UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Criptoanálisis paralelo de sistemas RSA

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jorge López Ribes

Tutor: Damián López Rodríguez

Curso 2017-2018

Resum

En aquest treball de fi de grau es realitza l'implementació d'un algoritme per criptoanalitzar de forma paral·lela sistemes RSA. Per a fer-ho, s'han analitzat i estudiat els diferents algoritmes de factorització existents, amb l'objectiu de triar un que pugui complir de la manera més òptima les necessitats del treball. Sobre l'algoritme escollit s'han realitzat una sèrie de modificacions i adaptacions per poder ser implementat en un ordinador convencional. També s'ha realitzat un estudi comparatiu dels temps d'execució de l'algoritme implementat respecte a altres tecnologies ja existents, amb l'objectiu d'avaluar el seu rendiment.

Paraules clau: Factorització, RSA, Criptoanàlisi

Resumen

En este trabajo de fin de grado se realiza la implementación de un algoritmo para criptoanalizar de forma paralela sistemas RSA. Para ello se ha realizado un repaso de distintos algoritmos de factorización existentes, de cara a elegir uno que pueda cumplir de forma óptima las necesidades del trabajo. Sobre el algoritmo escogido se han realizado una serie de modificaciones y adaptaciones para poder ser implementado en un ordenador convencional. También se ha realizado un estudio comparativo de los tiempos de ejecución del algoritmo implementado respecto a otras tecnologías ya existentes, con el objetivo de evaluar su rendimiento.

Palabras clave: Factorización, RSA, Criptoanálisis

Abstract

In this final degree project is explained the implementation of an algorithm to cryptanalyze in parallel RSA systems. For that purpose, several factoring algorithms have been analyzed and studied, with the aim of choosing one that optimally meets project requirements. The chosen algorithm has been modified and adapted in order to be implemented in a conventional computer. Moreover, a comparative study of the algorithm's execution time has been made so as to evaluate its performance.

Key words: Factorization, RSA, Cryptanalysis

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En la antigüedad, la criptografía hacía referencia exclusivamente a los procesos de cifrado, basados en convertir información en un texto inteligible y de descifrado, acción inversa a la anterior. David Kahn relata en su obra The Codebreakers de forma cronológica la evolución de la criptografía desde el antiguo Egipto hasta el crucial papel que ésta tuvo en el desarrollo de ambas guerras mundiales durante la primera mitad del sigo XX. Es durante la Segunda Guerra mundial cuando se empieza a hablar del nacimiento de la criptografía moderna con el conocido como Proyecto ULTRA(Blentchley Park, A.Turing) con el objetivo de romper el famoso sistema ENIGMA, coincidiendo además con el desarrollo del primer computador.

Por otro lado, se presenta el concepto de criptoanálisis, que se define como el estudio sobre como descifrar un criptograma en ausencia de su clave, hecho que se puede realizar en el caso de debilidades del texto cifrado, ya sea en la distribución de sus letras o en la frecuencia en la que aparecen. De esta breve explicación sobre el concepto de criptoanálisis se puede concluir que es posible encontrar y descifrar un texto en el caso de que el algoritmo de cifrado presente ciertas debilidades. Es en este contexto donde toma sentido el Principio de Kerckhoffs:

- **Principio de Kerckhoffs**. En 1883, el criptógrafo holandés Auguste Kerckhoffs expuso en su libro La Cryptographie Militaire que la seguridad de un sistema de cifrado debe depender única y exclusivamente de que su clave sea mantenida en secreto, y no del diseño de su algoritmo.

Pese a que en un primer pensamiento parece absurdo pensar que hacer público un algoritmo de cifrado va a aumentar su seguridad, se deben estudiar los beneficios que se pueden obtener a largo plazo. El hecho de que sea un diseño accesible a todo el mundo permite que otros expertos en la materia puedan dar su opinión respecto al algoritmo, o incluso encontrar fallos en su diseño. Este pensamiento podría compararse a la metodología de código abierto que se emplea hoy en día, en la cual se expone públicamente el código fuente en busca de contribuciones y críticas que puedan mejorarlo.

Dicho todo esto, y aceptando que la seguridad de un sistema debe recaer sobre su clave, se pueden distinguir dos principales tipos de sistemas de cifrado, el cifrado de clave simétrica y el cifrado de clave pública o asimétrica.

Los sistemas de clave simétrica son aquellos que usan las mismas claves tanto para el cifrado como para el descifrado del mensaje. Estas claves pueden ser idénticas o ser una transformación entre ambas, hecho que permite obtener fácilmente una de ellas conociendo solo la otra. Uno de los principales inconvenientes que presentan estos sistemas es encontrar un método eficiente para poder intercambiar las claves de forma segura.

Los sistemas de clave asimétrica, explicados más en profundidad a lo largo de la Sección 2.1, ofrecen una solución al problema que presentan los sistemas simétricos a la hora de transmitir la clave de forma segura. Estos sistemas se definen como aquellos que utilizan un par de claves distintas, una pública, que será la que el receptor envíe al emisor para que éste cifre el mensaje, y una privada, que será posteriormente utilizada por el receptor para descifrar el mensaje. En este caso no se requiere de un canal seguro para compartir la clave.

1.1 Motivación

El uso de las tecnologías de la información por parte de la población ha sufrido un crecimiento exponencial en los últimos años. Actualmente, está a la orden del día el uso de estas tecnologías para cualquier tipo de gestión, ya sea realizar una transacción bancaria, compartir datos financieros de una empresa etc.

Como consecuencia de esto, también es necesario la aparición de un conjunto de medidas y protocolos que garanticen la confidencialidad, la accesibilidad, la autenticidad y la integridad de un mensaje. El estudio del estado actual de esta seguridad es la principal motivación que lleva a realizar este trabajo.

Para ello, se realiza un estudio de uno de los sistemas de cifrado más populares en la actualidad, el sistema de cifrado de clave pública RSA. Podemos establecer dos niveles de seguridad en los sistemas de cifrado:

- **Seguridad incondicional**. Obtener la clave es imposible y no depende de la cantidad de recursos computacionales con los que contemos.

- **Seguridad computacional**. Obtener la clave es posible pero exige una cantidad de recursos computacionales que lo hace inviable.

El sistema RSA cumple con la seguridad computacional, ya que se se centra en la dificultad de factorizar números enteros de gran tamaño. A día de hoy, no existe ningún algoritmo de factorización de propósito general capaz de encontrar un factor en un tiempo admisible.

En este trabajo, se va a realizar un estudio de las principales tecnologías de factorización existentes, comentando brevemente sus ventajas e inconvenientes. Posteriormente se implementará un algoritmo basado en la computación paralela que sea capaz de criptoanalizar un sistema RSA, y se comparará a algoritmos ya existentes con el objetivo de evaluar su rendimiento respecto a ellos.

1.2 Objetivos

El propósito de este trabajo es la realización de un estudio sobre las principales características y debilidades del sistema de cifrado RSA, así como la implementación de un algoritmo basado en programación paralela que pueda criptoanalizarlo.

El primer paso para cumplir el objetivo principal consiste en la realización de un análisis y un estudio sobre los algoritmos y herramientas disponibles en la actualidad, con el fin de determinar si pueden ser útiles de cara al trabajo. Posteriormente se escogerá aquel algoritmo que más se adapte a las necesidades y a los requisitos del mismo, y se realizarán las modificaciones y adaptaciones pertinentes para que pueda adecuarse a los objetivos finales del proyecto. En este caso, se ha elegido realizar una implementación convencional del algoritmo cuántico de Shor, basada en programación multihilo e implementado en Java.

Finalmente, se realizará un estudio en base a su tiempo de ejecución. Por un lado, se comparará con otros algoritmos de factorización y por otro, se evaluará como afecta la cantidad de hilos de ejecución al tiempo total del algoritmo.

En resumen, el trabajo persigue lograr los siguientes propósitos:

- Estudio del sistema de cifrado RSA.
- Analizar los algoritmos y herramientas disponibles en la actualidad.
- Escoger un algoritmo que se adecue a las necesidades del trabajo.
- Implementación del algoritmo escogido con sus pertinentes modificaciones.
- Realización de varios casos de estudio con distintos números e hilos.
- Estudio de los tiempos de ejecución y conclusiones obtenidas.

1.3 Estructura de la memoria

Este trabajo está estructurado en seis capítulos. En el Capítulo 2 se explican conceptos necesarios para la comprensión final del trabajo, como son la definición de criptografía de clave pública y sistema RSA. En el Capítulo 3 se exponen los principales algoritmos de factorización existentes, comentando brevemente su funcionamiento, ventajas y desventajas. Además de realizar una breve introducción a la solución propuesta para alcanzar los objetivos del proyecto, en el Capítulo 4 es donde se concentra toda la información relacionada con el diseño y el desarrollo del algoritmo de factorización, comentando aspectos como su arquitectura, su funcionamiento y las tecnologías utilizadas. El Capítulo 5 recoge la información obtenida de la ejecución del algoritmo con diferentes parámetros, así como una pequeña conclusión de los resultados obtenidos. Por último, en el Capítulo 6 se realiza una conclusión final del proyecto y se comentan posibles mejoras a realizar en un futuro.

CAPÍTULO 2
Sistema RSA

En este capítulo, se definen una serie de conceptos imprescindibles de cara a la posterior comprensión de los objetivos principales del proyecto. Más concretamente, se detallarán las características principales de los cifrados de clave pública y de su sistema más conocido, el sistema de cifrado RSA.

2.1 Cifrado de clave pública

La criptografía de clave pública o asimétrica es un método de cifrado basado en una función unidireccional con trampa, en el cual cada usuario posee un par de claves. La clave de cifrado es pública y es la que se encuentra disponible para todo el mundo. Por otro lado, la clave de descifrado constituye la trampa. Pertenece al receptor del mensaje y es privada, sin ella, la resolución del problema no es abordable.

Debido a su elevada complejidad, es muy común utilizar este sistema para transmitir bloques pequeños de información. Es por eso que muchas veces se decide combinar con el sistema de clave simétrica. De esta forma, se transmite de forma asimétrica una clave cifrado simétrico con la que posteriormente se puede acceder a toda la información.

En la Figura 2.1, se ilustra el funcionamiento de este tipo de sistemas. En este caso, Alice quiere enviar un mensaje a Bob, para ello Bob elegirá su par de claves (e,d) y enviará a Alice su clave de cifrado pública (e) y mantendrá en privado su clave de descifrado (d). Posteriormente, Alice envía el mensaje m cifrado con la clave pública de Bob (e), y Bob descifrará este mensaje aplicando el proceso inverso mediante el uso de su clave privada (d).

2.2 Sistema RSA

Desarrollado por R. Rivest, A. Shamir y L. Adleman en 1977, RSA es uno de los sistemas de clave pública más utilizado en la actualidad. Una gran cantidad de protocolos, como SSH, SSL/TLS, OpenPGP, S/MIME emplean RSA tanto para tareas de cifrado como para funciones de firma digital.

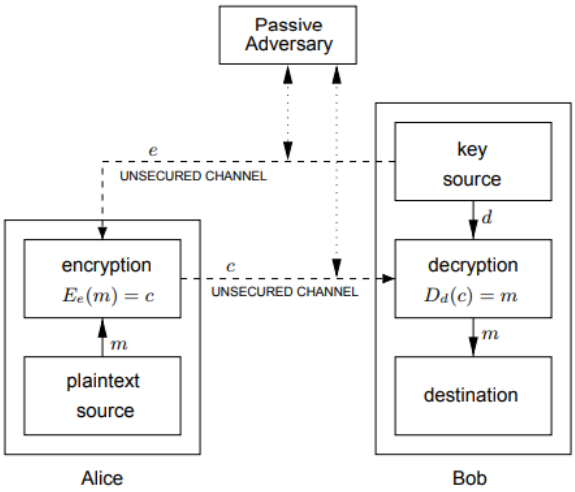

Figura 2.1: Criptografía de clave pública

La popularidad de este sistema de cifrado reside en la dificultad de factorizar enteros de gran tamaño que son producto de dos números primos. Encontrar uno de estos factores en un tiempo admisible no es factible a pesar de los superordenadores que existen en la actualidad.

2.2.1. Operaciones de cifrado y descifrado

- Algoritmo para la generación de claves.

  1. Generar aleatoriamente dos números primos, distintos y de gran tamaño. Estos números p y q deben ser aproximadamente del mismo tamaño.
  2. Calcular n = pq y $\phi = (p-1)(q-1)$
  3. Elegir un entero aleatorio e, $1 < e < \phi$, de forma que $mcd(e, \phi) = 1$.
  4. Utilizar el algoritmo de Euclides extendido para encontrar el único entero d ,$1 < d < \phi$, que cumpla con $ed \equiv 1(mod\phi)$.
  5. Su clave pública es (n,e) y su clave privada es (d).

- Algoritmo de cifrado RSA (B cifra un mensaje m para A).

  1. Cifrado. B realiza lo siguiente:
     (a) Obtener la clave pública de A (n,e).
     (b) Representar el mensaje como un entero m en el intervalo $[0, n-1]$.
     (c) Calcular $c = m^e mod n$.
     (d) Enviar el texto cifrado c a A.
  2. Descifrado. Para recuperar el mensaje m desde c, A debe realizar lo siguiente:
     (a) Utilizar la clave privada d para recuperar $m = c^d mod n$.

2.2.2. Ejemplo práctico del uso de RSA

Para facilitar la comprensión del sistema, se ilustrará con un sencillo ejemplo práctico.

El usuario A genera sus claves RSA eligiendo dos primos: p = 11 y q = 13. El módulo será $n = pq = 143$ y su $\phi = (p-1)(q-1) = 120$. Después elige un entero e que cumpla con la condición $mcd(e, \phi) = 1$. En este caso se elige el número 7, y se calcula su clave privada d= 103 mediante el el algoritmo de Euclides extendido.

Ahora el usuario B pretende enviar al A un mensaje cifrado M, para ello obtiene su clave pública RSA (n,e), (143,7) en este ejemplo. El usuario B envía un mensaje (M) cuyo contenido es el número 9 y lo cifra (C) del siguiente modo:

$$M^e mod\, n = 9^7 mod\, 143 = 48 = C$$

El usuario A recibe el mensaje y lo descifra utilizando su clave privada (d,n), (103,143) en este ejemplo.

$$C^d mod\, n = 48^{103} mod\, 143 = 9 = M$$

2.2.3. Seguridad

Después de toda la información expuesta anteriormente, se puede deducir que RSA es un sistema fuertemente seguro, ya que, aunque se podría llegar a descifrar por fuerza bruta, no se realizaría en un margen de tiempo factible.

Actualmente, no existen algoritmos de propósito general capaces de factorizar números de más de 200 dígitos en un tiempo razonable. Pero para reforzar aún más este argumento, se van a exponer algunas situaciones que un criptoanalista se plantearía para descifrar este sistema y se explicará porqué éste no conseguirá romper el sistema utilizando esos métodos.

Factorizando n

En la siguiente Tabla 2.1, publicada por los autores de RSA en 1978, se muestran varios ejemplos del tiempo que se tardaría en factorizar un número N en base a su longitud. Se asume que las operaciones tardan un microsegundo en realizarse.

En base a estos resultados, los autores de RSA recomendaron que el número n tuviese aproximadamente 200 dígitos de longitud.

En la actualidad, la mayor parte de claves RSA son de 1024 bits (309 dígitos decimales), sin embargo algunos expertos prevén que en un futuro cercano estas claves podrán ser rotas con facilidad, hecho que está llevando a muchas empresas y gobiernos a cambiar a claves de 2048 bits (unos 617 dígitos decimales).

Digits	Number of operations	Time
50	$1{,}4 \times 10^{10}$	3.9 hours
75	$9{,}0 \times 10^{12}$	104 days
100	$2{,}3 \times 10^{15}$	74 years
200	$1{,}2 \times 10^{23}$	$3{,}8 \times 10^{9}$ years
300	$1{,}5 \times 10^{29}$	$4{,}9 \times 10^{15}$ years
500	$1{,}3 \times 10^{39}$	$4{,}2 \times 10^{25}$ years

Tabla 2.1: Tiempos de factorización RSA

Calcular $\phi(n)$ sin factorizar n

La implementación de un método para calcular ϕ, podría romper el sistema RSA, ya que a partir de este valor se podría calcular la clave privada d como el inverso multiplicativo de la clave pública $emod\phi(n)$. Sin embargo, al igual que no hay una forma práctica de factorizar n, tampoco la hay para calcular $\phi(n)$, ya que es aún más costosa incluso.

Determinar la clave privada d, sin factorizar n ni calcular $\phi(n)$

La lógica es similar al apartado anterior. Conociendo d, permite al criptoanalista calcular $ed - 1$, que resultaría en un múltiplo de $\phi(n)$. Miller demostró que un número n puede ser factorizado utilizando alguno de sus múltiplos, sin embargo un criptoanalista no puede determinar d de forma sencilla.

Para ello, sería necesario encontrar una clave d', que resultaría ser equivalente a la clave d, en ese caso sería posible realizar un ataque de fuerza bruta. Sin embargo, encontrar esta d' es tan complicado como factorizar el propio número n.

2.2.4. Conclusiones

Teniendo en cuenta todos los aspectos comentados a lo largo de esta sección, se puede concluir que RSA es un sistema de clave pública que asegura tanto como comunicaciones como firmas digitales, basando principalmente su seguridad en la dificultad requerida para factorizar números de gran tamaño. El hecho de que nadie haya tenido éxito a la hora de romper el sistema en un margen de tiempo aceptable certifica y asegura la seguridad de RSA.

Dicho esto, al focalizar toda su seguridad en la complejidad de la factorización de enteros, en el caso de que en algún futuro cercano se desarrolle un algoritmo de propósito general capaz de factorizar enteros en un tiempo admisible, este sistema quedaría completamente roto y obsoleto.

Se prevé que el tamaño medio de las claves vaya aumentando con el tiempo a medida que se diseñan nuevos algoritmos de factorización más eficientes y las computadoras aumentan su velocidad. Sin embargo, no hay ninguna previsión de que números de 4096 bits puedan ser factorizados en un futuro cercano.

CAPÍTULO 3
Estado del arte

En este capítulo se va a realizar una síntesis de las principales tecnologías de factorización de enteros existentes en la actualidad, comentando brevemente sus características principales, ventajas e inconvenientes.

La factorización de enteros se define como la descomposición de enteros en un producto de enteros más pequeños. En el caso de que estos enteros estén restringidos a ser primos, a este proceso se le conoce como factorización prima. Actualmente, no existe ningún algoritmo que sea capaz de factorizar todos los enteros en un marco de tiempo computacionalmente aceptable. Dejar clara la distinción entre n, que hace referencia a la talla del problema y N al valor modular propiamente. Antes de entrar a explicar cada algoritmo en profundidad, éstos se dividirán en 3 principales grupos, de cara a facilitar su comprensión.

3.1 Propósito específico

Los algoritmos de factorización de propósito específico, son aquellos cuyo tiempo de ejecución depende de las propiedades del número a factorizar, o bien de las de alguno de sus factores.

3.1.1. División por tentativa

El algoritmo de división por tentativa es el algoritmo de factorización más sencillo de implementar. Su principal uso es el de introducir a nuevos usuarios en el mundo de la factorización de enteros, ya que se trata de algoritmo más fácil de entender, además funciona bien con números pequeños.

Su funcionamiento es el siguiente:

1. Dado un entero N compuesto.

2. La primera acción a realizar es dividirlo por todos los primos iguales o menores que N hasta $\sqrt{N}$.

3. Si se encuentra un divisor de N, éste será un factor de N.

4. En caso de no encontrar ningún factor, se demuestra que el número realmente era primo.

Pese a su sencillez, en el caso de que el número a factorizar no tenga un factor primo pequeño, el algoritmo es extremadamente lento debido al elevado coste computacional que presenta. Su complejidad computacional es exponencial:

$$\Theta(n^{1/2})$$

3.1.2. Criba de Eratóstenes

La criba de Eratóstenes es un algoritmo ideado en el Siglo III antes de Cristo por el matemático Griego Eratóstenes de Cirene, que permite listar todos los factores primos menores que un número natural N.

Primero, se escriben en una tabla todos los números naturales entre 2 y N. Empezando en el 2, cuando se encuentra un número primo, se eliminan todos los múltiplos de ese número primo, para posteriormente avanzar al siguiente número que no se haya eliminado. En el momento en que se encuentre un número primo p tal que p^2 >N, la iteración se detendrá. Todos los números no tachados serán primos.

En su peor caso, este algoritmo requerirá $\sqrt{N}$ iteraciones para obtener un resultado, por lo que se puede concluir que no es un algoritmo práctico si se necesita trabajar con números de gran tamaño.

3.1.3. Algoritmo de Fermat

Con excepción de la división por tentativa, el algoritmo de factorización de Fermat es el método más antiguo para factorizar enteros.

El método se basa en la representación de un entero impar como la diferencia de dos cuadrados, es decir, $N = a^2 - b^2$. Esta diferencia es algebraicamente factorizable como (a+b)(a-b), si ninguno de los dos factores es igual a uno, se considera una factorización correcta.

Todo número impar puede presentar esta representación, por tanto si N = cd es una factorización de N, entonces

$$N = (\frac{c+d}{2})^2 - (\frac{c-d}{2})^2$$

Como N es impar, entonces c y d también lo son, así pues, se puede afirmar que esas mitades son enteros

Es considerado un algoritmo de propósito específico puesto que funciona especialmente bien en el caso concreto de que uno de los factores de un número N se encuentre cerca de su raíz cuadrada. Más concretamente, si el factor c difiere menos de $(4N)^{1/4}$ de $\sqrt{N}$ el método solo requiere una iteración.

Pese a que en general no es un algoritmo muy eficiente, pudiendo en su peor caso ser incluso más lento que el algoritmo de división por tentativa, su idea prin-

cipal es en la que están basados otros importantes algoritmos de factorización como por ejemplo la criba cuadrática y la criba general del cuerpo de números, que son los algoritmos de propósito general más conocidos para factorizar semiprimos de gran tamaño.

3.1.4. Algoritmo de Euler

El científico suizo Leonhard Euler, realiza su primera mención al concepto de factorizar un entero utilizando la representación de la suma de dos cuadrados en una carta escrita en 1745 al matemático alemán Christian Goldbach.

Se trata de un algoritmo de propósito específico ya que solo se puede aplicar a aquellos números que pueden ser representados de dos maneras distintas como la suma de dos cuadrados perfectos, es decir $N = a^2 + b^2$. Al igual que en el algoritmo de Fermat, se asume que N es un número impar, así que se puede afirmar que uno de los números es impar (a), y el otro es par (b).

El siguiente paso de la factorización de Euler es encontrar la segunda forma para representar n como la suma de dos cuadrados perfectos, siendo $N = c^2 + d^2$. De nuevo se tiene un número impar c y un número par d.

Por tanto, dado $N = a^2 + b^2 = c^2 + b^2$ se deducen tanto $a^2 - c^2 = d^2 - b^2$ como $(a - c)(a + c) = (d - b)(d + b)$.

Ahora, se tiene un $k = mcd(a - c, d - b)$ siendo k par puesto que a-c y d-b ambos son pares. Así que $a - c = kl$ y$d - b = km$ siendo $mcd(l, m) = 1$. Por sustitución se obtiene $kl(a + c) = km(d + b) \Rightarrow l(a + c) = m(d + b)$, y, como l y m son relativamente primos entonces m debe dividir a+c, lo cual lleva a un $a + c = mr \Rightarrow lmr = m(d + b) \Rightarrow lr(d + b)$. Así que siendo $a + c = mr$ y $d + b = lr$ pero con el $mcd(l, m) = 1$, esto significa que $r = mcd(a + c, d + b)$, y que también es par. En resumen, la factorización buscada es la siguiente:

$$N = ((\frac{k}{2})^2 + (\frac{r}{2})^2)(m^2 + l^2)$$

Este algoritmo es más efectivo que el de Fermat para aquellos enteros cuyos factores no están próximos entre si, y es potencialmente más eficiente que la división por tentativa. Sin embargo, sus restricciones de aplicabilidad a algunos números, éstos deben poder se representados de dos formas distintas como la suma de cuadrados perfectos, le han hecho perder en popularidad respecto a otros algoritmos de factorización.

3.1.5. Algoritmo de Pollard p-1

Antes de explicar este algoritmo, es necesario introducir brevemente los conceptos de B-liso y B-potencia-lisa:

- Un número N se considera B-liso si todos sus factores primos son menores que B, es decir, si $N = p_1^{k_1} p_2^{k_2} ... p_n^{k_n}$, es B-liso si $p_i \leq B \; \forall i = 1...n$.

- Un número N se dice B-potencia-lisa si las potencias de los factores primos de N son menores que B, es decir, Si $N = p_1^{k_1} p_2^{k_2} ... p_n^{k_n}$, es B-potencia-lisa si $p_i^{k_i} \leq B \; \forall i = 1...n$.

El algoritmo de factorización p-1 de Pollard, es un método de propósito específico que permite encontrar eficientemente un factor primo p de un entero compuesto N para el cual p-1 es B-potencia-lisa. Su funcionamiento es el siguiente.

- Algoritmo p-1 para la factorización de enteros.

  1. Elegir un número B-potencia-lisa.
  2. Elegir un entero aleatorio a, $2 \leq a \leq N-1$ y calcular $d = mcd(a, N)$. Si $d \geq 2$ se devuelve d.
  3. Para cada primo $q \leq B, \Rightarrow$ calcular $l = \left\lfloor \frac{\ln n}{\ln q} \right\rfloor$ y calcular $a \leftarrow a^{q^l} mod N$.
  4. Calcular $d = mcd(a-1, N)$.
  5. Si d = 1 o d=N, el algoritmo termina con fallo, en caso contrario, devuelve un valor no trivial d de N.

En el caso de encontrar un número N que tenga un factor primo p, cuyo p-1 se B-liso, el tiempo de ejecución del algoritmo p-1 de Pollard para encontrar el factor P es de $\Theta(B \ln n / \ln B)$ multiplicaciones modulares.

3.1.6. Algoritmo de Pollard rho

El algoritmo rho de Pollard, es un método de factorización de propósito específico, que funciona especialmente bien cuando para un entero de gran tamaño, uno de sus factores es un número pequeño. Fue ideado por el matemático británico John Pollard En 1975

Las ideas principales para encontrar el factor p en un número N son las siguientes:

- Construir una secuencia de enteros x_i que sea periódicamente recurrente mod p.
- Encontrar (i,j) de forma que $x_i \equiv x_j$ mod p.
- Identificar el factor p de N.

Su funcionamiento es el siguiente:

- Algoritmo rho de Pollard para la factorización de enteros.

  1. Elegir $a \leftarrow 2$ y $b \leftarrow 2$.
  2. Para i=1,2,3 ..., hacer:
     2.1. Calcular $a \leftarrow a^2 + 1 mod N$, $b \leftarrow b^2 + 1 mod N$, $b \leftarrow b^2 + 1 mod N$

2.2. Calcular $d = mcd(a - b, N)$.

2.3. Si $1 < d < N$, devolver d y terminar el algoritmo con éxito.

2.4. Si $d = N$, terminar el algoritmo con fallo.

Asumiendo que la función $f(x) = x^2 + 1 mod p$ se comporta como una función aleatoria, el tiempo esperado de ejecución por parte del algoritmo rho de Pollard para encontrar un factor p de n es $\Theta(\sqrt{p})$ multiplicaciones modulares. Esto implica que para encontrar un factor no trivial de n se estima un tiempo de $\Theta(n^{1/4})$ multiplicaciones modulares.

3.1.7. Algoritmo de factorización utilizando curvas elípticas

Algoritmo desarrollado por Hendrik Lenstra en 1987, que factoriza números enteros utilizando la teoría de curvas elípticas. Pese a que utilizado como propósito general, se trata del tercer algoritmo más rápido de factorización de enteros, en la práctica es considerado un método de propósito específico, ya que funciona extremadamente bien con números que tienen factores pequeños.

Este método es una generalización del algoritmo p-1 de Pollard, en el sentido de que el grupo $\mathbb{Z}_p^*$ se remplaza por un grupo aleatorio de curva elíptica sobre $\mathbb{Z}_p$. El orden de este grupo se distribuye de manera uniforme en el intervalo $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$. Si el orden del grupo elegido es B-ligado, el algoritmo encontrará muy probablemente un factor no trivial de N, si no lo es, el algoritmo fallará y se deberá repetir utilizando otro grupo de curva elíptica.

El algoritmo tiene un tiempo de ejecución subexponencial de $[\frac{1}{2}, \sqrt{2}]$, este tiempo depende del factor p y no del tamaño del número, hecho que implica que encuentre primero los factores más pequeños y cosa que le convierte principalmente en un algoritmo de propósito específico como ya se ha mencionado anteriormente.

Su principal uso es para encontrar factores p que tengan igual o menos de 40 dígitos, para números mayores se le da preferencia al algoritmo de criba cuadrática ya que en la práctica es más eficiente.

3.1.8. Criba especial del cuerpo de números

SNFS (Special Number Field Sieve), es considerado el método más eficiente inventado hasta la fecha para factorizar números que cumplan con la condición de $N = r^e \pm s$, siendo tanto r como s enteros pequeños. Los ejemplos más importantes de factorización por parte de este algoritmo son el número 9 de Fermat, $(F_9 = 2^{512} + 1)$, y en número de Mersenne $(M_5 23 = 2^{523} - 1)$.

Este algoritmo pretende encontrar los factores en un tiempo de $L_n[\frac{1}{3}, c]$, siendo $c = (32/9)^{1/3} \approx 1526$.

Existe una versión generalizada de este algoritmo que es aplicable a cualquier valor N, conocida como GNFS (General Number Field Sieve), que será expuesta más adelante.

3.2 Propósito general

3.2.1. Criba general del cuerpo de números

Este algoritmo se corresponde con la generalización del algoritmo SNFS (Subsección 3.1.8) visto anteriormente. Es aplicable a todos los enteros y tiene un tiempo esperado de ejecución de $L_n[\frac{1}{3}, c]$, siendo $c = (64/9)^{1/3} \approx 1923$.

Diferentes experimentos en la década de los 90 concluyeron que a partir de números de 100 dígitos, este método se vuelve el más eficiente dentro de los algoritmos de propósito general, superando así a la criba cuadrática. Ésto se debe a que los números candidatos escogidos en el algoritmo de criba general son por lo general mucho más pequeños que los de la criba cuadrática. Estudios posteriores remarcan que el punto de inflexión donde supera este método a la criba cuadrática se sitúa alrededor de los 115, por lo tanto se puede concluir que este método es el campeón cuando hablamos de algoritmos de factorización de propósito general.

La primera tarea a realizar para poder generalizar este algoritmo es encontrar un campo número algebraico definido por un polinomio irreducible de un grado n, que pueda ser usado para este número N. Debido a que grandes coeficientes en las ecuaciones llevan a cálculos muy pesados, por ello se establece un límite m. El número de ecuaciones algebraicas de grado n con todos sus coeficientes menores que m son m^{n+1}, el cual debe ser del mismo orden de magnitud con N para que pueda haber una probabilidad razonable de que una de estas ecuaciones se pueda utilizar.

Está demostrado que el tiempo de calculo de este algoritmo se minimiza si la n escogida entre $(\frac{3}{2} \ln N / \ln \ln N)^{1/3}$, y escoger un $m \approx N^{1/n}$. Para escribir N como un entero en el sistema de notación posicional en base m:

$$N = \sum_{i=0}^{n} a_i m^i, \text{ con } 0 \leq a_i \leq m - 1.$$

A partir de esto se toma el polinomio f(x) que cumple con los requisitos necesarios. Posteriormente será necesario realizar un estudio sobre el dominio sobre el que se va a aplicar el algoritmo, mediante 3 bases, la base del factor racional, la base del factor algebraico y la base del carácter cuadrático. Como se trata de una tarea bastante compleja, no se profundizará sobre ella en el algoritmo.

3.2.2. Algoritmo de Dixon

Publicado en 1981 por el matemático John D. Dixon, es el algoritmo prototípico cuando hablamos del uso del método base de factores. Es el único método de este tipo cuyo tiempo de ejecución no depende de conjeturas sobre las propiedades de suavidad del polinomio conocido. El algoritmo depende de encontrar dos enteros cuyos cuadrados sean congruentes módulo el entero a factorizar. Es decir, busca encontrar un $x, y \in Zn$ que cumpla con $x^2 \equiv y^2 (mod N)$.

Para factorizar un número N:

Se elige un valor B y se identifica la base de factores P (todos los primos menores o iguales que B). Posteriormente se buscan los enteros positivos que cumplan que $z^2 mod N$ sea liso a B.

$$\text{Para los exponentes adecuados } a_k\text{: } z^2 \equiv \prod_{p_i \in P} p_i^{a_i} (mod N)$$

Una vez generadas suficientes relaciones, por medio del álgebra lineal, por ejemplo la eliminación de Gauss, se multiplican entre si de maner

■

a que los exponentes de los primos del lado derecho son todos pares:

$$z_1^2\ z_2^2 ... z_k^2 = \prod_{p_i \in P} p^{a_{i,1}+a_{i,2}+...+a_{i,k}} (mod N)$$

Ésto nos lleva a una congruencia de cuadrados del tipo $a2 \equiv b2$, que puede desembocar en la factorización de N, $N = mcd(a+b, N)(N/mcd(a+b, N))$. En el caso de obtener una solución trivial, se repetirá el proceso con otra combinación de relaciones, en caso contrario se obtendrán un par de factores no triviales de N, y el algoritmo finalizará.

Pese a que se trata de un algoritmo que no exige ninguna condición al entero a factorizar y ser fácilmente paralelizable, existen en la actualidad algoritmos más eficientes que éste. Un ejemplo sería el algoritmo de criba cuadrática, publicado por Carl Pomerence en 1981, y que se trata de una optimización de este mismo algoritmo de Dixon.

3.2.3. Algoritmo de fracciones continuas

El método de fracciones continuas ideado por Morrison y Brillgart, es uno de los algoritmos de factorización de propósito general más eficiente, hecho que le ha permitido tener un uso extendido en los ordenadores. Fue el primer método en encontrar un factor de N en un tiempo de ejecución subexponencial, convirtiéndose así en el algoritmo más usado durante la década de los 70.

La idea principal de este algoritmo se basa en encontrar una solución no trivial a la congruencia $x^2 \equiv y^2 mod N$, posteriormente se calcula un factor p de N vía el algoritmo de Euclides aplicado sobre $(x+y, N)$.

Para factorizar un número N:

Se debe calcular la expansión de la fracción continua regular de $\sqrt{N}$, al igual que en el método de Shanks (Subsección 3.2.6). Mientras el algoritmo de Shanks espera a encontrar un cuadrado perfecto, Morrison y Brillhart intentan formar combinaciones que saquen un cuadrado multiplicando juntos alguno de los residuos cuadráticos generados. Aunque por norma general este algoritmo es más lento que el de Shanks, podría darse el caso de encontrar un cuadrado con muchos menos ciclos que éste.

3.2.4. Criba cuadrática

Algoritmo de propósito general propuesto por el matemático estadounidense Carl Pomerance en 1981, como una extensión de las ideas de Dixon. Actualmente se trata del algoritmo de factorización más rápido encontrado hasta la fecha, solo superado por la criba general del cuerpo de números cuando se trabaja con números mayores de 100 dígitos.

Eligiendo un número n para factorizar. Siendo $m = \left\lfloor \sqrt{N} \right\rfloor$,y considerando el polinomio $q(x) = (x+m)^2 - N$:

$$q(x) = x^2 + 2mx + m^2 - N \approx x^2 + 2mx,$$

el cual es pequeño respecto a n si x es pequeño en valor absoluto. El algoritmo elige un $a_i = (x+m)$ y comprueba que $b_i = (x+m)^2 - N$ sea p_t-liso. El funcionamiento del algoritmo se explica en los siguientes pasos.

- Algoritmo de factorización de criba cuadrática.
  1. Elegir la base de factores $S = \{p_1, p_2, ..., p_t\}$, donde $p_1 = -1$ and $p_j (j \geq 2)$ es el $(j-1)^o$ primo p para el cual n es un residuo cuadrático módulo p.
  2. Calcular $m = \left\lfloor \sqrt{N} \right\rfloor$.
  3. Juntar t+1 pares (a_i, b_i). Fijar $i \leftarrow 1$. Mientras $i \leq t+1$:
     3.1. Calcular $b = q(x) = (x+m)^2 - N$, y comprobar utilizando la división por tentativa por los elementos en S para los cuales b sea p_t-liso. Si no lo es, se repite este paso.
     3.2. Si b es p_t-liso, sea b =$\prod_{j=1}^{t} p_j^{e_{ij}}$, se establece $a_i \leftarrow (x+m)$, $b_i \leftarrow b$ y $v_i = (v_{i1}, v_{i2}, ..., v_{it})$, donde $v_{ij} = e_{ij} mod 2$ para $1 \leq j \leq t$.
     3.3. $i \leftarrow i+1$.
  4. Utilizar álgebra lineal sobre $\mathbb{Z}_2$ para encontrar un subconjunto $T \subseteq \{1, 2, ..., t+1\}$ de forma que $\sum_{i \in T} v^i = 0$.
  5. Calcular $x = \prod_{i \in T} a_i mod n$.
  6. Para cada j, $1 \leq j \leq t$, calcular $l_j = (\sum_{i \in T} e_{ij})/2$.
  7. Calcular $y = \prod_{j=1}^{t} p_j^{l_j} mod N$
  8. Si $x \equiv \pm y (mod N)$, se debe encontrar otro subconjunto $T \subseteq \{1, 2, ..., t+1\}$ de forma que $\sum_{i \in T} v^i = 0$ y volver al paso 5.
  9. Calcular $d = mcd(x-y, N)$ y devolver d.

3.2.5. Criba racional

El algoritmo de criba racional es un caso especial del algoritmo de criba general del cuerpo de números, es menos eficiente pero a cambio ofrece una implementación más sencilla y es útil para empezar a entender como funciona el algoritmo general que se expondrá en la siguiente sección.

Para factorizar un número N:

Se elige un valor B y se identifica la base de factores P . Posteriormente se buscan los enteros positivos que cumplan que z y z+N sean lisos a B.

Para el exponente a_k: z= $\prod_{p_i \in P} p_i^{a_i}$

Para el adecuado b_k: z+N= $\prod_{p_i \in P} p_i^{b_i}$

Una vez generadas suficientes relaciones, por medio del álgebra lineal se puede llegar a obtener un par no trivial de los factores de N, finalizando así el algoritmo.

3.2.6. Factorización de formas cuadradas de Shanks

El algoritmo SQUFOF (Shank's Square Forms Factorization), es un método para la factorización de enteros creado por el matemático Daniel Shanks como una mejora respecto al algoritmo de Fermat ya comentado en la Subsección 3.1.3. El método hace uso de la expansión de la fracción continua regular de $\sqrt{N}$ bajo la siguiente fórmula:

$$A_{n-1}^2 \equiv (-1)^n Q_n mod N$$

Se expande $\sqrt{N}$ hasta encontrar un cuadrado $Q_n = R^2$ para un valor par de N. Después, la congruencia de Legendre encuentra la solución $x \equiv A_{n-1}, y \equiv R$, y en el caso de que no sea una solución trivial, los factores p y q de N pueden obtenerse por medio del método de Euclides aplicado sobre N y $A_{n-1} \pm R$.

3.3 Computación cuántica

3.3.1. Conceptos previos

Como paso previo a definir el concepto de computación cuántica, se deben exponer brevemente algunos conceptos básicos sobre complejidad computacional. Actualmente, los problemas computacionales se clasifican en distintas clases de complejidad de acuerdo a su dificultad inherente. Un problema es considerado difícil si se requiere una gran cantidad de recursos computacionales(tiempo y memoria) para obtener su solución.

La Figura 3.1 muestra la clasificación de los problemas en base a la Teoría de la Complejidad. Para este proyecto solo será necesario profundizar en las clases P y NP.

La clase de complejidad P contiene a todos los problemas de decisión que pueden ser resueltos por una máquina determinista, utilizando una cantidad polinómica de tiempo computacional. En otras palabras, pertenecen a esta clase los problemas que son eficientemente resolubles o tratables.

Por otro lado, la clase de complejidad NP(non deterministic polynomial time), se utiliza para clasificar aquellos problemas donde una vez obtenida la solución,

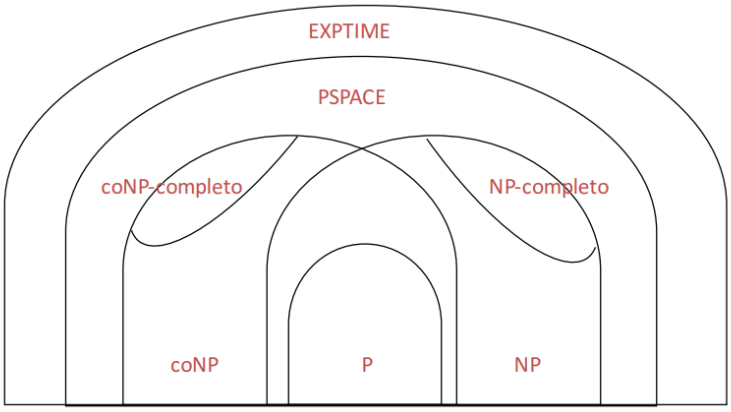

Figura 3.1: Clasificación en base a la Teoría de la Complejidad

ésta es fácilmente verificable en tiempo polinómico, sin embargo este coste polinómico se obtiene en máquinas no deterministas. Es decir, con los ordenadores clásicos actuales este tipo de problemas son irresolubles de forma eficiente.

Es en este contexto donde cobra sentido la computación cuántica, un nuevo paradigma de computación que permite obtenerla resolución de los problemas NP, como por ejemplo el problema de factorización de enteros, en un tiempo polinómico.En la actualidad ya se han presentado algunos algoritmos que hacen uso de este paradigma de computación para descomponer factores de forma eficiente, como es el caso del algoritmo de Shor, sobre el que se profundizará más adelante.

3.3.2. Conceptos básicos

Bits cuánticos

Un bit cuántico, abreviado como qubit, al igual que un bit clásico, puede presentar los estados 0 y 1, representados en notación cuántica como: $|0\rangle$ y $|1\rangle$. Sin embargo, la diferencia real entre un bit y un qubit es que este último, además de los estados anteriores puede obtener otros valores conocidos como superposiciones,

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle,$$

siendo α_0 y α_1 números complejos que deben satisfacer que $|\alpha_0|^2 + |\alpha_1|2 = 1$. Los estados de un qubit son vectores unitarios en un espacio bidimensional complejo.

Para un bit clásico basta con examinarlo para determinar si éste presenta un estado 0 o 1. Sin embargo, para un qubit no basta solo con examinarlo. La naturaleza estocástica de la teoría cuántica muestra que se puede medir un qubit y obtener un resultado 0 con probabilidad $|\alpha_0|^2$ o un resultado 1 con probabilidad $|\alpha_1|2$.

Diferentes experimentos físicos han demostrado que los qubits están presentes como objetos físicos en varios sistemas físico., como por ejemplo los dos estados de un electrón o las diferentes polarizaciones de un fotón.

Un qubit es el sistema cuántico más simple. El sistema cuántico de b qubits se describe como un espacio vectorial complejo de $2^b - dimensiones$ con cada estado de superposición especificado por 2^b amplitudes. Este crecimiento exponencial dificulta la implementación de un sistema cuántico, como consecuencia de la gran cantidad de recursos que se requieren.

Modelo de circuito cuántico

Al igual que un ordenador clásico se construye mediante un circuito eléctrico formado por un sistema de cableado encargado de trasladar la información y un conjunto de puertas lógicas para realizar las tareas computacionales, un ordenador cuántico puede ser creado utilizando un circuito cuántico formado por una estructura de puertas lógicas que permiten realizar computaciones cuánticas y trabajar con información cuántica.

Un circuito cuántico trabaja con b qubits para un entero b. El estado se estructura como $|x_1...x_b\rangle$. Para $x_i = 0o1$, los estados $|x_1...x_b\rangle$ son los estados básicos de un ordenador cuántico.

Mientras que una puerta lógica clásica basa su funcionamiento en alterar los estados de un bit clásico entre 0 y 1, una puerta cuántica opera en qubits. Una puerta cuántica que trabaja sobre b qubits la forman 2^b por 2^b matrices unitarias en el espacio $2^b - dimensional$ de Hillbert.

Un ejemplo de puerta cuántica para 1 qubit sería la matriz unitaria 2x2 que realiza las siguientes transformaciones:

$$|0\rangle \rightarrow \frac{|0\rangle+|1\rangle}{\sqrt{2}}, \qquad |1\rangle \rightarrow \frac{|0\rangle-|1\rangle}{\sqrt{2}}.$$

Transformada cuántica de Fourier

La transformada de Fourier es una transformación matemática empleada para descomponer una señal en todas las frecuencias que la componen. La transformada cuántica de Fourier no es más que una analogía cuántica de la transformación discreta de Fourier, con el objetivo de realizar transformaciones unitarias. Estas transformaciones se realizarán eficientemente mediante una descomposición en un producto de matrices unitarias simples

Esta transformación cuántica se define como una transformación linear de n qubits que mapea los estados básicos $|j\rangle, j = 0, 1, ..., 2^n - 1$, a la superposición de estados:

$$|j\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi ijk/2^n} |k\rangle, \qquad i = \sqrt{-1}$$

Se ha demostrado que mediante la paralelización cuántica la transformada cuántica de Fourier ouede ser realizada en un circuito de tan solo $O(n^2)$ operaciones, mientras que de clásicamente requiere $O(n2^n)$ operaciones para procesar 2^n datos, lo cual supone una aceleración exponencial. Ejemplos exitosos de esto utilizan el algoritmo de Shor para la factorización.

3.3.3. Algoritmo de Shor

Diseñado por el matemático americano Peter Shor en 1994, el algoritmo de Shor es un algoritmo cuántico que habilita la factorización de un número N en tiempo $O((logN)^3)$ y en un espacio $O(logN)$.

La importancia de este algoritmo reside en el hecho de que los principales sistemas de cifrado de clave pública utilizados en la actualidad, como RSA, quedarían obsoletos en el caso de que este algoritmo pudiese ser implementado en la práctica.

Se trata de un algoritmo de carácter probabilístico, es decir, da una solución con alta probabilidad, y la probabilidad de fallo va decreciendo a medida que se repite el algoritmo.

El problema a resolver es el siguiente, dado un número impar y compuesto N, se busca encontrar un entero d en el intervalo [1,N] que divida a N. El algoritmo consta de 2 partes, una primera para descomponer un número en sus factores, la cual puede ser implementada en un ordenador clásico, y una segunda compuesta por un algoritmo cuántico para obtener el periodo.

- Parte clásica

  1. Elegir un número aleatorio a<N
  2. Calcular el mcd(a,N) mediante el algoritmo de Euclides.
  3. Si el mcd(a,N) $\neq 1$, este número a ya es un factor no trivial de N, así que finaliza el algoritmo.
  4. En caso contrario, se debe utilizar un método cuántico para obtener r, el periodo de la siguiente función:
  $$f(x) = a^x mod N.$$
  5. En el caso de que r sea impar, volver al paso 1.
  6. Si $a^{r/2} \equiv -1(modN)$, volver al paso 1.
  7. El $mcd(a^{r/2}+1, N)$ y el $mcd(a^{r/2}-1, N)$ son ambos factores no triviales de N. Finaliza el algoritmo.

- Parte cuántica

  1. Inicializar los registros de q qubits:
  $$Q^{-\frac{1}{2}} \sum_{x=0}^{Q-1} |x\rangle$$
  2. Construir la función cuántica f(x) y aplicarla sobre el estado anterior para obtener:
  $$Q^{-\frac{1}{2}} \sum_x |x, f(x)\rangle.$$
  3. Aplicar la transformada cuántica de Fourier sobre el registro de entrada. Esta transformada se define por:
  $$U_{QFT}|x\rangle = Q^{-\frac{1}{2}} \sum_y w^{xy}|y\rangle.$$
  Esta transformada desemboca en el estado final:

$$Q^{-1} \sum_x \sum_y w^{xy} |y, f(x)\rangle.$$

4. Realizar una medición para obtener un valor y en el registro de entrada y un valor f(x) en el registro de salida.
5. Convertir y/N en una fracción irreductible y extraer el denominador r', candidato a r.
6. Comprobar que $f(x) = f(x+s) \Leftrightarrow a^s \equiv 1(modN)$. Si se cumple, el método termina.
7. En caso contrario, obtener más candidatos de r mediante el uso de múltiplos de s, o utilizando otros s con d/s cercano a y/Q. Si algún candidato funciona, el método finaliza.
8. Si no, volver a empezar desde el paso 1.

CAPÍTULO 4
Diseño y desarrollo de la solución

4.1 Identificación y análisis de soluciones posibles

Tras haber realizado un análisis en profundidad de todos los algoritmos de factorización actuales, además del estudio del sistema de cifrado de clave pública más popular, el sistema RSA, se ha procedido a seleccionar un algoritmo de descifrado que pueda adaptarse a las necesidades de este proyecto.

El abanico de opciones era bastante amplio, puesto que el requisito principal que se persigue es comprobar como responde el sistema RSA a un algoritmo de descifrado basado en multihilo. Así que se podría haber elegido cualquiera de los algoritmos expuestos en el Capítulo 3, y adaptarlo a las exigencias iniciales del proyecto.

Sin embargo, al final se ha decido optar por el algoritmo de Shor, ya que su naturaleza cuántica y el hecho de que podría llegar a romper RSA en tiempo polinómico de ser implementado en una máquina cuántica práctica, han resultado ser unas características lo suficientemente atractivas para que se plantee realizar una implementación convencional de este algoritmo.

4.2 Solución propuesta

Como se ha mencionado en la Sección 4.1, el algoritmo de Shor es el elegido como el punto de partida de la implementación del algoritmo multihilo. Para ello, habrá que pasar por diferentes fases a lo largo de su desarrollo.

En primer lugar se realizará un análisis de las características del algoritmo, donde se establecerán las distintas tecnologías a utilizar para su implementación. Posteriormente se planteará tanto su diseño como su arquitectura, fase en la cual se descompondrá el sistema en bloques en función de su funcionalidad. Una vez definidos, empezará la etapa de programación, en este caso se utilizará el lenguaje de programación Java, debido a las facilidades que ofrece a la hora de trabajar e interaccionar con hilos.

Finalmente, se realizarán una serie de pruebas para asegurar el correcto funcionamiento del algoritmo.

4.3 Teconologías Utilizada

Para implementar el algoritmo de factorización se han utilizado un conjunto de tecnologías y herramientas. A continuación se describen de forma breve y se comenta su utilidad y ventajas.

- **Java**. Es un lenguaje de programación de propósito general, concurrente y orientado a objetos. Se trata de un lenguaje muy simple con un aprendizaje muy sencillo, además ofrece un amplio conjunto de bibliotecas que facilitan mucho el diseño y la programación de aplicaciones. Cualquier sistema operativo capaz de ejecutar una Máquina Virtual de Java será capaz de ejecutar un programa escrito en Java. Este carácter multiplataforma lo ha convertido en uno de los lenguajes de programación más populares del mundo.

  Por otra parte, su API ofrece muchas facilidades a la hora de realizar procesos de forma paralela mediante hilos de ejecución, hecho que ha propiciado que éste sea el lenguaje escogido para el desarrollo de este trabajo.

- **BigInteger**.Librería de java que permite realizar operaciones matemáticas en las que aparecen enteros de gran tamaño que superan el límite marcado por los tipos de datos primitivos. Como los objetivos que persigue el proyecto requieren trabajar con números de muchos dígitos, es estrictamente necesario hacer uso de esta librería.

- **ExecutorService**. Herramienta ofrecida por el JDK (Java Development Kit), que simplifica de forma considerable la ejecución de tareas de forma asíncrona. La herramienta incluye por defecto un conjunto de hilos y una API para asignarles tareas.

  Se trata de una herramienta de mucha utilidad para realizar operaciones paralelas, ya que permite ajustar dinámicamente la cantidad de hilos a ejecutar, las tareas asignadas a cada hilo y gestionar sus paradas .

- **Mathematica**. Programa de gran utilidad en áreas científicas y computacionales. Su soporte para trabajar con números de gran tamaño y su amplia variedad de bibliotecas con funciones matemáticas han sido de gran utilidad a la hora de generar los enteros necesarios para los casos de estudio del proyecto.

- **Thread**. Herramienta ofrecida por el JDK, que habilita en Java la programación multitarea mediante la creación de hilos de ejecución que pueden procesarse de forma concurrente, optimizando así el tiempo de ejecución de una tarea.

4.4 Arquitectura del Sistema

En esta sección se profundiza en la estructura interna que muestra el algoritmo implementado, en primer lugar comentando brevemente sus principales bloques, y posteriormente entrando con más detalle a las distintas partes que lo forman.

La Figura 4.1 muestra los bloques principales en los que se podría dividir el algoritmo, en base a las funciones que se realizan en cada parte. El primer bloque se corresponde con la inicialización de parámetros, en la cual el usuario puede introducir el número que desea factorizar y además, tiene la opción de elegir de forma manual cuantos hilos de ejecución serán lanzados para la resolución del problema.

El segundo bloque se corresponde con la clase principal del algoritmo, ésta es la que se encarga de la administración general del algoritmo, así como del tratamiento de los datos de entrada y de salida. Es aquí donde a partir de los datos de entrada, se crearán los hilos correspondientes y se realizará la invocación al método en paralelo, posteriormente recibirá los resultados obtenidos del método en paralelo y los guardará.

Por último, el tercer bloque se corresponde con los hilos que serán ejecutados en paralelo. Es aquí donde se obtendrá el periodo y se realizarán las comprobaciones pertinentes, si todo va bien y se cumplen las condiciones, la resolución de este método obtendrá uno de los dos factores del número N, y lo enviará a la clase principal.

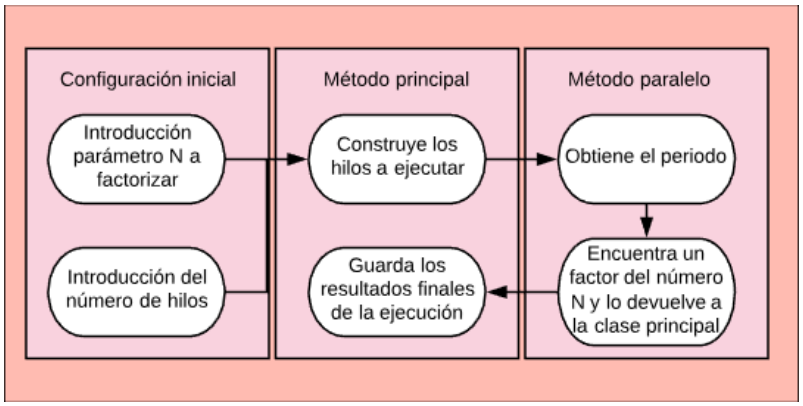

Figura 4.1: Diagrama de la estructura del algoritmo

4.5 Implementación de la clase principal

Como ya se ha mencionado antes de forma breve, esta clase se centra principalmente en el tratamiento de los datos y la gestión de los hilos. En primer lugar, se reciben los parámetros introducidos por el usuario, si el número N recibido es un número correcto, lo transforma al formato BigInteger para poder ser tratado por el algoritmo, en caso contrario informa de un error. Con el parámetro correspondiente al número de hilos, y utilizando el framework Executor Service de Java, se crea un conjunto de hilos que se irán lanzando en paralelo.

Cuando un hilo finaliza, si ha obtenido una respuesta válida ésta será enviada a la clase principal, la cual detendrá la ejecución del resto de hilos y guardará el resultado final para posteriormente comunicárselo al usuario. Si un hilo acaba su ejecución sin obtener una respuesta, simplemente será relanzado por la clase principal con otros parámetros.

También es la clase encargada de monitorizar el tiempo de ejecución total del programa.

4.6 Implementación de la clase en paralelo

En esta clase es donde se realizan los cálculos correspondientes a la obtención del periodo, y la búsqueda de uno de los factores del número N. Recibe un entero a, en el intervalo [1,N-1], y el número N a factorizar en formato BigInteger por parte de la clase principal y comienza la ejecución del hilo.

Es en esta clase donde se recoge toda la carga computacional, ya que se harán una serie de cálculos y comprobaciones con números de gran tamaño. En resumen, las operaciones a realizar son las siguientes:

1. Se construye el hilo con los parámetros a y N y comienza su ejecución.
2. Se comprueba que el $mcd(a, N) \neq 1$, si se cumple, se finaliza la ejecución del algoritmo, puesto que se ha encontrado un valor a no trivial de N.
3. En caso contrario, hay que encontrar el periodo t, que cumpla con

$$a^t \equiv 1 mod N.$$

4. Si el periodo t es impar, el hilo comunica a la clase principal que no ha obtenido una solución y finaliza su ejecución.
5. Si t es par, y se cumple que el $mcd(a^{t/2} + 1, N) \neq N$, se devuelve $mcd(a^{t/2} + 1, N) \neq N$.
6. En caso contrario el hilo finaliza sin obtener una solución y se lo comunica a la clase principal.

4.7 Implementación del algoritmo rho de Pollard

Pese a que el objetivo principal de este proyecto es la implementación de un algoritmo orientado al estudio, y no a ser extremadamente competitivo, si que se pretende que pueda funcionar en unos tiempos de ejecución aceptables. Es por ello que se ha decido utilizar otro algoritmo de factorización como referencia.

Se ha optado por la implementación del algoritmo rho de Pollard, ya que se ha considerado que su programación es bastante asequible y además se adecua bien al tipo de muestras que se van a utilizar para el caso de estudio posterior, ya que es eficiente para tallas pequeñas.

Al igual que el otro algoritmo, su implementación se ha realizado en java, utilizando de nuevo BigInteger para poder trabajar con números de tan gran tamaño.

CAPÍTULO 5
Casos de estudio

En este capítulo se plantean dos casos de estudio diferentes, ambos con el objetivo de ayudar a la mejora del algoritmo implementado. El primer caso busca enfrentar al algoritmo a una situación más real, en la cual se compararán sus tiempos de ejecución a los del algoritmo rho de Pollard. En el segundo estudio se pretende evaluar el número de fallos que presenta el algoritmo con respecto al algortimo de Pollard rho.

En ambos casos se recogerá la información obtenida en distintas tablas y gráficas, con el objetivo de facilitar el análisis de los resultados obtenidos.

5.1 Estudio del comportamiento del algoritmo en comparación con Pollard rho

El principal objetivo que persigue este caso de estudio, es analizar el comportamiento que muestra el algoritmo implementado. Para ello se utilizarán muestras de distintos tamaños y con diferentes números de hilos de ejecución, se obtendrá su tiempo de ejecución y se comparará al algoritmo de Pollard rho.

5.1.1. Definición del caso

Para este caso, se han utilizado números N de distintos tamaño para obtener tiempos de ejecución. Para darle consistencia a los resultados, se ha utilizado un gran número de muestras, más concretamente grupos de 200 números para el intervalo de 32-96 bits y grupos de 10 números para el intervalo 104-128.

Para facilitar la generación de estas muestras se ha decidido utilizar el software Mathematica. Este software ofrece la posibilidad de utilizar la función RandomPrime[tamaño, n], que es básicamente una implementación eficiente del algoritmo de generación de pseudoprimos de Miller-Rabin, mediante el cual dado un tamaño de bits y un número n, devolverá n números primos de b bits.

Como se pretende utilizar números semiprimos para las pruebas, bastará con realizar un producto entre cualquiera de los números generados anteriormente. Por ejemplo, para obtener un número semiprimo de bits, se realizará el producto

de dos números primos de 32bits. Estas operaciones también se han realizado utilizando Mathematica.

5.1.2. Resultados obtenidos

Debido al gran incremento que presentan los tiempos de ejecución, y con el objetivo de que la representación visual quede lo más claro posible, se ha optado por dividir los resultados en dos gráficas diferentes. En la primera de ellas se recogerán los números de menor tamaño, es decir, las muestras de 32,48,64,80 y 96 bits. Por otro lado, la segunda gráfica recogerá las muestras de 104,112,120 y 128 bits.

La Tabla 5.1 muestra los resultados numéricos obtenidos tras la ejecución de las primeras muestras. Por otro lado, la Figura 5.1 muestra de una forma más visual los resultados, además se puede observar de una forma más clara la complejidad de tipo exponencial que presenta la factorización de enteros.

Bits	**Pollard rho**	**2 hilos**	**4 hilos**	**8 hilos**
32	59ms	63ms	65ms	69ms
48	76ms	88ms	87ms	90ms
64	174ms	197ms	199ms	203ms
80	1698ms	2074ms	1945ms	1873ms
96	20059ms	27134ms	25542ms	23768ms

Tabla 5.1: Resultados obtenidos en milisegundos

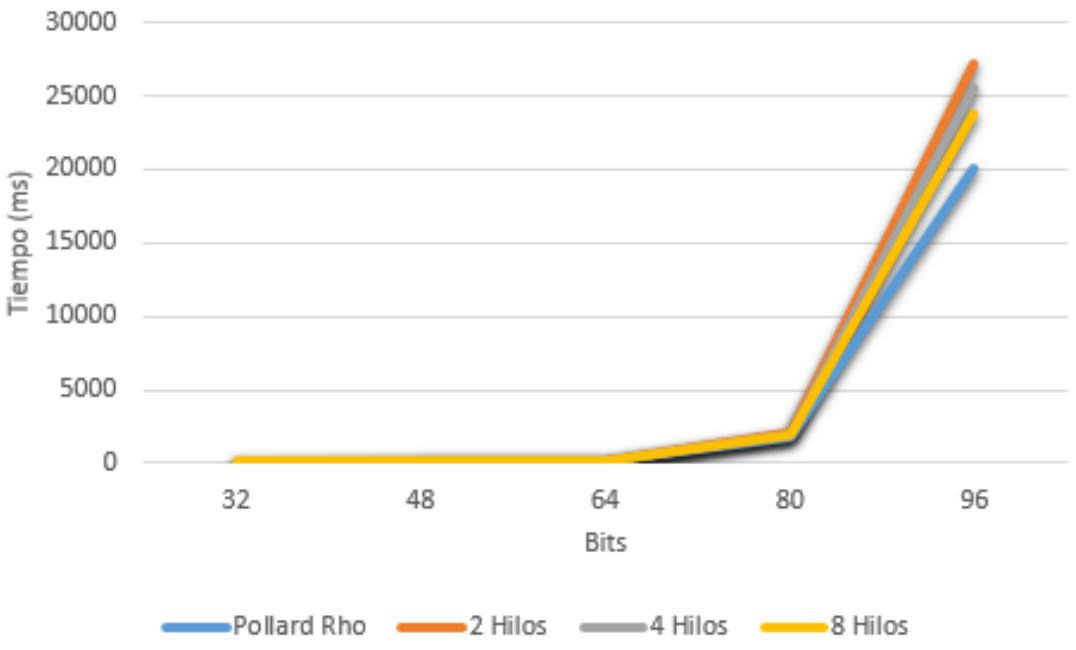

Figura 5.1: Gráfica de resultados de el intervalo 32-96 bits

Por otro lado la Tabla 5.2 recoge los resultados de los números de mayor tamaño. En este caso se ha optado por ir escalando de 8 en 8 bits, ya que los saltos de tiempo requerido son demasiado elevados para mantener la progresión.

Bits	**Pollard rho**	**2 hilos**	**4 hilos**	**8 hilos**
104	58,427s	92,568s	86,844s	79,532s
112	328,2s	5 25,985s	477,695s	447,938s
120	980,675s	1875,456s	1459,602s	1230,432s
128	1983,174s	3152,673s	2896,543s	2475,045s

Tabla 5.2: Resultados obtenidos en segundos

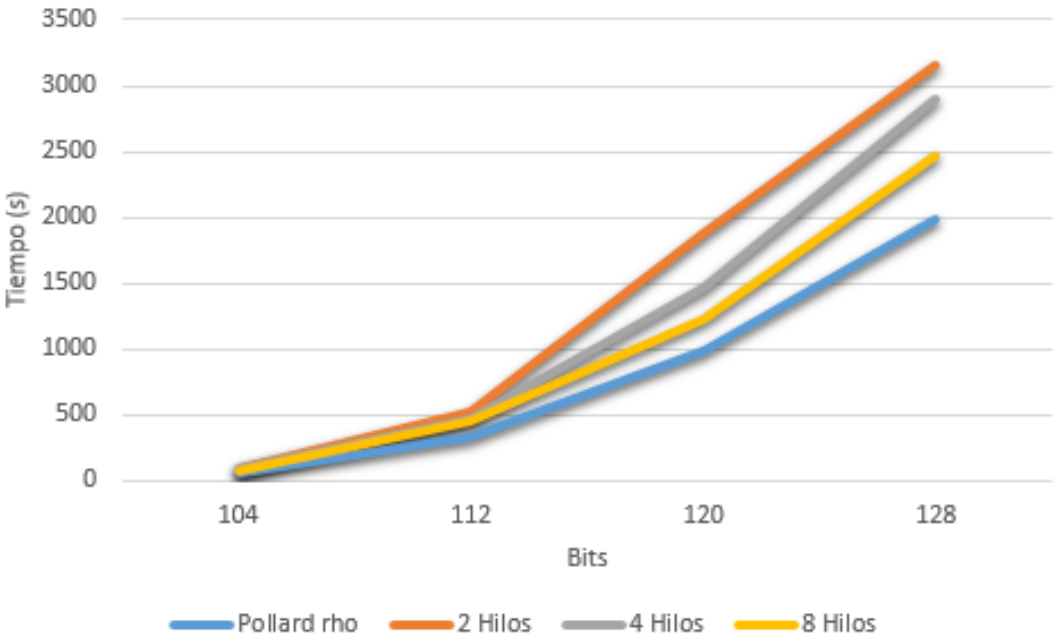

Figura 5.2: Gráfica de resultados de el intervalo 104-128 bits

5.1.3. Evaluación de los resultados

De los resultados obtenidos en el apartado anterior se pueden sacar varias conclusiones. Pese a que el algoritmo tiene unos promedios de tiempo inferiores al utilizado como referencia, este si que muestra una consistencia en los resultados.

Además, como ya se expuso con anterioridad, no se perseguía la obtención de un algoritmo extremadamente competitivo, si no de estudiar como se comportaría un algoritmo en paralelo a la hora de intentar romper el sistema RSA, y con estas pruebas se ha conseguido ese objetivo.

Otro aspecto a considerar es como se puede observar una mejora de tiempo a medida que se aumenta la cantidad de hilos, por lo que se podría esperar que si se realizase un estudio para 16 o 32 hilos, los resultados finales se acercarían más a Pollard rho, o incluso llegarían a ejecutarse en un tiempo menor.

Eficiencia en base al número de hilos

Además, el hecho de haber incluido la opción de poder utilizar un diferente número de hilos, ha habilitado también entender mejor como funciona la programación concurrente, y entender mejor cuando son necesarios más o menos hilos de ejecución.

Como se puede observar de los resultados obtenidos en la Tabla 5.1 y en la Figura 5.1 para los números de tamaño pequeño el aumento de hilos no supone ninguna mejora, llegando incluso a ralentizar un poco la ejecución debido al coste que conlleva la ejecución de un hilo adicional.

Sin embargo, para números mayores de 64 bits, ya se empieza a notar una mejora considerable respecto a usar 2,4 u 8 hilos, mejora que supone un ahorro cada vez mayor a medida que aumenta la complejidad del número a factorizar.

5.2 Estudio comparativo de los errores de cada algoritmo

5.2.1. Definición del caso

Para este segundo caso, se pretende evaluar el porcentaje de fallo del algoritmo y compararlo al porcentaje que presenta el algoritmo de Pollard rho.

Se entiende como fallo que el algoritmo no sea capaz de factorizar el número, o bien que sea capaz de hacerlo pero en un tiempo excesivo para lo requerido en este estudio. Para ello, se establecerá un timeout que marcará la línea entre la factorización y el error.

Se utilizarán muestras entre 104 bits y 128 bits, ya que con números más pequeños la probabilidad de que aparezca un error es mínima.

5.2.2. Resultados obtenidos

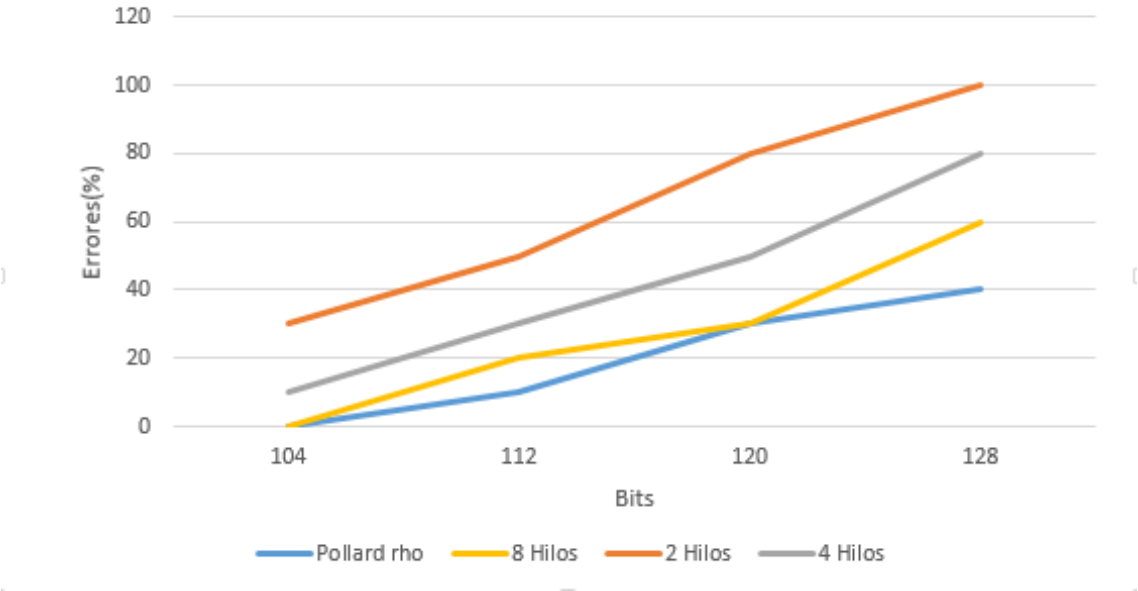

Figura 5.3: Gráfica de errores de el intervalo 104-128 bits

La Figura 5.3 muestra el porcentaje de los números que han tardado excesivo tiempo en factorizarse, se han utilizado 10 muestras para cada tamaño.

5.2.3. Evaluación de los resultados

Como se puede observar, el algoritmo implementado presenta una tasa de error algo superior a la del algoritmo de Pollard rho. De este hecho se puede concluir que a menos que se consiga realizar alguna mejoras para reducir esta tasa, éste no se trata del algoritmo más interesante para utilizar como algoritmo de propósito general para factorizar enteros.

Aunque el número de muestras no sea el suficiente para poder llegar a una conclusión sólida, si que permite observar a simple vista que ninguno de los algoritmos presentes en el gráfico anterior muestran un buen comportamiento general, si no que funcionan bien en casos específicos.

A pesar de no reducir la tasa de error respecto a Pollard rho, si que se puede observar una tendencia descendente de ésta a medida que aumenta del número de hilos. Debido a esta tendencia, podría esperarse que a medida que se aumentasen el número de hilos (16, 32 etc.) la tasa continuase bajando, llegando incluso a mejorar los resultados obtenidos por Pollard rho.

CAPÍTULO 6
Conclusión

6.1 Conclusiones

El desarrollo de un método para criptoanalizar de forma paralela RSA, si se habla estrictamente de la implementación, no ha supuesto una gran complejidad. Sin embargo, todo el trabajo previo a ésta si que ha sido un proceso costoso,ya que se ha realizado un recorrido por todas las tareas necesarias para la correcta implementación de un algoritmo. Al inicio de este proyecto se marcaron una serie de objetivos principales, los cuales han sido cumplidos a su finalización.

La primera tarea a realizar fue el estudio y análisis de las tecnologías de factorización vigentes en la actualidad, así como explicar de forma breve sus ventajas e inconvenientes. Esta parte del trabajo ha resultado de gran utilidad para aumentar los conocimientos generales sobre la materia, además de entender con precisión la dificultad que requiere factorizar un número de gran tamaño, complementando así las competencias aprendidas a lo largo del grado.

Posteriormente se decidió el algoritmo que mejor se adaptaba a las necesidades del proyecto. El proceso de la implementación del algoritmo ha seguido un proceso durante el cual se han utilizado distintos conocimientos adquiridos a lo largo de la estancia en la universidad. En primer lugar,a la hora de determinar que algoritmo se adaptaba mejor al proyecto, y cual sería su funcionamiento, se ha basado la elección utilizando métodos analíticos vistos en distintas asignaturas de software. La parte estrictamente relacionada con el código, ha sido facilitada por los conocimientos de programación aprendidos a lo largo de los años de estudio.

Finalmente, se ha realizado un estudio sobre el algoritmo, en el cual se ha comparado con otro ya consolidado en la actualidad. El hecho de evaluar su rendimiento utilizando como base un método conocido, ayuda a dar una visión más realista sobre el trabajo realizado, pudiendo concluir si la implementación realizada ha sido buena.

6.2 Trabajos futuros

Pese a que el algoritmo se desenvuelve bastante bien, se podrían realizar una serie de mejoras tanto a nivel de interfaz como a nivel más funcional, convirtiendo de esta forma un algoritmo utilizado para el estudio en una aplicación software más completa.

El punto más importante a mejorar sería el extender la experimentación, de cara a analizar mejor el número de hilos adecuado para cada ejecución, así como ir aumentando el tamaño de bits máximo que el algoritmo puede factorizar. Además, sería interesante su comparación con más algoritmos de los expuestos en el estado del arte, para mejorar la evaluación de su comportamiento.

Respecto a la interfaz, se podría realizar el diseño e implementación de una interfaz para poder trabajar con el algoritmo. Esto facilitaría el uso a la gente que muestra interés en el área, pero no posee ni los conocimientos ni la experiencia necesaria para trabajar directamente con código o por terminal.

Por otro lado, y entrando ya en el apartado más funcional, estaría bien que el sistema ofreciese la posibilidad de generar gráficas desde la propia aplicación, para facilitar el tratamiento de los datos, ya que actualmente solo guarda los resultados manualmente en un archivo de texto, y es el propio usuario quién debe montar los gráficos manualmente.

Finalmente, otra posible tarea consiste en la adaptación de la aplicación de cara a incluir más algoritmos para permitir a los usuarios comparar diferentes métodos de factorización. Además sería de gran interés académico que los propios usuarios pudieran integrar sus propios diseños de algoritmos para poder realizar estudios de rendimiento de una forma cómoda y eficiente.

Bibliografía

[1] Mohd Zaid Waqiyuddin Mohd Zulkifli. Evolution of Cryptography. *Evolution of Cryptography*, January, 2007.

[2] Prof.Waghmare S.P, Simran Sikhwal, Shreyas Nimje, Tanvi Pawar Bharati Vidyapeeth College Of Engg, Kharghar, Navi Mumbai (India) History of Cryptography. *International Journal For Technological Research In Engineering Volume 4, Issue 8*, April, 2017.

[3] Connelly Barnes. Integer Factorization Algorithms. *Department of Physics, Oregon State University*, December, 2004.

[4] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Ann. Math. 126, 649-673*, 1987.

[5] J. M. Pollard. Theorems in factorization and primality testing. *Proc. Cambridge Philos. Soc. 76 (1974), 521-528*.

[6] Yazhen Wang. Quantum Computation and Quantum Information. *Statistical Science, Vol. 27, No. 3, 373-394*, 2012.

[7] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *Computer Society Press*, 1996.

[8] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory IT-22, (Nov. 1976), 644-654*.

[9] A. Menezes, P. van Oorschot and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[10] Hans Riesel. *Prime Numbers and Computer Methods for Factorization*. Second Edition, 1994.

[11] R. Crandall, and C.Pomerance. *Prime Numbers. A Computational Perspective*. Springer, 2005.

[12] David Bressoud. *Factorization and Primality Testing*. Springer-Verlag, New York, 1989.

[13] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.