = 693.592$
- $c_{13} = 693.592^{418.547} \bmod 714.559 = 600.843$
- $c_{14} = 600.843^{418.547} \bmod 714.559 = \mathbf{2}$
- $c_{15} = 2^{418.547} \bmod 714.559 = \underline{116.943}$

Tras realizar el seguimiento al ataque se puede concluir que los resultados obtenidos son correctos.

Por último se realizarán diversos ataques al mensaje cifrado = 123.456, en ellos las claves tendrán en común el valor de la clave pública (65.537). En la tabla se muestran el número de vuelta en la que prospera el ataque, el tiempo empleado y la media de cifrados por segundo.

Bits	Módulo n	Vuelta	Tiempo(seg)	Cifrados/Seg
20	1.014.647	5.879	0,298	-
30	725.998.433	1.591.589	2,545	625.378
35	19.054.972.543	2.419.559	3,717	650.944
40	763.494.120.509	16.476.095	23,069	714.209
50	567.208.772.618.027	59.620.319	79,356	751.301

Para poder comprobar que el número de vueltas es correcto se recomienda utilizar la opción de ataque cíclico sin haber generado una clave previamente. Por otro lado, si se quieren obtener los primos p y q se puede realizar el ataque por factorización a los módulos de la tabla.

## Ataque Factorización

El ataque por factorización permite obtener los primos  $p$  y  $q$  que conforman el módulo  $n$ . Para ello hará uso únicamente del algoritmo Pollar rho.

Para comprobar el correcto funcionamiento de este ataque se han generado distintas claves de manera automática y se ha comparado el resultado del ataque por factorización con los primos  $p$  y  $q$  de la generación. En este caso se van a emplear números hexadecimales para ejecutar la prueba.

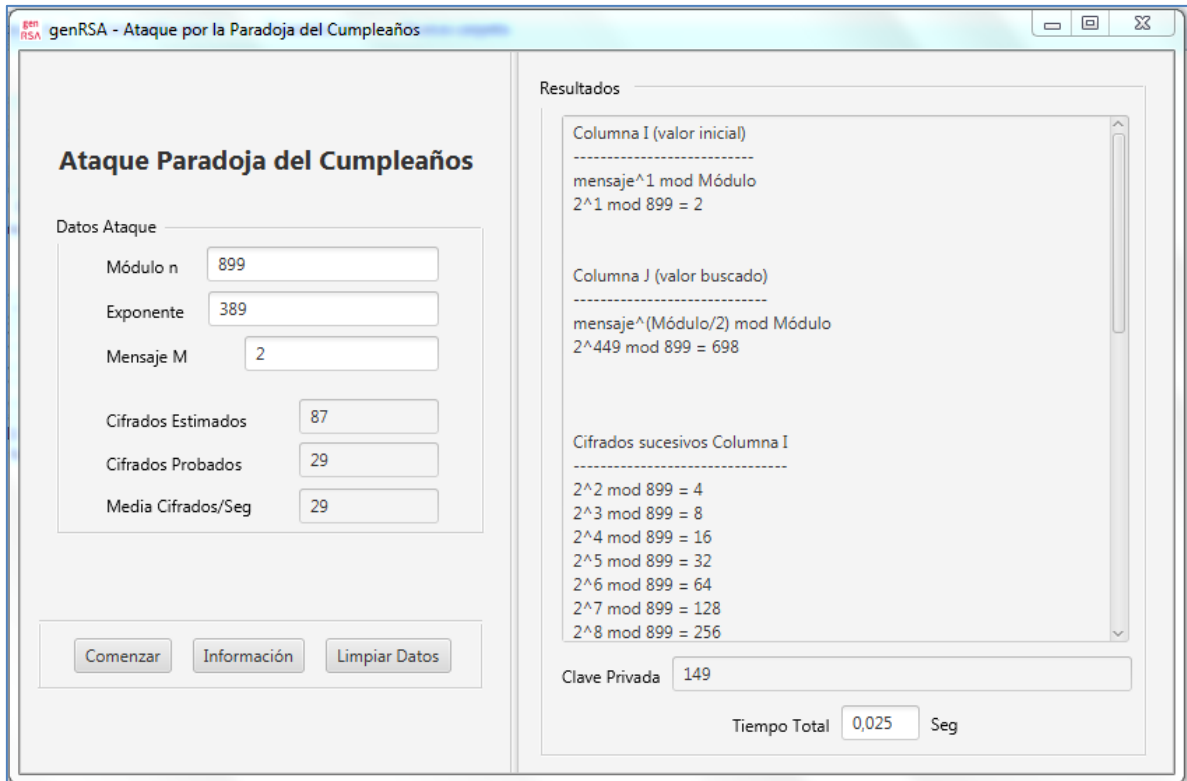
GenRSA v2.1 no cambia de algoritmo conforme se modifican las unidades de la clave (decimal o hexadecimal). Esta premisa es válida para todas las funcionalidades de la aplicación.

Bits	Módulo N	Primo p	Primo q	Tiempo
50	0x 174FF2758CD2D	0x 14442ED1	0x 12679D	0,131 s
75	0x 25941FB5E836ED484BF	0x 1E113D293	0x 13FF3DE84A 5	0,441 s
90	0x 299EDF83183F1D0168DD32 D	0x 19D4CD1A919	0x 19C7ADA217 935	10,813 s
100	0x D610D7F871FED1E2EE74DF 50F	0x 396F4407B9F9	0x 3BA2492384 A947	34,906 s
110	0x 1E1323FF96DC0425914014 A9B483	0x 441558026689 D	0x 7115815891 9379F	4m 1s
115	0x CBD3AAE460F49598742B79 6C5D7F8F	0x EA29F0C555F1 D1	0x DED5825195 72535F	8m 25s

## Ataque Paradoja del Cumpleaños

Para realizar este ataque es preciso haber generado una clave previamente o introducir los valores módulo y clave pública.

Primer caso de ataque: a continuación se procede a comprobar que el resultado es correcto. Datos del ataque: Mensaje  $M=2$ . Datos de la Clave RSA: primo  $p = 31$ , primo  $q = 29$  y clave pública  $e = 389$ .



The screenshot shows the 'genRSA - Ataque por la Paradoja del Cumpleaños' window. It is divided into two main sections: 'Datos Ataque' on the left and 'Resultados' on the right.

**Datos Ataque:**

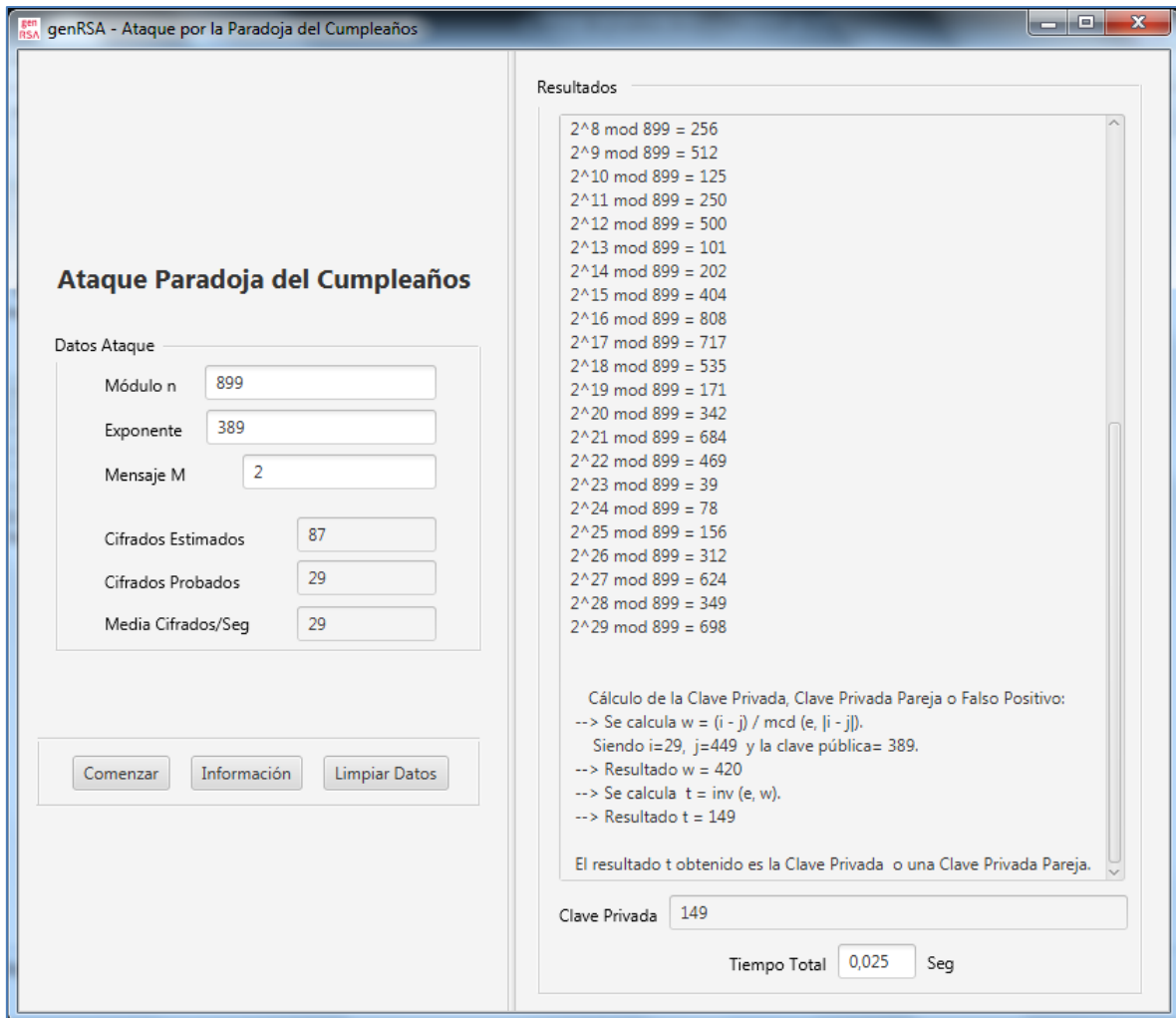
- Módulo n: 899
- Exponente: 389
- Mensaje M: 2
- Cifrados Estimados: 87
- Cifrados Probados: 29
- Media Cifrados/Seg: 29

At the bottom of this section are three buttons: 'Comenzar', 'Información', and 'Limpiar Datos'.

**Resultados:**

- Columna I (valor inicial):**
  - mensaje<sup>1</sup> mod Módulo
  - $2^1 \text{ mod } 899 = 2$
- Columna J (valor buscado):**
  - mensaje<sup>(Módulo/2)</sup> mod Módulo
  - $2^{449} \text{ mod } 899 = 698$
- Cifrados sucesivos Columna I:**
  - $2^2 \text{ mod } 899 = 4$
  - $2^3 \text{ mod } 899 = 8$
  - $2^4 \text{ mod } 899 = 16$
  - $2^5 \text{ mod } 899 = 32$
  - $2^6 \text{ mod } 899 = 64$
  - $2^7 \text{ mod } 899 = 128$
  - $2^8 \text{ mod } 899 = 256$
- Clave Privada:** 149
- Tiempo Total:** 0,025 Seg

En la imagen superior se pueden ver los primeros valores del ataque como son la columna I y el primer valor de la columna J. También se pueden observar los cifrados estimados y los cifrados que se han tenido que probar para obtener la solución a este ataque.



Como se muestra en esta otra imagen, en la vuelta 29 se encuentra el valor 698 que corresponde con el cifrado de la columna J. Ahora se va a comprobar que el cálculo de la clave obtenida sea el correcto.

- $w = (i - j) / \text{mcd} (e, |i - j|)$  tal que  $i=29, j=449$  y  $e=389$ .

$$\begin{aligned}
 w &= (29 - 449) / \text{mcd} (389, |29 - 449|) = \\
 &= -420 / \text{mcd}(389, 420) = -420 / 1 = -420
 \end{aligned}$$

- $t = \text{inv} (e, w) = \text{inv} (389, 420) = 149$
- Se comprueba que t es una clave privada o una clave privada pareja cifrando un número distinto del mensaje M y viendo que se puede descifrar con t.

$$\text{Cifrado de 5: } 5^{389} \bmod 889 = 689$$

$$\text{Descifrado de 689: } 689^{149} \bmod 889 = 5$$

Queda comprobado que el valor  $t$  obtenido es una clave privada pareja o la propia clave privada. Si este valor se compara con la clave que se había generado, se observa que el resultado obtenido es la clave privada.

Clave RSA

Componentes privados RSA

Numero primo  $p$

31

▼

dec

5

Bits

Numero primo  $q$

29

▼

dec

5

Bits

$\Phi(n)$

840

dec

10

Bits

Clave privada  $d$

149

dec

8

Bits

Componentes públicos RSA

Módulo  $n$

899

dec

10

Bits

Clave pública  $e$

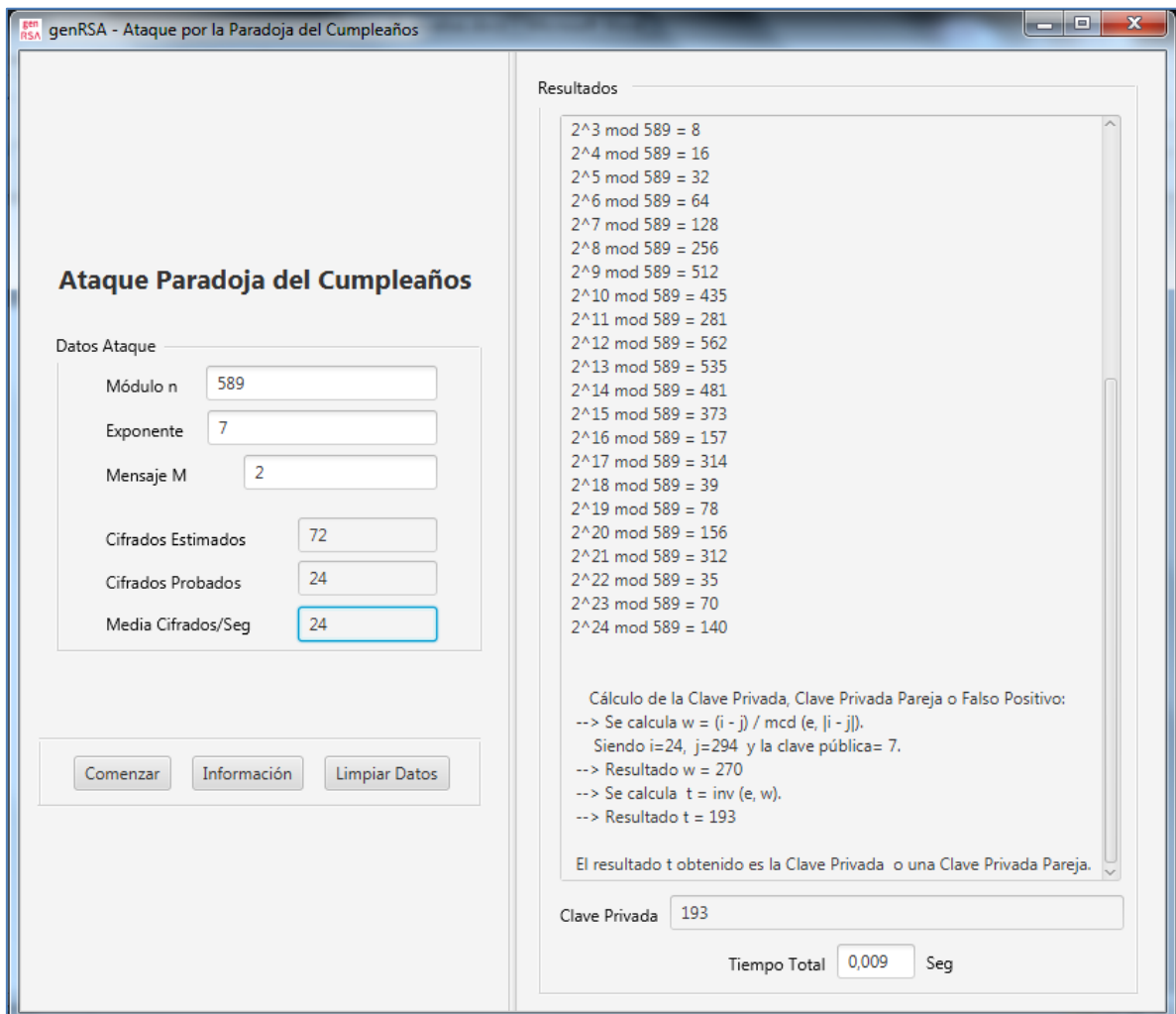
389

dec

9

Bits

Segundo caso de ataque: a continuación se procede a comprobar que el resultado es correcto. Datos del ataque: Mensaje M=2. Datos de la Clave RSA: primo p = 31, primo q = 19 y clave pública e = 7.



**genRSA - Ataque por la Paradoja del Cumpleaños**

**Ataque Paradoja del Cumpleaños**

Datos Ataque

Módulo n: 589

Exponente: 7

Mensaje M: 2

Cifrados Estimados: 72

Cifrados Probados: 24

Media Cifrados/Seg: 24

Comenzar Información Limpiar Datos

**Resultados**

$2^3 \bmod 589 = 8$   
 $2^4 \bmod 589 = 16$   
 $2^5 \bmod 589 = 32$   
 $2^6 \bmod 589 = 64$   
 $2^7 \bmod 589 = 128$   
 $2^8 \bmod 589 = 256$   
 $2^9 \bmod 589 = 512$   
 $2^{10} \bmod 589 = 435$   
 $2^{11} \bmod 589 = 281$   
 $2^{12} \bmod 589 = 562$   
 $2^{13} \bmod 589 = 535$   
 $2^{14} \bmod 589 = 481$   
 $2^{15} \bmod 589 = 373$   
 $2^{16} \bmod 589 = 157$   
 $2^{17} \bmod 589 = 314$   
 $2^{18} \bmod 589 = 39$   
 $2^{19} \bmod 589 = 78$   
 $2^{20} \bmod 589 = 156$   
 $2^{21} \bmod 589 = 312$   
 $2^{22} \bmod 589 = 35$   
 $2^{23} \bmod 589 = 70$   
 $2^{24} \bmod 589 = 140$

Cálculo de la Clave Privada, Clave Privada Pareja o Falso Positivo:

--> Se calcula  $w = (i - j) / \text{mcd}(e, |i - j|)$ .  
 Siendo  $i=24$ ,  $j=294$  y la clave pública= 7.

--> Resultado  $w = 270$

--> Se calcula  $t = \text{inv}(e, w)$ .

--> Resultado  $t = 193$

El resultado t obtenido es la Clave Privada o una Clave Privada Pareja.

Clave Privada: 193

Tiempo Total: 0,009 Seg

Como se observa en la imagen superior, en la vuelta 24 se encuentra el valor 140 que corresponde con el cifrado de la columna J. Ahora se va a comprobar que el cálculo de la clave obtenida sea el correcto.

- $w = (i - j) / \text{mcd}(e, |i - j|)$  tal que  $i=24$ ,  $j=294$  y  $e=7$ .

$$\begin{aligned}
 w &= (24 - 294) / \text{mcd}(7, |24 - 294|) = \\
 &= -270 / \text{mcd}(7, 270) = -270 / 1 = -270
 \end{aligned}$$

- $t = \text{inv}(e, w) = \text{inv}(7, 270) = 193$

- Se comprueba que  $t$  es una clave privada o una clave privada pareja cifrando un número distinto del mensaje  $M$  y viendo que se puede descifrar con  $t$ .

Cifrado de 10:  $10^7 \bmod 589 = 547$

Descifrado de 547:  $547^{193} \bmod 589 = 10$

Queda comprobado que el valor  $t$  obtenido es una clave privada pareja o la propia clave privada. Si este valor, se comparas con la clave que se había generado, se puede observar que el resultado obtenido es una clave privada pareja.

Clave RSA

Componentes privados RSA

Numero primo p	31	dec	5	Bits
Numero primo q	19	dec	5	Bits
$\Phi(n)$	540	dec	10	Bits
Clave privada d	463	dec	9	Bits

Componentes públicos RSA

Módulo n	589	dec	10	Bits
Clave pública e	7	dec	3	Bits

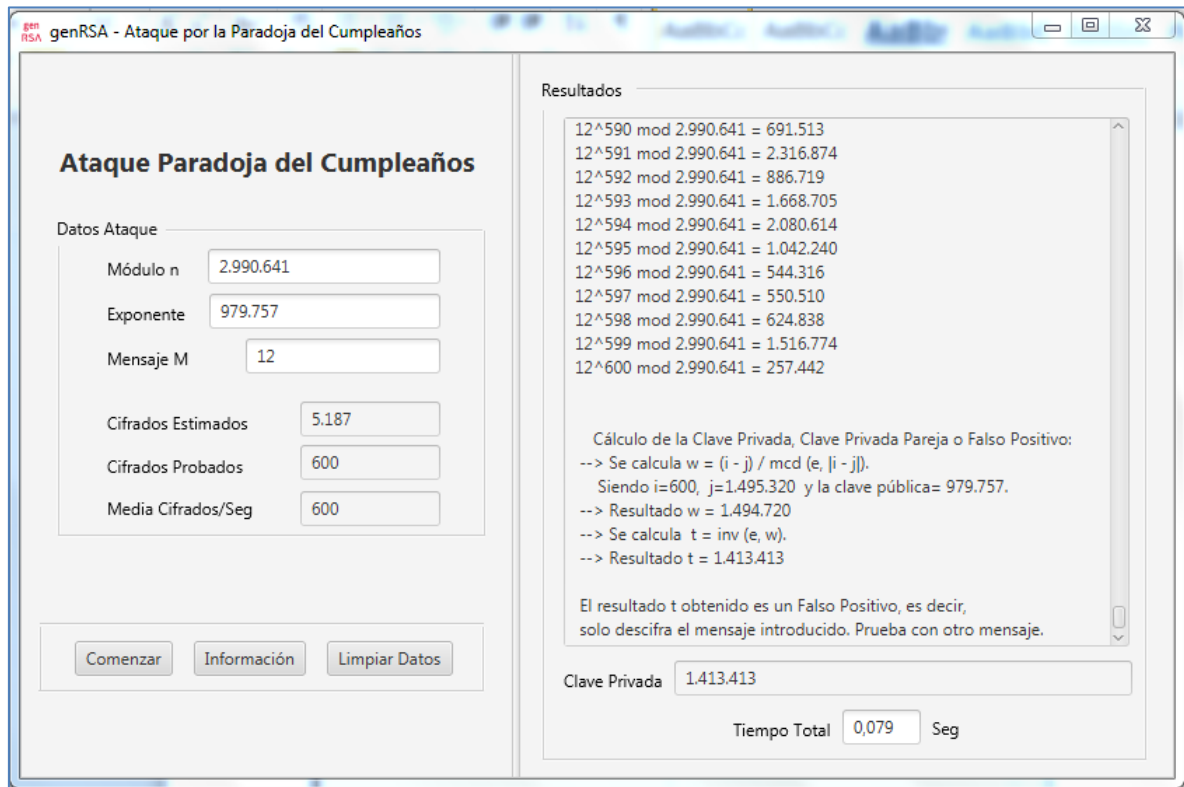
Claves Privadas Parejas

Cantidad de Claves	6	13 -> 4 bits 103 -> 7 bits 193 -> 8 bits 283 -> 9 bits 373 -> 9 bits 553 -> 10 bits
--------------------	---	--



Finalmente, el tercer caso de ataque se da cuando el resultado obtenido es un falso positivo. Un ejemplo se da con la clave formada por los componentes: primo  $p=3.889$ , primo  $q=769$  y clave pública=979.757.

Con dichos datos de clave y el mensaje  $M = 12$ , no se obtiene una clave privada ni una clave privada pareja (ver imagen siguiente). El resultado obtenido solo servirá para descifrar el propio mensaje.



**Ataque Paradoja del Cumpleaños**

Datos Ataque

Módulo n: 2.990.641

Exponente: 979.757

Mensaje M: 12

Cifrados Estimados: 5.187

Cifrados Probados: 600

Media Cifrados/Seg: 600

Resultados

$12^{590} \bmod 2.990.641 = 691.513$   
 $12^{591} \bmod 2.990.641 = 2.316.874$   
 $12^{592} \bmod 2.990.641 = 886.719$   
 $12^{593} \bmod 2.990.641 = 1.668.705$   
 $12^{594} \bmod 2.990.641 = 2.080.614$   
 $12^{595} \bmod 2.990.641 = 1.042.240$   
 $12^{596} \bmod 2.990.641 = 544.316$   
 $12^{597} \bmod 2.990.641 = 550.510$   
 $12^{598} \bmod 2.990.641 = 624.838$   
 $12^{599} \bmod 2.990.641 = 1.516.774$   
 $12^{600} \bmod 2.990.641 = 257.442$

Cálculo de la Clave Privada, Clave Privada Pareja o Falso Positivo:

--> Se calcula  $w = (i - j) / \text{mcd}(e, |i - j|)$ .  
 Siendo  $i=600$ ,  $j=1.495.320$  y la clave pública= 979.757.  
 --> Resultado  $w = 1.494.720$   
 --> Se calcula  $t = \text{inv}(e, w)$ .  
 --> Resultado  $t = 1.413.413$

El resultado  $t$  obtenido es un Falso Positivo, es decir, solo descifra el mensaje introducido. Prueba con otro mensaje.

Clave Privada: 1.413.413

Tiempo Total: 0,079 Seg

Se va a comprobar que el resultado es un falso positivo:

- Se elige un valor dentro del cuerpo de cifra al azar: 318.239
- Se cifra con la clave pública.

$$318.239^{979.757} \bmod 2.990.641 = 2.670.055$$

- Se comprueba que el resultado  $t$  obtenido no descifra este mensaje.

$$2.670.055^{1.413.413} \bmod 2.990.641 = 2.165.631 \text{ ERROR!}$$

- Se comprueba que la clave privada si lo descifra.

$$2.670.055^{2.536.613} \bmod 2.990.641 = 318.239$$

## Cifrado, Descifrado, Firma y Validación

Todas estas operaciones tienen en común que se pueden realizar sobre datos que sean solo numéricos (decimales o hexadecimales) o bien datos que contengan texto (caracteres ASCII).

En el caso de solo querer utilizar estas operaciones con números no se requiere longitud mínima en la clave generada. En el caso de querer realizar alguna de las operaciones en las que los datos contengan texto será necesario haber generado una clave de longitud mayor o igual a 12 bits.

Más aspectos a tener en cuenta son los siguientes:

- Cifra y Firma: Si los datos introducidos no son texto, se introducirá un número por línea. Si el número introducido es mayor que el módulo se dividirá en números de menor o igual valor que el módulo. Si los datos introducidos son texto, se codificarán en ASCII y se cifrarán/firmarán en bloques de bytes. Los bloques serán de tamaño máximo igual al número de bytes del módulo menos uno.
- Descifrado y Validación: Si los datos introducidos no son texto, se introducirá un número por línea. Si el número introducido es mayor que el módulo se dividirá en números de menor o igual valor que el módulo. Si los datos introducidos son texto, se descifrarán/validarán y se codificarán en ASCII. Es el usuario el que debe asegurarse que los datos introducidos tienen caracteres ASCII legibles.

## Cifra

El cifrado de información se realiza empleando la clave pública del receptor y su módulo o cuerpo de cifra.

Se van a realizar cuatro pruebas del cifrado de información. Todas ellas con la misma clave: primo  $p=79$ , primo  $q=103$  y clave pública  $e=4.333$ .

- Primera prueba: Cifrar dos números inferiores al módulo poniendo cada uno en una línea. Estos números son: 4.563 y 1.223.



Comprobación:

$$4.563^{4.333} \bmod 8.137 = 3.505$$

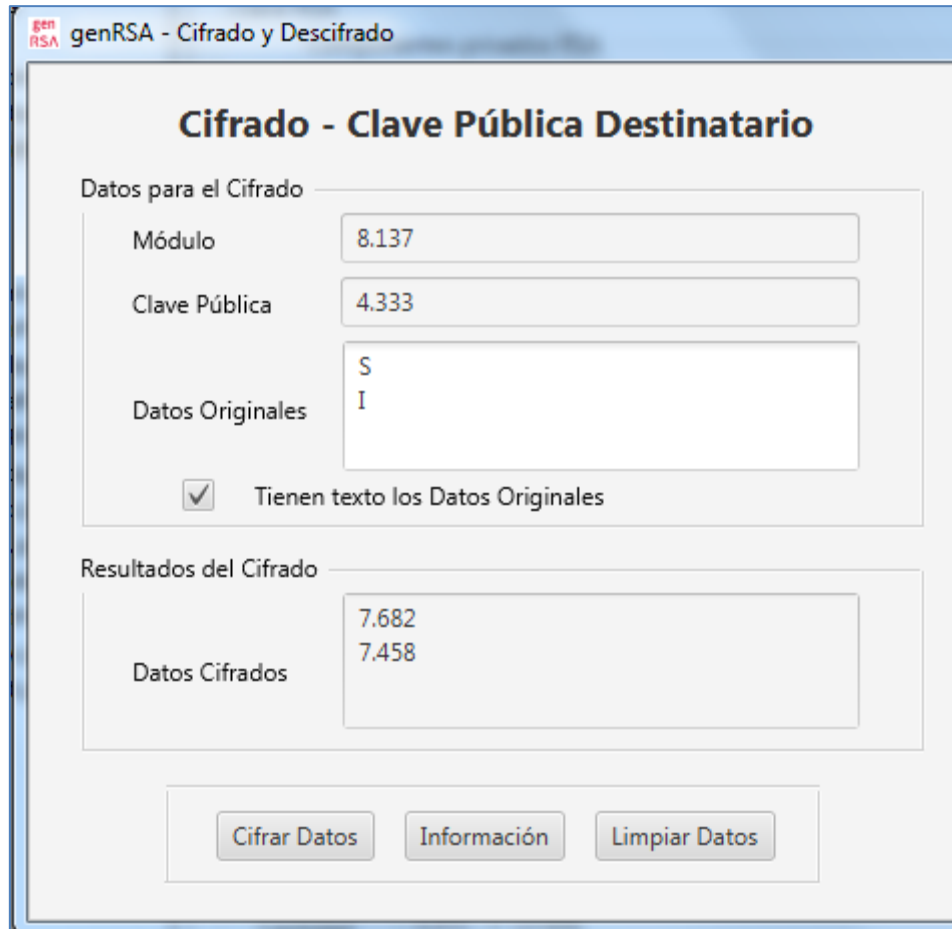
$$1.223^{4.333} \bmod 8.137 = 3.644$$

Comparando estas operaciones con los de la imagen se comprueba que los resultados son correctos.

- Segunda prueba: Se intenta cifrar un número mayor que el módulo. Concretamente, es el número formado por los dos números en claro que se han cifrado en la primera prueba: 45.631.223.

Al intentar ejecutar la operación de cifra se muestra un mensaje por pantalla indicando que no se han introducido de forma correcta los datos. Una vez se pulsa aceptar se puede ver como los datos se han fragmentado formando números inferiores al módulo pero lo más cercanos posible.

- Tercera prueba: cifrar dos caracteres introduciendo cada uno en una línea.



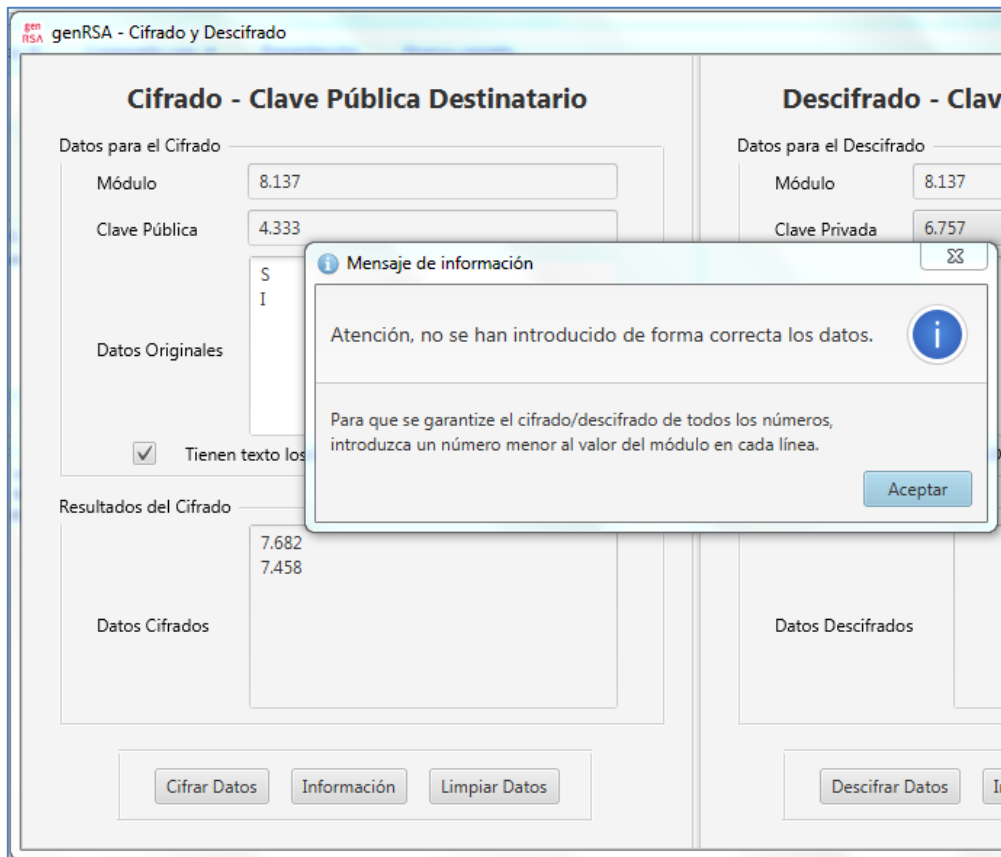
A continuación se procede a comprobar que los datos son correctos. Para ello primero se transformarán a ASCII los caracteres: S = 83 e I = 73.

$$\begin{aligned}
 83^{4.333} \bmod 8.137 &= 7.682 \\
 73^{4.333} \bmod 8.137 &= 7.458
 \end{aligned}$$

Comparando estas operaciones con los de la imagen se comprueba que los resultados son correctos.

- Cuarta prueba: Se intenta cifrar los dos caracteres del apartado anterior, pero esta vez introduciéndolos en la misma línea.

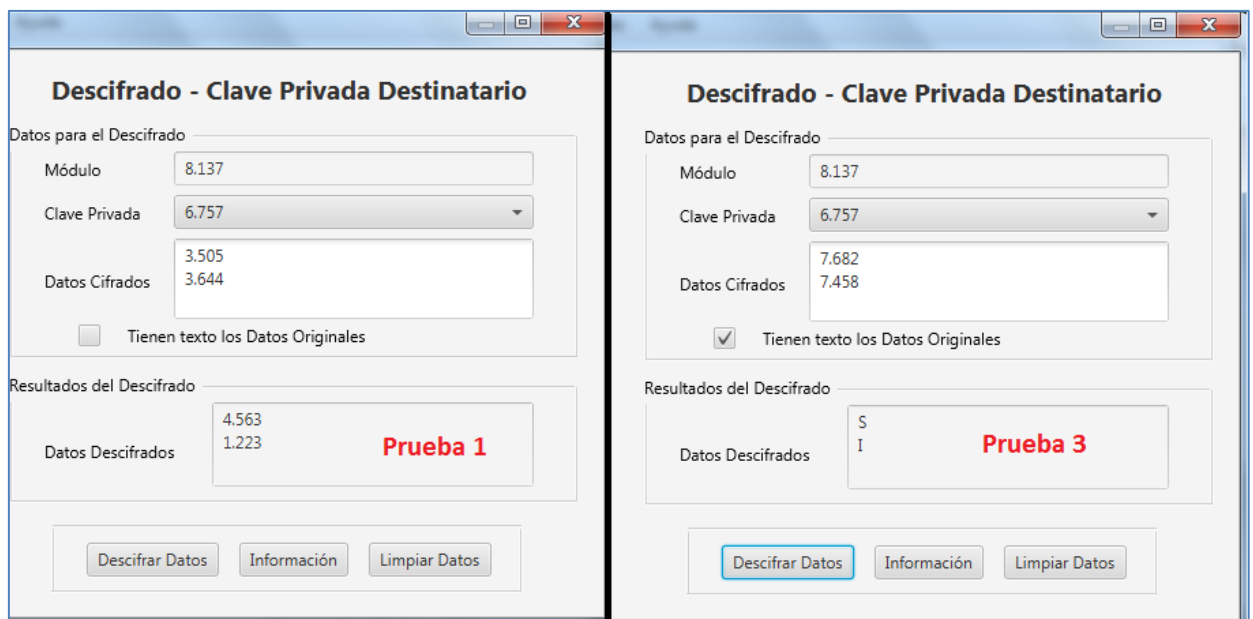
Al intentar ejecutar la operación de cifra se muestra un mensaje por pantalla indicando que no se han introducido de forma correcta los datos. Una vez mostrado el mensaje se puede ver como los datos se han fragmentado quedando cada carácter en una línea. Esto es debido a que cada carácter ocupa 8 bits y el módulo es de tan solo 13 bits.



## Descifrado

El descifrado de información se realiza empleando la clave privada del receptor y su módulo o cuerpo de cifra. En esta ocasión se usará la misma clave que en la cifra.

En la imagen siguiente se puede comprobar que los resultados de las pruebas primera y tercera del apartado de cifra son correctos. Para ello se introducen los datos cifrados de ambas pruebas en el apartado de descifrado de genRSA v2.1. Se comprueba que el resultado obtenido del descifrado es igual al mensaje original antes de realizar el cifrado.



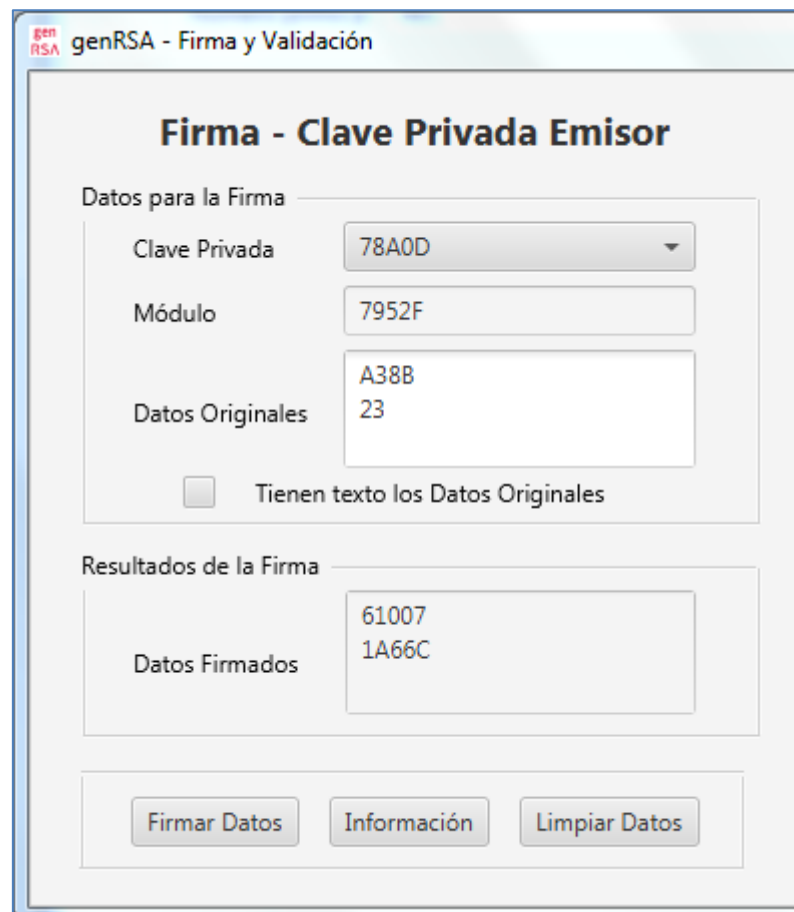
Prueba	Módulo	Clave Privada	Datos Cifrados	Datos Descifrados
1	8.137	6.757	3.505 3.644	S I
3	8.137	6.757	7.682 7.458	S I

## Firma

La operación de firma se realiza empleando la clave privada del emisor (o cualquiera de las claves privadas parejas) y su módulo.

Dado que la cifra y la firma emplean los mismos métodos, en este caso se van a realizar pruebas con números hexadecimales. Para ello lo primero es generar una clave hexadecimal: clave pública  $e=0x\ 6BDC5$ , primo  $p=0x\ 481$  y primo  $q=0x\ 1AF$ .

- Primera prueba: firmar dos valores hexadecimales inferiores al módulo introduciendo cada uno en una línea. Estos valores son:  $0x\ A38B$  Y  $0x\ 23$ .



$$\begin{aligned}
 A38B^{78A0D} \bmod 7952F &= 61007 \\
 23^{78A0D} \bmod 7952F &= 1A66C
 \end{aligned}$$

Tras realizar los cálculos queda demostrado que el resultado mostrado por la aplicación es correcto.

- Segunda prueba: firmar el mensaje HOLA introduciendo los datos en una sola línea. Esta vez la firma se realizará con la clave privada pareja.

El mensaje HOLA al codificarlo en ASCII se obtiene 4 bytes. Pero dado que la clave es de solo 19 bits, no es posible firmar este mensaje sin dividirlo. Con 19 bits, la firma se va a realizar en bloques de 2 Bytes. De este modo el mensaje, a cifrar quedará así: "HO" "LA".

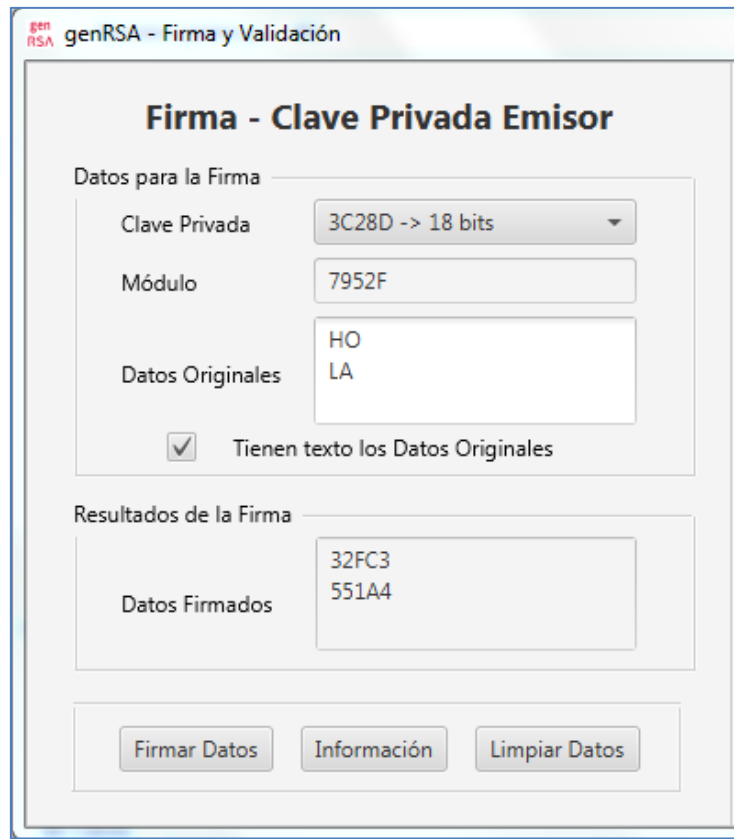
Antes de mostrar la imagen en la que se realiza la firma con la aplicación, se van a realizar los cálculos para obtener los resultados.

**H** = 0x 48, **O** = 0x 4F, **L** = 0x 4C, **A** = 0x 41

"HO" cifrado =  $686F^{3C28D} \bmod 7952F = 32FC3$

"LA" cifrado =  $4C41^{3C28D} \bmod 7952F = 551A4$

La imagen siguiente verifica que los resultados son correctos.

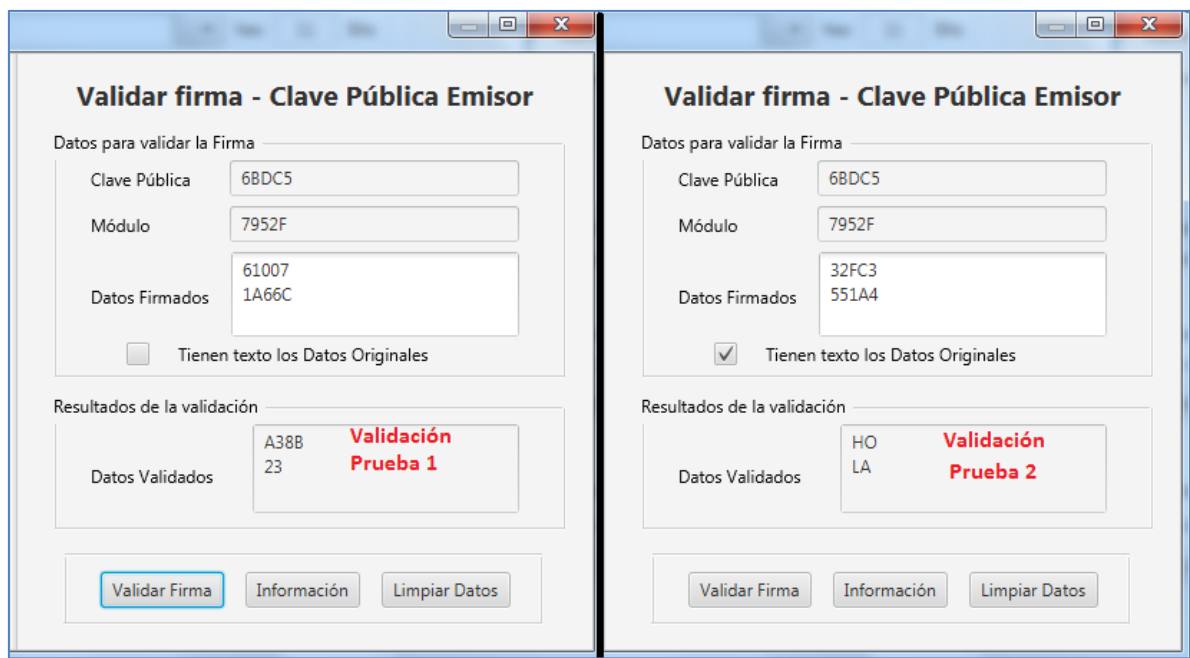




## Validación

La operación de validación se realiza empleando la clave pública del emisor y su módulo. En esta ocasión se usará la misma clave que en la firma.

Con esta operación se puede validar la firma realizada en las pruebas del apartado de firma. Para ello se introducen los datos firmados de ambas pruebas en la caja "Datos Firmados" de la aplicación genRSA v2.1. Una vez obtenido el resultado se valida que es igual a los datos originales introducidos antes de realizar la firma.



Validar firma - Clave Pública Emisor	
Datos para validar la Firma	
Clave Pública	68DC5
Módulo	7952F
Datos Firmados	61007 1A66C
<input type="checkbox"/> Tienen texto los Datos Originales	
Resultados de la validación	
Datos Validados	A38B 23 <b>Validación Prueba 1</b>
<input type="button" value="Validar Firma"/> <input type="button" value="Información"/> <input type="button" value="Limpiar Datos"/>	

Validar firma - Clave Pública Emisor	
Datos para validar la Firma	
Clave Pública	68DC5
Módulo	7952F
Datos Firmados	32FC3 551A4
<input checked="" type="checkbox"/> Tienen texto los Datos Originales	
Resultados de la validación	
Datos Validados	HO LA <b>Validación Prueba 2</b>
<input type="button" value="Validar Firma"/> <input type="button" value="Información"/> <input type="button" value="Limpiar Datos"/>	

