ALGORITMOS DE ENCRIPTACIÓN: ANÁLISIS DEL PROBLEMA DE LA FACTORIZACIÓN PRIMA EN EL MÉTODO RSA DE CLAVE PÚBLICA- ALGORITMO DE SHOR

ING. JUAN CARLOS MONTAÑO MACHACÓN

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA (UNAD)
ESCUELA DE CIENCIAS BÁSICAS, TÉCNOLOGIA E INGENIERIA
ESPECIALIZACIÓN EN SEGURIDAD INFORMÁTICA
CARTAGENA DE INDIAS D.T. y C.
2015

ALGORITMOS DE ENCRIPTACIÓN: ANÁLISIS DEL PROBLEMA DE LA FACTORIZACIÓN PRIMA EN EL MÉTODO RSA DE CLAVE PÚBLICA-ALGORITMO DE SHOR

ING. JUAN CARLOS MONTAÑO MACHACÓN

Documento monográfico presentado como requisito para optar al título de:

Especialista en Seguridad Informática

Director y Asesor:

Ing. y M.Sc. DEIVIS EDUARD RAMÍREZ MARTÍNEZ

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA (UNAD)
ESCUELA DE CIENCIAS BÁSICAS, TÉCNOLOGIA E INGENIERIA
ESPECIALIZACIÓN EN SEGURIDAD INFORMÁTICA
CARTAGENA DE INDIAS D.T. y C.
2015

Nota de aceptación:

_____

_____

_____

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma del jurado

_____

Firma del jurado

Cartagena de Indias D.T. y C., 05 de Junio de 2015

Dedicado:

A mis padres, maestros, y amigos que desde un principio me han brindado su apoyo incondicional e inspiración no solo para la realización del presente documento sino de cada uno de los objetivos de mi vida.

AGRADECIMIENTOS

Para la realización del presente documento me permito expresar mis más sinceros agradecimientos por su conocimiento aportado, esfuerzo, y dedicación, a mi asesor de tesis, ingeniero y M.Sc. Deivis Eduard Ramírez Martínez y que además gracias su paciencia y motivación me han llevado a culminar satisfactoriamente el presente documento.

RESUMEN

En el presente documento se desarrollará un análisis del algoritmo RSA de cifrado público y del algoritmo de factorización prima de Shor, planteando este último como una solución al problema de la factorización de enteros largos. El análisis se realizará desde el punto de vista de la complejidad asintótica que acarrea cada uno de estos algoritmos, comparando a su vez la eficiencia computacional ante métodos de factorización clásicos como lo es la criba general del cuerpo de números. En este sentido, se podrá apreciar la eficiencia en el tiempo de ejecución del algoritmo de Shor al obtener un tiempo polinomico de $O((\log n)^3)$ para resolver el problema de la descomposición prima para una entrada n, en comparación a la criba general del cuerpo de números cuya complejidad asintótica está dada heurísticamente por $O(exp^{[((64/9)^{1/3}+o(1)).(\log n)^{1/3}.(\log\log n)^{2/3}]})$ considerándose este último como el método clásico más eficiente en resolver el problema de la factorización contemplado en los sistemas de cifrado como el RSA.

ABSTRACT

In this paper an analysis of RSA public encryption algorithm and prime factorization algorithm Shor be developed, considering the latter as a solution to the problem of factoring large integers. The analysis is done from the point of view of asymptotic complexity that carries each of these algorithms, comparing computational efficiency turn to classical factorization methods such as the number field sieve. In this sense, they will appreciate the efficient execution time Shor algorithm to obtain a polynomial time. In this sense, they will appreciate the efficient execution time Shor algorithm to obtain a polynomial time $O((\log n)^3)$ to solve the problem of raw decomposition for an input n, compared to general body sieve numbers whose asymptotic complexity is heuristically given by $O(exp^{[((64/9)^{1/3}+o(1)).(\log n)^{1/3}.(\log\log n)^{2/3}]})$ considering the latter as the most efficient in solving the problem of factoring classic approach set out in ciphers like RSA.

CONTENIDO

1. INTRODUCCIÓN 1

2. DEFINICIÓN DEL PROBLEMA 3

2.1 DESCRIPCIÓN DEL PROBLEMA 3

2.2 FORMULACIÓN DEL PROBLEMA…….5

2.3 ALCANCE……..…….5

2.4 DELIMITACIÓN ……....5

3. JUSTIFICACIÓN……..…….……6

4. OBJETIVOS……...…….……7

4.1 OBJETIVO GENERAL……...……...…….……..…7

4.1 OBJETIVO ESPECÍFICO……...…….……...……..7

5. MARCO REFERENCIAL…….……8

5.1 ANTECEDENTES……..……...……..…...……..…8

5.2 MARCO TEÓRICO…..……..……...……..…...……..…9

5.3 MARCO CONCEPTUAL.…..……...…….…...…..…..12

5.4 MARCO LEGAL……...…..……...……..…...…..…..16

6. DISEÑO METODOLÓGICO…….……..…17

6.1 TIPO DE INVESTIGACIÓN.…..……...…….……..…….....17

6.2 METODO DE INVESTIGACIÓN.…..……...…….…...……....18

6.3 POBLACION Y MUESTRA.…..……...…….…...…….....18

6.4 FUENTES DE INFORMACIÓN……...…….…....…….....19

7. DISIPOSICIONES PRELIMINARES…….……..…19

7.1 MÁQUINA DE TURING DETERMINISTA……..…….……..…....20
7.2 TIEMPOS DE EJECUCIÓN Y COMPLEJIDAD ALGORITMICA…….……...21
7.3 CLASES DE ALGORITMOS…….…….…....26
8. ANÁLISIS ……..…….…….…29
8.1 CONSIDERACIONES GENERALES…....29
8.2 CRIPTOSISTEMA DE LLAVE PÚBLICA RSA.....31
8.2.1 Funcionamiento y aspectos centrales…….....32
8.2.2 Estructuración del algoritmo y fundamentos…….....35
8.2.3 Aspectos sobre la complejidad del RSA …….....40
8.2.4 Métodos clásicos de factorización…….....42
8.2.5 Ejemplo proceso de cifrado RSA…….....49
8.3 EL ALGORITMO DE SHOR COMO SOLUCION A LA FACTORIZACIÓN.....52
8.3.1 Algunos aspectos fundamentales…….....54
8.3.2 El algoritmo de Shor como solución al problema …….....62
8.3.3 Observaciones generales y complejidad algoritmica …….....66
9. CONCLUSIÓNES…….…….…….…….……..…73
10. CRONOGRAMA …….…….…….……....75
11. PRESUPUESTO Y RECURSOS …….…….…….....76
11.1 RECURSO HUMANO……..…...76
11.2 PRESUPUESTO …….……..…...76
12. REFERENCIAS BIBLIOGRAFÍCAS.…….…….…….……....77

1. INTRODUCCIÓN

Dentro del contexto actual, hemos sido testigos del papel fundamental que los sistemas de información han desempeñado en el desarrollo tanto industrial, económico, y social, formado así un activo esencial para cualquier organización. En este sentido existe una cantidad diversa de elementos que integran los activos de una organización concreta, estos pueden diferirse dependiendo de la naturaleza de ellas. Sin embargo e independientemente de la razón social o comercial, todas las organizaciones guardan un activo en común, siendo además el más preciado: la información. Bajo este enfoque, la protección de los datos resulta de vital importancia para el funcionamiento y la sostenibilidad de cualquier empresa. Para esta tarea se disponen de un gran conjunto de herramientas dirigidas a proteger uno de los aspectos más fundamentales y asociados al manejo de la información: la confidencialidad (además de ello existen otros parámetros importantes en el proceso para la gestión de los datos tales como la integridad, disponibilidad, o la trazabilidad, para los cuales a su vez existen también una variedad de herramientas y técnicas enfocadas a ellos y que son de interés ajeno a la criptografía). En efecto, es común encontrar dentro de este conjunto, herramientas tales como: hardware o equipos especializados, software, políticas, planes de acción, aplicaciones, etc. Sin embargo, resulta claro que la mayor parte de la solución al problema de la confidencialidad, no la ofrecen las políticas, ni el hardware o el sistema de gestión en sí, sino la solides de las técnicas por los cuales dicha información es cifrada. En conformidad a esto, y partiendo de la premisa que no existe ningún sistema completamente seguro, un atacante podría tener acceso a los sistemas de información (S.I.) que contengan toda la información relevante de una organización, sin embargo, si esta se encontrase cifrada sería una medida más que el atacante tendría que superar para poder acceder a la información útil.

En este aspecto, el tema de estudio estará centrado en el análisis de uno de los métodos de cifrado más implementados en los sistemas actuales, específicamente el método clásico

RSA[1] de cifrado asimétrico (siendo este considerado un algoritmo fuerte ante ataques clásicos por descomposición prima), contrastándolo a su vez ante un algoritmo de factorización más moderno como lo es el algoritmo de Shor. Se implementará las funciones de tiempo de ejecución y complejidad (Función O de Landau de cota superior asintótica), además de algunas disposiciones de la teoría de números y otros principios para demostrar la eficiencia del algoritmo de Shor ante el sistema de cifrado RSA.

[1] En criptografía, RSA *(Rivest, Shamir y Adleman)* es un sistema criptográfico de clave pública desarrollado en 1977. Es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente.

2. DEFINICIÓN DEL PROBLEMA

2.1. DESCRIPCIÓN DEL PROBLEMA

Muchos métodos de cifrado clásicos, como el RSA basan su seguridad en la dificultad que representa la factorización prima de grandes números (aquellos que pueden ser representados como producto de potencias de los llamados *primos fuertes*), esto es si n es un número entero positivo cualquiera, entonces por el *teorema fundamental de la aritmética*, tenemos que dicho número puede ser representado exactamente y con unicidad como un producto de potencias de números primos, esto es:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k} = \prod_{i=1}^{k} p_i^{\alpha_i}$$

donde $p_1 < p_2 < \dots < p_k$ son primos y α_i son enteros positivos. En efecto, cuando el valor de n (con n$\in Z^+$) es muy grande y cuya descomposición factorial está dada por dos primos de aproximadamente el mismo tamaño (con valores correspondientes de n superiores a 700 bits o superiores a 200 dígitos decimales), no existe dentro del marco clásico un algoritmo que resuelva eficientemente el problema de su factorización, es decir, no hay actualmente un algoritmo que permita describir la complejidad relacionada al tiempo de ejecución en función de un tiempo polinomial (esto es O(b^k) para alguna constante k, siendo b el número de bits correspondientes a n en representación binaria), en este sentido y dentro del marco clásico el mejor tiempo asintótico de ejecución podemos encontrarlo en el Algoritmo CGCN (Criba General del Cuerpo de Números), a través de la forma canónica para O dada por:

$$O\left(exp^{\left(\left(\frac{64}{9}b\right)^{\frac{1}{3}}.(\log b)^{\frac{2}{3}}\right)}\right)$$ o bien:

$$exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln n)^{1/3}.(\ln\ln n)^{2/3}\right)=L_n=\left[\frac{1}{3},\sqrt[3]{\frac{64}{9}}\right]$$

para un $n > 0$ siendo b el número de bits correspondientes a n en notación binaria (esto es $b = log_2 n$). En efecto y cómo se puede observar, dicho tiempo asintótico de ejecución dado por la primera ecuación no es polinomico, de hecho esta imposibilidad técnica es la que da soporte solido a los sistemas de cifrado clásicos como el RSA, al asignar al cifrado de las claves publicas pares de números primos relativamente grandes.

Por otro lado y en contraste a lo mencionado, aunque los algoritmos para la factorización de números grandes son ineficientes al correr en sistemas de cómputos ordinarios, este problema se soluciona fácilmente cuando la ejecución se efectúa en un computador cuántico a través del algoritmo de Shor, el cual a diferencia del RSA, puede romper el cifrado por descomposición prima en un tiempo polinómico notablemente pequeño, siendo específicos el tiempo de ejecución, se realizaría en solo $O((log\ n)^3)$, n>0, lo cual es una diferencia notable en contraste a la ecuación mostrada al principio. En este sentido, se pretende realizar, un análisis más profundo de ambos casos, demostrando no solo los atributos del algoritmo de Shor como técnica para romper el RSA, sino también que las técnicas de cifrado actuales basadas en descomposición de primos fuertes llegarían a ser obsoletas.

2.2. FORMULACIÓN DEL PROBLEMA

¿Cómo y mediante qué mecanismos es posible desarrollar un análisis completo de la eficiencia del algoritmo de RSA de clave pública en contraste con el algoritmo de factorización prima Shor, que permita a su vez establecer una noción más amplia de la seguridad en los sistemas de cifrados?

2.3. ALCANCE

La presente monografía ha sido desarrollada con el propósito de lograr un análisis del problema de la factorización prima implicado en el sistema de cifrado RSA lo cual se realizara mediante un enfoque tanto descriptivo como explicativo del problema. En este orden de ideas se abordaran algunos temas previos tales como orden de complejidad, definición de algoritmo, algoritmo de cifrado RSA, definición de qbit, algoritmo de Shor, transformada de Fourier, entre otros aspectos de vital importancia que permitan el desarrollo descriptivo del tema central de estudio, y que conlleven a establecer la ventaja del algoritmo de Shor como una solución óptima al problema de la factorización prima desde el punto de vista computacional.

2.4. DELIMITACIÓN

Aunque existen una gran variedad de programas para el estudio o implementación de los algoritmos clásicos como el RSA, es importante resaltar que el algoritmo de Shor requiere de equipos de cómputo no convencionales para su ejecución, y que a pesar de haberse desarrollado algunos de este tipo, esta es un área que recién se encuentra en pleno desarrollo, por lo cual este tipo de hardware se encontraría a poco menos de un par de décadas en estar al alcance de todo público. Por lo tanto el desarrollo del análisis que se plantea en el presente documento será netamente teórico.

3. JUSTIFICACIÓN

A lo largo de la última década se ha visto como la necesidad de resguardar cierta información es evidente a la hora de considerar sistemas de procesamiento de datos personales, transacciones económicas, o bien ante el riesgo que requieran verificación de identidad. En este aspecto, la criptografía desempeña un papel clave en la protección de los datos, seguidamente, y bajo el contexto actual muchas organizaciones tanto gubernamentales como privadas están yendo un paso más allá de los métodos de cifrado y descifrado tradicionales, esto respondiendo a los requerimientos de eficiencia en los tiempos de ejecución y los avances de la tecnología.

Teniendo en cuenta lo anterior, la importancia de la elaboración de la presente monografía radica en mostrar la obsolencia de ciertos algoritmos clásicos muy comunes ante el posible ataque mediante otros algoritmos mucho más modernos, proporcionando de esta manera un soporte fundamental tanto para un entendimiento de los métodos de cifrado, como para el desarrollo de sistemas de información más seguros.

4. OBJETIVOS

4.1. OBJETIVO GENERAL

Analizar los aspectos más fundamentales en los cuales los algoritmos de cifrado clásicos como el RSA soportan la seguridad de su cifrado, estableciendo la solides del mismo al ser contrastados por ataques basados en algoritmos modernos de factorización prima para claves públicas como lo es el algoritmo de Shor.

4.2. OBJETIVOS ESPECÍFICOS

- ✓ Mostrar la eficiencia de los métodos de factorización más modernos como técnica para vulnerar algoritmos clásicos como el RSA, a través de una reducción del orden de complejidad en tiempo polinomial.

- ✓ Detallar y sintetizar el componente teórico que integra los algoritmos criptográficos de Shor y RSA.

5. MARCO REFERENCIAL

5.1. ANTECEDENTES

Dentro el campo de las ciencias básicas, tecnología e ingeniería se evidencia diversidad de fundamentos teóricos y prácticos que permiten establecer un conjunto de métodos que enfocados a asegurar en un alto grado la confidencialidad de los datos. Estas herramientas (en su mayoría matemáticas y tecnológicas), permiten en particular crear mecanismos que proporcionan un cifrado cada vez más seguro. En este sentido, se han elaborado una gran variedad de algoritmos de encriptación clásicos como lo son RSA, DSA, los de Curvas Elípticas, todos estos muy usados por protocolos como el SSL, el SSH, TLS entre otros; además de una serie de algoritmos de cifrado modernos como el algoritmo de Grover, o el método de cifrado de Deutsch-Jozsa. En este aspecto, se han llevado una gran variedad de estudios, artículos y análisis de estos mismos desde el punto de vista tanto introductorio como científico.

En 1977, Ron Rivest, Adi Shamir y Leonard Adleman, del Instituto Tecnológico de Massachusetts (MIT) describieron el algoritmo de encriptación clásico RSA; sin embargo Clifford Cocks, un matemático británico, había descrito un sistema equivalente en un documento interno en 1973. Debido al elevado coste de las computadoras necesarias para implementarlo en la época su idea no trascendió. El algoritmo fue patentado por el MIT en 1983 en Estados Unidos con el número 4.405.829.

Por su parte en 1995 Peter W. Shor del AT&T Research púbico en un documento titulado Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, el algoritmo de descifrado que se le conoce hoy por su nombre, tuvo por objeto dar una solución frente algunos problemas inmersos en los mismos métodos de cifrado de los algoritmos clásicos de factorización prima. Bajo este orden de ideas, el

algoritmo consistía en aumentar la eficiencia del mismo reduciendo a un tiempo polinomico la decodificación de una clave pública basada en el principio de factorización prima clásico. Por otro lado, se han publicado una serie de artículos científicos, en especial algunos pertenecientes de la rama de la matemática aplicada y las ciencias de la computación, en los cuales se hace un esfuerzo por introducir al lector a los fundamentos teóricos y prácticos de estos algoritmos. Uno de estos documentos en los cuales se describe de manera bastante técnica, y que se tendrá muy en cuenta a la hora de realizar este proyecto, fue publicado durante el 2005 por Rigolin Gustavo y Rieznik Aníbal A, titulado Introduction to quantum cryptography, en este se presenta de manera detallada uno de los muchos análisis que se han elaborado para el entendimiento de los algoritmos cuánticos.

Por último, podemos contar con el soporte teórico de otros artículos que naturalmente se tendrán muy encuentra para la elaboración de este proyecto, uno de estos documentos data de 2006 el cual fue elaborado por el David Mermin del departamento de física de Cornell University cuyo título es Breaking RSA Encryptation with a Quantum Computer: Shor's Factoring Algorithm. En este se describe detalladamente el funcionamiento de los dos algoritmos (RSA y de Shor) dando además las pautas por las cuales se precisan las ventajas del algoritmo cuanto frente al clásico.

5.2. MARCO TÉORICO

Dentro de la a investigación se emplean teorías relacionadas al campo de la computación clásica junto con algunas herramientas matemáticas como la Teoría de Módulos, Teoría de Anillos y Campos, Teoría de Números, Métodos Criptográficos, y Principios de la Computación Cuántica. En primer lugar; Peter Shor expone un marco para sintetizar de manera más simple el problema de la factorización prima, reduciendo así el tiempo de ejecución a un tiempo polinomico, esto es llevando el operador de cotas superior asintótica dado por:

$$O\big(g(x)\big) = \left\{ \begin{matrix} f(x) : existen\ c, x_0 > 0\ tales\ que \\ \forall_{x \geq x_0} : 0 \leq |f(x)| \leq c|g(x)| \end{matrix} \right\}$$

A un operador de la forma $O(b^k)$, donde O es el espacio de todas las funciones acotadas superiormente por $g(x)$. En este aspecto, los sistemas clásicos conducen a una ineficiencia en los tiempos de ejecución de los algoritmos de decodificación para primos grandes, puesto que los métodos algorítmicos comunes conducen a un operador de cota superior asintótica no polinomico, radicando en este problema la seguridad del cifrado, en este caso los tiempos de ejecución para el mejor de los métodos clásicos corresponde al de la Criba General del Cuerpo de Números (CGCN), cuyo tiempo corresponde a un operador de la forma:

$$O\left(exp\left(\left(\frac{64}{9} b \right)^{\frac{1}{3}} . (\log b)^{\frac{2}{3}} \right) \right)$$

Siendo este último no polinomico! dando lugar a un tiempo de ejecución notablemente grande.

En este sentido, Shor plantea una salida, partiendo de una asociación de los Espacios de Hilbert con un sistema cuántico a través de vectores complejos de 2^n estados como base vectorial, seguidamente como al multiplicar estos vectores por un vector normal complejo, estos no varían, el comportamiento de estos vectores no cambia al multiplicarse por una fase compleja de longitud unitaria, entonces eso significa que necesitaríamos $2^n - 1$ números complejos para describir completamente el estado. Este sentido se representa teóricamente las superposiciones de estos estados mediante la combinación lineal de los vectores de la base, dada por:

$$\sum_{i=0}^{2^n-1} a_i |S_i\rangle$$

Donde la amplitud por cada uno de los a_i es un número complejo tal que $\sum_i |a_i|^2$, en este sentido, desde que se consideren este grupo de transformaciones unitarias, la herramienta que nos permitirá centrar un algoritmo de ejecución más eficiente, se basa en el uso de la transformada de Furier conforme los espacios vectoriales elegidos y su combinación lineal presentada anteriormente en los Espacios Hilbert, dada por:

$$\frac{1}{q^{1/2}} \sum_{c=0}^{q-1} |c\rangle \, exp(2\pi iac/q)$$

Esto es tomando $0 \leq a < q$ para algún q cuyo número de bit sea ejecutado en un tiempo polinomial. En este caso se implementará una matriz cuya entrada es

$$\frac{1}{q^{1/2}} exp(2\pi iac/q)$$

Para llevar el sistema aun estado

$$\frac{1}{q} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} exp(2\pi iac/q) |c\rangle |x^a (mod\ n)\rangle$$

De aquí, el algoritmo de Shor pasa a obtener una condición de desigualdad dada por:

$$\left| \frac{c}{q} - \frac{d}{r} \right| \leq \frac{1}{2q}$$

Este aspectos será el que se aprovechará, y que además será nuestro punto de partida para analizar los tiempos de eficiencia del algoritmo de Shor y realizar así un contraste con el método clásico de cifrado RSA en conformidad con otros algoritmos de factorización como el método de la criba del cuerpo de números.

5.3. MARCO CONCEPTUAL

Algoritmo de Shor: Es un algoritmo cuántico probabilístico capaz de descomponer en factores primos un número *N* en tiempo O((log N)^3) y espacio O(log*N*).

Base ortonormal: Una base de un espacio vectorial es ortonormal cuando es una base ortogonal y sus vectores son unitarios, es decir: Si V es un espacio vectorial dotado de un producto interno $\langle\ ,\ \rangle$ entonces $B = \{e_1, e_2, \dots, e_n\}$ es una base ortonormal, si y solo si todo vector $x \in V$ puede ser generado mediante una única combinación lineal de los elementos de B, siendo ademas $\langle e_k\ ,\ e_j\rangle = 0$ para todo $k \neq j$, $\langle e_k\ ,\ e_k\rangle = 1$.

Complejidad algorítmica: En ciencias de la computación, se define como aquella medida de la cantidad de recursos en tiempo de ejecución, memoria y procesamiento que requiere un algoritmo en ejecución para desempeñar una tarea de cómputo específica.

Cota superior asintótica: En análisis de algoritmos una cota superior asintótica es una función que sirve de cota superior de otra función cuando el argumento tiende a infinito. Usualmente se utiliza la notación de Landau O*(g(x))* (o coloquialmente llamada Notación O Grande) para referirse a las funciones acotadas superiormente por la función *g(x)*.
Más formalmente se define:

$$O\big(g(x)\big) = \left\{ \begin{matrix} f(x) : existen\ c, x_0 > 0\ tales\ que \\ \forall_{x \geq x_0} : 0 \leq |f(x)| \leq c|g(x)| \end{matrix} \right\}$$

Una función *f*(*x*) pertenece a O(*g*(*x*)) cuando existe una constante positiva *c* tal que a partir de un valor x_0, *f*(*x*) no sobrepasa a $cg(x)$. Quiere decir que la función *f* es inferior a g a partir de un valor dado salvo por un factor constante.

Criptoanálisis: Conjunto de técnicas implementadas para romper los esquemas de protección por cifrado aplicados para el ocultamiento de información relevante.

Criptografía: Conjunto de técnicas enfocadas a la protección de la información a través del ocultamiento (o cifrado) de la misma ante observadores no autorizados, englobando áreas diversas de la ciencia como teoría de la información, la teoría de números (o matemática discreta, que estudia las propiedades de los números enteros), y la complejidad algorítmica.

Criptografía simétrica: La criptografía simétrica, también llamada criptografía de clave secreta o criptografía de una clave, es un método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la clave a usar. Una vez que ambas partes tienen acceso a esta clave, el remitente cifra un mensaje usando la clave, lo envía al destinatario, y éste lo descifra con la misma clave.

Criptosistema: Una cifra o criptosistema es un método secreto de escritura, mediante el cual un texto en claro se transforma en un texto cifrado o criptograma. El proceso de transformar un texto en claro en texto cifrado se denomina cifrado, y el proceso inverso, es decir la transformación del texto cifrado en texto en claro, se denomina descifrado. Ambos procesos son controlados por una o más claves criptográficas.

Dual (Espacio): Dado cualquier espacio vectorial V sobre un cierto cuerpo F, definimos el espacio dual V* como el conjunto de todos las funcionales lineales en F, es decir, transformaciones lineales en V a valores escalares.

Factorización prima: Propiedad de los números de poder ser representado exactamente de una única manera como un producto de potencias de números primos, esto es:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k} = \prod_{i=1}^{k} p_i^{\alpha_i}$$

donde $p_1 < p_2 < \dots < p_k$ son primos y α_i son enteros positivos.

Fracción continúa: Una fracción continua es una expresión matemática que permite aproximar un número real x a través de números racionales, dicha expresión está dada por la forma:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_{n-1} + \cfrac{1}{a_n}}}}}$$

Donde a_0 es un entero cualquiera, mientras que para $i = 1,2,..,n$ los a_i son enteros positivos. Se acostumbra a denotar la expresión anterior como $[a_0; a_1, a_2, \ldots a_n]$.

Función de Euler: Se define como el número de enteros positivos menores o iguales a n y que son primos relativos con n. Es decir $\phi(n) = card\,\{m \in \mathbb{N} : m \leq n \ \wedge mcd(n,m) = 1\}$

Paralelismo cuántico: Propiedad física intrínseca de las partículas atómicas que permiten realizan varias operaciones simultáneamente y sobre todas las combinaciones de una cierta entrada.

Periodo (de una función): Se define el periodo de una función f al elemento r que permite que los valores de la función se repitan conforme este se añada a la variable independiente, es decir, es el r tal que permite que f satisfaga la siguiente expresión $f(x + r) = f(x)$ para todo x del dominio.

Principio de superposición cuántica: Es el principio matemático por el cual nos permite representar los estados de una partícula o un sistema mediante una combinación lineal de uno vectores o estados bases.

Producto interno (escalar): En un espacio vectorial V, y un campo escalar K, un producto interno se defino como una aplicación $\langle\ ,\ \rangle : \mathrm{V} \times \mathrm{V} \to \mathrm{K}$ que cumple con las siguientes condiciones:

- $\langle\, ax + by\,,\ z\,\rangle = a\langle\, x\,,\ z\rangle + b\langle\, y\,,\ z\rangle$ y $\langle\, x, ay + bz\,\rangle = \bar{a}\langle\, x\,,\ y\rangle + \bar{b}\langle\, x\,,\ z\rangle$
- $\langle\, x\,,\ y\rangle = \overline{\langle\, y\,,\ x\rangle}$
- $\langle\, x\,,\ x\rangle \geq 0$ y $\langle\, x\,,\ x\rangle = 0$ si y solo si $x = 0$

Siendo $x, y \in \mathrm{V}$ y $a, b \in \mathrm{K}$, y donde el elemento $\bar{a}, \bar{b}$ representan los conjugados de a y b respectivamente.

Producto exterior: Una matriz $m \times n$ obtenida como producto de una matriz $m \times 1$ (un vector ket $|\varphi\rangle$ y una matriz $1 \times n$ (vector bra $\langle\varphi|$), es llamado producto exterior.

Puerta de Hadamard: Es definida como una puerta lógica que opera sobre un único qbit, y que es representada mediante la matriz $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, la cual aplica una rotación de π sobre los ejes coordenados x y z.

RSA: En criptografía, **RSA** *(Rivest, Shamir y Adleman)* es un sistema criptográfico de clave pública desarrollado en 1977. Es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente. La seguridad de este algoritmo radica en el problema de la factorización de números enteros. Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto. Actualmente estos primos son del orden de 10^{200}, y se prevé que su tamaño crezca con el aumento de la capacidad de cálculo de los ordenadores.

Transformada de Hadamard: La transformada de Hadamard es un caso particular de la transformada discreta de Fourier. La transformada sobre un qbit, se define como la transformada H que toma un registro de entrada $|x\rangle$ y es aplicado mediante la expresión

$$\boldsymbol{H}|x\rangle = \frac{1}{\sqrt{2}}\sum_{y=0,1}(-1)^{x.y}|y\rangle$$

Transformada de Fourier discreta: Sea $x_0, x_1, \dots, x_{N-1}$ una secuencia de N números complejos, se define la transformada de Fourier discreta como aquella función que transforma cada elemento x_k de la secuencia anteriormente dada en el elemento X_k, dado por $X_k = \sum_{n=0}^{N-1} x_n\, e^{-\left(\frac{2\pi i}{N}\right)kn}$ con $k = 0,1,..N-1$. Donde i es la unidad imaginaria, y $e^{\left(\frac{2\pi i}{N}\right)}$ es la N-esima raíz de unidad.

Vector adjunto (matriz $n \times 1$ adjunta): En general se denomina matriz adjunta (denotada comúnmente por A^*) o hermítica, o conjugado hermítico de una matriz A, a toda aquella matriz obtenida mediante la transpuesta y posteriormente tomar su conjugada compleja. Es decir sea $A = (a_{ij})$ una matriz con elementos $a_{ij} \in \mathbb{C}$, entonces $A^* = (\bar{a}_{ji})$.

5.4. MARCO LEGAL

Como podemos evidenciar, nuestro tema de estudio es en su mayor parte de naturaleza teórica, no obstante se dispondrá a tratar con algoritmos, técnicas, y algunas herramientas criptografías propias que nos permitirán un desarrollo satisfactorio de nuestro objetivo general, en este sentido estos son temas que de una u otra forma desembocan en el campo de la seguridad informática y por lo tanto, es menester adoptar la normatividad Colombiana que regula estos temas en cuanto a la implementación de software y la protección de datos.

Bajo este escenario, y en pro de respetar los derechos de autor de las fuentes consultadas, la fundamentación legal para lo que respecta a la realización de este trabajo, se centrará bajo la siguiente normatividad:

- ✓ Ley 23 de 1982, Articulo 1, 2, 3 del Capítulo I y demás cotejados en esta ley a los que tenga lugar.

- ✓ Ley 1273 de 2009, Articulo 1 Capítulo I Artículo 269E: *Uso de software malicioso*. El que, sin estar facultado para ello, produzca, trafique, adquiera, distribuya, venda, envíe, introduzca o extraiga del territorio nacional software malicioso u otros programas de computación de efectos dañinos, incurrirá en pena de prisión de cuarenta y ocho (48) a noventa y seis (96) meses y en multa de 100 a 1.000 salarios mínimos legales mensuales vigentes.

6. DISEÑO METODOLÓGICO

6.1. TIPO DE INVESTIGACIÓN

La investigación que se realizará en este trabajo será de tipo descriptiva y bajo un enfoque explicativo, teniendo como objetivo prioritario detallar el conjunto de procedimientos técnicos y metodológicos que conllevan a determinar cuándo un algoritmo de cifrado puede considerarse seguro, confrontando el punto de vista tanto clásico como moderno. La investigación de tipo descriptiva englobará el caso de estudio de tipo dentro de lo ilustrativo, para luego emplear el análisis como herramienta deductiva y que permitirá determinar el objetivo planteado en el documento.

Bajo esta condición, se analizará los métodos y procedimientos teóricos desde el punto de vista clásico como moderno, y que conllevan a determinar la ventaja en la eficiencia

computacional del algoritmo de factorización de Shor para el descifrado de claves tratadas en el método RSA de llave pública.

6.2. MÉTODO DE INVESTIGACIÓN

En la presente investigación se empleará el método deductivo y el método analítico. Primeramente, el método se encargará de establecer un conjunto de premisas generales para llegar a conclusiones particulares, es decir, va de lo general a lo particular. En segunda y último instancia, se implementará el método analítico en todo su cuerpo (esto es incluyendo conceptos, definiciones, y demás aspectos substanciales) para llegar a obtener conclusiones que nos permitan determinar la ventajas de los algoritmos de encriptación modernos como el de Shor frente a los clásicos para resolver el problema de la factorización prima en que se fundamenta el sistema de cifrado RSA de clave pública.

6.3. POBLACIÓN Y MUESTRA

El desarrollo del presente trabajo se centrara en el análisis de la complejidad computacional de dos algoritmos de naturaleza distinta, estos son: el algoritmo de cifrado RSA de clave pública y el algoritmo de factorización de Shor para enteros largos. En este sentido, cabe manifestar dos aspectos: en primer lugar el alcance de la investigación que se ha de realizar guarda un carácter explicativo y descriptivo, en segundo lugar el enfoque a tratar no es de carácter inferencial sino netamente teórico (esto debido a la naturaleza de la temática a tratar). Por lo tanto, y en virtud a que no existen datos que puedan manejarse o tomarse como población o muestra aleatorias para la realización de un estudio inferencial del mismo, se prescindirá de estos aspectos de estudios muéstrales o poblacionales.

6.4. FUENTES DE INFORMACIÓN

Se implementaran fuentes de consultas e información diversas naturalezas tales como libros (tanto digitales como físicos), artículos científicos elaborados por los mismos autores de los algoritmos descritos, internet, notas sobre cursos desarrollados de criptografía, algebra abstracta y algebra lineal, entre otros. Además de ello, se implementaran las sugerencias en cuanto a contenido temático y diseño del documento suministradas por el asesor de tesis, también se implementaran las recomendaciones para el desarrollo y la estructuración de documentos escritos suministradas por la norma técnica colombiana NTC 1486. En conformidad a ello, se procederá a presentar posteriormente en el cuerpo de la bibliografía todas las fuentes de información tenidas en cuenta tanto para la elaboración y el análisis de la temática como para la estructuración del presente documento.

7. DISPOSICIONES PRELIMINARES

En la actualidad y con el propósito que estos cumplan con su tarea, los sistemas de información, comúnmente son dotados con mecanismos que garanticen un alto nivel de seguridad y protección de toda la información que se transfiere en cada uno en los procesos, estos mecanismos pueden estar enfocados a distintos aspectos básicos como la confidencialidad, la integridad, disponibilidad o bien sea la trazabilidad. En este orden de ideas, los algoritmos de cifrado guardan un aspecto relevante en cuanto a la protección de los datos y desde el punto de vista de la confidencialidad, esto toda vez que se cumpla con algunos criterios básicos como la eficiencia, solides del cifrado en relación al tamaño de las entradas o claves a implementar, calidad del código generado por el compilador, entre otros. En conformidad a esto, es posible encontrar una gran cantidad de algoritmos que sustraen la solides de su cifrado a través de una diversidad de problemas numéricos de alta complejidad computacional, en general existe una amplia gama de algoritmos que implementan el problema de la descomposición prima de enteros largos como mecanismo

para soportar el cifrado de datos, o en su defecto emplean grandes primos en otro tipo de cómputo para soportarlo (como bien se hace al implementar el problema de logaritmos discretos). En efecto y para lo que nos concierne, el algoritmo RSA de clave pública es de hecho uno de las más notables y cuyo funcionamiento está ligado a la complejidad computacional que existe en determinar los factores primos de un entero largo. En este sentido y en virtud a la naturaleza diversa de los problemas en que estos algoritmos pueden ser estructurados, se abordará el caso de estudio de la presente monografía, iniciando con algunos conceptos esenciales para elaborar nuestro análisis y que permitirán establecer una clasificación de los algoritmos de cifrado según su naturaleza.

7.1. MAQUINA DE TURING DETERMINISTA

En esta sección se procederá introduciendo un concepto fundamental que nos llevará a establecer un estudio más formal de los tiempos de ejecución de un algoritmo, y de la eficiencia o complejidad de los mismos.

Definición (Función parcial o parcialmente computable): Sean A y B conjuntos no vacíos, se dice que una función $\delta : A \to B$, es parcial o parcialmente computable, si y solo si $(\exists C \subseteq A),\ \text{con}\ C \neq \emptyset,$ tal que $\forall x \in C,\ \exists y = \delta(x) \in B$. Es decir si existe un subconjunto C en el que todos sus elementos tienen imagen calculable a través de δ. Nótese que no todos los elementos de dominio tienen imagen calculable (tómese $x \in A \cap C'$).

Definición (Máquina de Turing determinista): Se define una máquina de Turing determinista (M o M.T.), como un modelo computacional compuesto por una 7-tupla, capaz de ejecutar los cálculos de una manera automática a través de un proceso de lectura/escritura sobre una cinta o entrada. Esto es $\mathcal{M} = (Q,\ \Sigma,\ \Gamma,\ q_0, b,\ F,\ \delta)$, siendo:

Q: un conjunto finito de estados.

Σ: el alfabeto de entrada.

Γ: es el alfabeto de la cinta, formado por un conjunto de símbolos, incluye a Σ, Σ⊆ Γ.

q_0: es el estado inicial ($q_0 \in$ Q)

b: es un símbolo denominado blanco, y es el único símbolo que se puede repetir un número infinito de veces, b ∈ Γ, b ∉ Σ. Por lo que en principio aparece en todas las casillas excepto en aquellas que contienen los símbolos de entrada.

F: es el conjunto de estados finales de aceptación. F ⊆ Q

δ: es una función parcial de transición tal que $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{I, D\}$. La parcialidad permitirá la existencia de elementos del dominio donde δ no está definida. Siendo I y D un movimiento a la izquierda y a la derecha respectivamente.

Algoritmo (Conceptualización): Nótese que una máquina de Turing $\mathcal{M}$ dada, cumple intrínsecamente con los siguientes aspectos: Puede asociarse a un conjunto finito de instrucciones ordenadas y ligadas a un conjunto de estados específicos, lo que puede resumirse a un procedimiento el cual llamaremos $\mathcal{A}$ con $\mathcal{A}$ (w) $\rightarrow (a_1, a_2, \ldots a_n)$ = f* siendo w un estado inicial, f* un estado final de aceptación y $a_i \subseteq$ Q con $1 \leq i \leq n$. Por tal motivo, parece natural asociar o más aun, definir intrínsecamente una máquina de Turing determinista al comportamiento de un algoritmo. *En este sentido se hablará indistintamente de una máquina de Turing ($\mathcal{M}$) o Algoritmo ($\mathcal{A}$) (tesis de Church).*

7.2. TIEMPOS DE EJECUCIÓN Y COMPLEJIDAD ALGORITMICA

Naturalmente una computadora entre más capacidad de procesamiento disponga, más rápido ejecutara un algoritmo, por lo cual se estaría propenso a pensar que el tiempo de

ejecución podía calcularse registrando el tiempo que le toma a un algoritmo resolver un problema en proporción a la velocidad de procesamiento de la máquina, sin embargo y aunque técnicamente es válido, se estaría dependiendo de la capacidad de computo de cada máquina. En este sentido, y para tratar de una manera más objetiva los tiempos de ejecución algorítmicos, se debe tener en cuenta que por ley general si dos algoritmos $\mathcal{A}_1$ y $\mathcal{A}_2$ con un número de pasos o instancias dadas por p_1 y p_2 respectivamente ($p_1 \ll p_2$) son diseñados para resolver un mismo problema $\mathcal{P}$, entonces independientemente de la capacidad de computo que tenga la maquina donde se implementen, tendremos que los tiempos de ejecución $t_{\mathcal{A}_1}(w) \leq t_{\mathcal{A}_2}(w)$, esto considerando una misma entrada $w \in Q$ de tamaño n (perteneciente al conjunto finito de estados) y teniendo en cuenta que cada paso o instancia a ejecutar en $\mathcal{A}_1$ y $\mathcal{A}_2$ es menos compleja que cada una de las instancias correspondientes de $\mathcal{A}_2$, esto es, si $\mathcal{A}_1$ (w) $\rightarrow$ $(a'_1, a'_2, \ldots a'_{p_1})$ y $\mathcal{A}_2$ (w) $\rightarrow$ $(a''_1, a''_2, \ldots a''_{p_2})$ entonces cada a''_i es menos complejo que cada a'_i respectivamente, con $1 \leq i \leq p_1$. Por lo tanto, resulta más ventajoso medir los tiempos de ejecución $t_{\mathcal{A}}(w)$ de un algoritmo, en función a el número de pasos ejecutados por $\mathcal{A}$ con entrada w. Para hacer esto se procederá a soportar dicha definición en correspondencia a lo siguiente:

Σ^*: Conjunto construido por todas las combinaciones posibles de entradas de Σ.

Paso de $\mathcal{A}$: Se define formalmente un paso de $\mathcal{A}$ como la ejecución de una instrucción proporcionada por su respectiva función de transición.

$t_{\mathcal{A}}(w)$: Se introduce formalmente el tiempo de ejecución de un algoritmo como el número de pasos ejecutados por $\mathcal{A}$ con entrada w $\in \Sigma^*$, esto es:

$$t_{\mathcal{A}} : \quad \Sigma^* \rightarrow \mathbb{R}^+$$
$$w \rightarrow t_{\mathcal{A}}(w)$$

Una vez aceptando estas correspondencias, se procederá construir una de las definiciones formales que trataremos a lo largo de nuestro estudio y que se conoce universalmente dentro de la teoría de algoritmos como el peor caso de ejecución.

Definición (Peor caso): El tiempo de funcionamiento de un algoritmo $\mathcal{A}$ en el peor caso es definido por la función $t^*_{\mathcal{A}}(n)$, como: $t^*_{\mathcal{A}}(n) = \max\{ t_{\mathcal{A}}(w) \mid w \in \Sigma^* \wedge |w| = n\}$ siendo $|w|$ el tamaño de w. Esto es, número de operaciones necesario para la distribución probabilística más pesimista de los datos de entrada del algoritmo.

En este orden de ideas, la medida de la variable $t^*_{\mathcal{A}}(n)$, sobre el conjunto Σ^* suele ser de gran importancia dentro de la teoría de algoritmos, esto en virtud a que dicha media proporcionará directamente el nivel de eficiencia de un algoritmo, tal como se establecerá más adelante, así pues un algoritmo que caracterice por un buen desempeño ante el peor caso siempre será un buen algoritmo, o expresado de mejor forma será un algoritmo eficiente.

Por otro lado y de manera análoga, se puede definir otro tipo de variable, y aunque no es muy práctico, servirá para establecer otro parámetro como lo es el caso promedio y que nos describirá el comportamiento más común o eficiencia media de un algoritmo. La variable a la que se hace referencia es lo que se conoce como el *mejor caso*, y es definida de manera análoga al peor caso de la siguiente manera:

Definición (Mejor caso): El tiempo de un algoritmo $\mathcal{A}$ en el mejor caso es definido por la función $t^{\sim}_{\mathcal{A}}(n)$, como: $t^{\sim}_{\mathcal{A}}(n) = \min\{ t_{\mathcal{A}}(w) \mid w \in \Sigma^* \wedge |w| = n\}$ siendo $|w|$ el tamaño de w. Esto es, número de operaciones necesario cuando los datos de entrada se encuentran distribuidos de la mejor y más sencilla forma posible para el algoritmo.

Es de resaltar que existe un amplio número de algoritmos que si bien al ejecutar el peor caso no tienen un buen desempeño, si tiene un comportamiento razonablemente eficiente en

la mayoría de los casos que se presentan. Para ello definimos lo que se conocerá como *caso promedio*.

Definición (Caso promedio): Sean $w^*, w^{\sim} \in \Sigma^*$ entradas de $\mathcal{A}$, tales que w^* reduce a $t_{\mathcal{A}}$ al peor caso y $w^{\sim}$ reduce a $t_{\mathcal{A}}$ al mejor caso, esto es $t^*_{\mathcal{A}}(w^*) = \max\{ t_{\mathcal{A}}(w) \mid w \in \Sigma^* \wedge |w| = n\}$ y $t^{\sim}_{\mathcal{A}}(w^{\sim}) = \min\{ t_{\mathcal{A}}(w) \mid w \in \Sigma^* \wedge |w| = n\}$. Definimos el caso promedio como $\bar{t}_{\mathcal{A}}(w)$, tal que $t^{\sim}_{\mathcal{A}}(w^{\sim}) \leq \bar{t}_{\mathcal{A}}(w) \leq t^*_{\mathcal{A}}(w^*)$, siendo w entradas equiprobables, con $\bar{t}_{\mathcal{A}}(n) = \sum_{w \in \Sigma^*} P(w).t_{\mathcal{A}}(w)$ donde $P(w)$ es la probabilidad de ocurrencia de w.

Por otro lado y como se mencionó anteriormente, existen algunos factores externos al algoritmo que tienen incidencia directa en el tiempo de ejecución, como pueden ser la velocidad de ejecución de instrucciones por parte de la unidad de procesos del ordenador que lo ejecuta o la calidad del código generado por el compilador. Por tal motivo y en virtud a que la potencia de los ordenadores crece a un gran ritmo (como bien se establece en la ley de Moore, se duplica anualmente), es fundamental analizar los algoritmos que se trataran más adelante con algún nivel de independencia de estos factores; es decir, se buscaran estimaciones generales y que sean ampliamente válidas. Para ello se definirán los límites asintóticos como un aspecto más general, que permitirá facilitar el estudio de la eficiencia de los algoritmos en cuanto su complejidad se refiere, tal como se muestra a continuación.

Sea $f: \mathbb{N} \to \mathbb{R}^+$ una función cualquiera, normalmente consideramos el tiempo de ejecución del algoritmo $\mathcal{A}$ como una función $f(n)$, del tamaño *n* de la entrada w, (esto es $|w| = n$), que bien puede cumplir los siguientes argumentos:

Límite superior asintótico: Sea $g: \mathbb{N} \to \mathbb{R}^+$ una función, y sea $O(g(n))$ la clase dada por:

$$O(g(n)) = \{ f: \mathbb{N} \to \mathbb{R}^+ \mid \exists_{c \in \mathbb{R}^+} \wedge \exists_{n_0 \in \mathbb{N}} : \forall_{n \geq n_0}\ ,\ 0 \leq f(n) \leq c.g(n) \}$$

Decimos que g es un límite superior asintótico de f si $f \in O(g(n))$, o como bien se referirá de ahora en adelante $f = O(g(n))$.

Límite inferior asintótico: Sea $g: \mathbb{N} \to \mathbb{R}^+$ una función, y sea $\Omega(g(n))$ la clase dada por:

$$\Omega(g(n)) = \{ f: \mathbb{N} \to \mathbb{R}^+ \mid \exists_{c \in \mathbb{R}^+} \wedge \exists_{n_0 \in \mathbb{N}} : \forall_{n \geq n_0}, \ 0 \leq c.g(n) \leq f(n) \}$$

Decimos que g es un límite inferior asintótico de f si $f \in \Omega(g(n))$, o $f = \Omega(g(n))$.

Límite exacto asintótico: Sea $g: \mathbb{N} \to \mathbb{R}^+$ una función, y consideremos la clase $\Theta(g(n))$ la dada por:

$$\Theta(g(n)) = \{ f: \mathbb{N} \to \mathbb{R}^+ \mid \exists_{c_1, c_2 \in \mathbb{R}^+} \wedge \exists_{n_0 \in \mathbb{N}} : \forall_{n \geq n_0}, c_1.g(n) \leq f(n) \leq c_2.g(n) \}$$

Decimos que g es un límite exacto asintótico de f si $f \in \Theta(g(n))$, o $f = \Theta(g(n))$. Nótese que el conjunto así definido es equivalente a afirmar que $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$. En este aspecto la definición sintetiza el hecho que la razón o tasa de crecimiento asintótico de las funciones f es similar al de la función g.

Notación *o*: Se dice que $f = o(g(n))$ si para $\forall c$, $\exists_{n_0 \in \mathbb{N}}$ tal que $0 \leq f(n) \leq c.g(n)$ $\forall_{n \geq n_0}$

Observación: Aunque $O(g)$, $\Omega(g)$, $\Theta(g)$, y $o(g)$ son clases o conjuntos de funciones, por costumbre y convenio general se escribe: $f = O(g(n))$, $f = \Omega(g(n))$, $f = \Theta(g(n))$ respectivamente, en lugar de $f \in O(g(n))$, $f \in \Omega(g(n))$, $f \in \Theta(g(n))$. Pues sobra

mencionar que $f = \mathrm{O}(g(n)),\quad f = \Omega(g(n))$, o bien $f = \Theta(g(n))$ carecen de sentido lógico.

7.3. CLASES DE ALGORITMOS

Dentro de las ciencias computacionales y de manera general podemos establecer las siguientes clases de algoritmos según su naturaleza funcional: *Algoritmos cualitativos y Algoritmos cuantitativos*. Los algoritmos cualitativos se definen como aquellos cuyas instancias no están asociadas a cálculos numéricos. Por otro lado, y al contrario de la definición anterior, los algoritmos cuantitativos involucran en sus instancias o iteraciones cálculos de tipo numérico. Sin embargo y para el interés que aquí se presenta, los algoritmos RSA y de Shor que se verán más adelante son algoritmos de naturaleza cuantitativa, por tal efecto y a conveniencia para el tema de estudio es posible establecer una clasificación más acertada de los algoritmos a tratar.

Algoritmos polinomales: Definiremos un algoritmo polinomial como todo aquel que puede ser ejecutado en tiempo polinomico en su peor caso, es decir, si al considerar la distribución más pesimista de sus datos de entrada (n), su límite superior asintótico es de orden O(n^k), siendo $n \in \mathbb{N}$ y k una constante, con $k \in Z^+$. Son en principio algoritmos eficientes.

Algoritmos exponenciales: Un algoritmo es definido como exponencial, si y solo si este no puede ser acotado asintóticamente por ninguna función polinomial. Esto es, dado $n \in \mathbb{N}$, un algoritmo es exponencial si y solo si su orden de complejidad está dado por O(k^n), para algún $k \in Z^+$, donde n corresponde al tamaño de la entrada. Son en principio algoritmos ineficientes al considerar valores de n grandes.

Algoritmos subexponenciales: Un algoritmo se denomina subexpeonencial si en el peor de los casos la función de ejecución es de la forma $b^{O(n)}$ donde b es la base del sistema numérico que se implementa, y $n \in \mathbb{N}$, siendo n el tamaño de la entrada o más generalmente el tamaño de la instancia. (Nótese que en sistema binario tal forma seria $2^{O(n)}$ mientras que para un diedro o subgrupo 8-ciclico esta seria $8^{O(n)}$. Son asintóticamente más rápidos de los exponenciales pero significativamente más lento que los algoritmos polinomiales.

Para finalizar este apartado y con el propósito de introducir al lector a la temática, se presenta la siguiente tabla de caracterización de los algoritmos a tratar.

Tabla 1: Resumen y Clasificación de los Algoritmos a Tratar

	Tipo de Criptografía	Tipo de Intercambio de Clave	Tipo de Algoritmo	Observaciones
RSA	Clásico	Pública (Asimétrica)	Cifrado	El tipo de criptografía empleada para el sistema de cifrado RSA está basado en un problema clásico de gran complejidad matemática: la descomposición prima de números grandes.
Criba General del Cuerpo de Números	Clásico	No aplica	Factorización Prima	No existen intercambios de claves en este tipo de algoritmos en virtud a que no hay procesos de cifrado o descifrado directo de información. En este sentido el algoritmo es implementado para dar con la clave privada, sin embargo no es eficiente al realizar dicha tarea
Algoritmo de Fermat (método)	Clásico	No aplica	Factorización Prima	No existen intercambios de claves. En este sentido el algoritmo es implementado para dar con la clave privada, sin embargo no es eficiente al realizar dicha tarea
Algoritmo de Shor	Moderno	No aplica	Factorización Prima	No existen intercambios de claves. En este sentido el algoritmo es implementado para dar con la clave privada, y a diferencia de los otros es realiza dicha tarea en un tiempo eficiente.

Fuente: El Autor

Obsérvese que los algoritmos que se muestran en la tabla anterior son de naturalezas distintas, siendo el RSA el único algoritmo de cifrado mostrado en el grupo. A diferencia de este, el resto son algoritmos de factorización, por lo cual guardan una función distinta a la del cifrado, manteniendo estos una relación más estrecha a los algoritmos de descifrado (sin llegar a cumplir de manera explícita la función de un algoritmo de descifrado en sí, los cuales generalmente trabajan mediante el análisis de patrones dentro de los criptogramas, analizando la redundancia u otros elementos). Por tal motivo, y puesto que ninguno los algoritmos presentados en la tabla pueden ser clasificados como algoritmos simétricos, entonces no se podrá establecer de manera directa ninguna clasificación ya sea por tipo de cifrado de flujo o por bloques (algunos ejemplos de estos tipos de algoritmos son el Data Encryption Standard o DES, Advanced Encryption Standard o AES, RC5, entre otros).

8. ANÁLISIS

8.1. CONSIDERACIONES GENERALES

En conformidad a lo tratado hasta el momento y con el fin de establecer una independencia de los aspectos puntuales que se puedan asociarse a cualquier plataforma informática establecemos un conjunto de consideraciones fundamentales que estarán presentes durante todo el planteamiento de análisis a desarrollar. Estas consideraciones son las siguientes:

- El análisis del coste computacional de los algoritmos a tratar se estimara siempre en el peor de los casos. Esto proporcionará una visión muy clara de la eficiencia del algoritmo que se estudie.

- No se hará especial énfasis al coste computacional de los algoritmos en el caso promedio, esto en virtud a que dicha medida, depende exclusivamente del contenido de los datos de entrada y su tamaño, que por lo general se basan en funciones de distribución desconocidas o difíciles de tratar, sumado al hecho de que el estudio de dicho caso específico no mostrará la complejidad del algoritmo o su solides.

- En relación al primer punto, se tomará pues en consideración el tamaño *n* de los datos de entrada. Centraremos pues el análisis de la eficiencia de los algoritmos en base al tamaño del conjunto de datos que éstos manipulen. Así pues además de mostrar la eficiencia de cada algoritmo que se estudie, nos brindara también un soporte por el cual comparar la eficiencia de varios algoritmos con independencia de cualquier característica que no sea la rapidez con que este alcanza la solución teórica al problema tratado.

- Se entenderá por valores de *n* “grande” o "suficientemente grandes" aquellos valores correspondientes al tamaño de la entrada tan elevados que hacen que las

particularidades (líneas de código, programación, etc.) de un algoritmo sean despreciables frente al volumen teórico del cálculo a procesar.

- Nótese que al definir $\mathrm{O}(g(n))$ tal como se definió anteriormente, puede presentarse el hecho de que existan funciones $g'(n) \neq g(n)$ tal que:

$$\{\, f: \mathbb{N} \to \mathbb{R}^{+} \mid \exists_{c \in \mathbb{R}^{+}} \wedge \exists_{n_0 \in \mathbb{N}} : \forall_{n \geq n_0}\ ,\ 0 \leq f(n) \leq c.g(n) \ \wedge\ g(n)\ \leq g'(n)\ \}$$

  Lo cual es un hecho muy común, por tal motivo se establecerá que el límite superior asintótico (orden de complejidad, o representante del orden de $\mathrm{O}(g(n))$ a la función $g(n)$ más sencilla que se encuentre en el mismo orden.

- Se partirá del supuesto de que el análisis del coste de complejidad de cada una de las instrucciones de asignación, entrada/salida, u operaciones que no involucren en su ejecución ya sean variables estructuradas o que dependan del tamaño de la entrada *n* que opera el algoritmo $\mathcal{A}$. En este caso el límite asintótico es exacto, y para este tipo de instrucciones supondremos que tal coste computacional es unitario. Es decir, el coste de una operación elemental es $\Theta(1)$.

- El coste o complejidad computacional en el peor caso de una secuencia consecutiva de instrucciones se calcula sumando los pasos el coste computacional de cada una de las instrucciones. Esto es, sean $(F_1, F_2, \dots, F_n)$ una secuencia consecutiva de instrucciones o fragmentos, el cual está asociado a un coste computacional de cada a una de sus instrucciones dadas por $(f_1, f_2, \dots, f_n)$, con $F_k \to f_k$ para *k* =1,2,..,n, entonces, el coste computacional del fragmento $F_1; F_2; F_3; \dots; F_n$, está dado por: $f_1 + f_2 + \cdots + f_n$.

- El coste computacional para una estructura de selección tipo "*if (B)* ⇒ F_1 *; else* F_2, *end if* " se establece como: $g + \max\{ f_1, f_2 \}$. Donde f_1 y f_2 representan el coste

computacional de F_1 y F_2 respectivamente y siendo g el coste computacional de evaluar *B*.

- El coste computacional de un bucle "*while(B), do F*" se calcula mediante la siguiente expresión: $T = g + N.(f + g)$. Donde *N* es el número de iteraciones, *f* el coste computacional de la iteración *F* y *g* el coste computacional de evaluar *B*. Cabe resaltar, que dicha expresión responde a solo una estructura sencilla o elemental del bucle en la cual los valores de *f* y *g* no varían. De esta forma, y en virtud a que los valores de f y g pueden variar en cada iteración, y que por lo tanto habría que tenerlos en cuenta para el cálculo, se podría establecer una expresión más general del coste computacional para el ciclo *while* para *f* y *g* variables, pudiéndose definir de la siguiente forma: Sea *F* un fragmento de iteraciones, siendo f_i el coste de la *i-esima* iteración de *F*, y sea g_i el coste de evaluar *B*, entonces el coste computacional de fragmento "*while(B) , F*" estaría dado por:

  $$T(n) = \sum_{i=0}^{h(n)} f_i(n) + g_i(n)$$

  Siendo *h(n)* el número de iteraciones en el bucle, en función a la entrada *n* a procesar en el bucle. Por lo que, si $f = \max\{ f_i + g_i\}$ para i = 1,2, …, *h(n),* entonces el límite superior asintótico u orden de complejidad del fragmento estaría dado por $T = O(hf)$.

8.2. CRIPTOSISTEMA DE LLAVE PÚBLICA RSA

Los criptosistemas de clave pública consisten en una familia $\{ f_k : \mathcal{M} \to \mathcal{C} \mid k \in \mathcal{K} \}$ de *funciones de dirección única con trampa*, esto es, un conjunto de funciones de dirección única para las cuales y cada una existe un conjunto de datos específicos que permite invertirla de manera eficiente, entendiéndose como *función de dirección única* como aquella $f : X \to Y$ tal que $f(x)$ es computacionalmente sencilla de calcular para todo $x \in X$

pero computacionalmente impracticable o poco factible calcular $f^{-1}(y)$ para todo $y \in Y$. En este orden de ideas, los criptosistemas RSA de llave publica son un conjunto de técnicas o métodos bien estructurados que operan sobre un conjunto de datos para ser cifrados, en este caso particular tales sistemas soportan su funcionamiento a través de un algoritmo cuyo proceso de encriptado se basa en la exponenciación modular y mediante la generación de un par de llaves (una pública y de una privada), es decir, la hipótesis central de que para $n, e \in \mathbb{Z}^+$ dados, donde $n = p.q$ (siendo p y q primos grandes y no próximos entre sí, esto con el propósito de dificultar al máximo la factorización de n ante ataques por criptoanálisis), la función $m \to m^e \, (mod\, n)$ es una funcion de dirección única con trampa. En este sentido, un criptosistema en general, permite enviar mensajes codificados entre dos o más partes de tal manera que ningún tercero pueda descifrarlo (o por lo menos que cualquier intento por hacerlo sea una tarea computacionalmente impracticable desde el punto de vista de la computación clásica) o bien desarrollar firmas de autenticación de documentos. El algoritmo de cifrado de clave pública RSA como se verá posteriormente, centra la solides de su seguridad y su eficacia en la notable diferencia entre la facilidad de encontrar grandes números primos y la dificultad de factorizar un número como producto de dos grandes números primos.

8.2.1. Funcionamiento y aspectos centrales

En todo criptosistema de clave pública y en particular el RSA, los algoritmos implementados para su funcionamiento han de estar asociados inicialmente una dupla (P_A, S_A) de claves, por cada participante, formadas por una clave publica P_A y una clave secreta o privada S_A para un usuario A. En los algoritmos de cifrado RSA, tales claves están formadas por una dupla de enteros, así un sistema de cifrado entre dos usuarios A y B, estaría compuesto inicialmente por un algoritmo $\mathcal{A}$, el mensaje M y las duplas (P_A, S_A) y (P_B, S_B) de los usuarios A y B respectivamente, las cuales como se verá más adelante son funciones que cumplen un conjunto de características especifico.

Estas claves generalmente se construyen mediante sistemas de *generadores "aleatorios"* los cuales se basan en la implementación de algunos test de primalidad y de aleatoriedad. En un criptosistema de clave pública, las claves públicas P_A y P_B de los usuarios A y B, pueden ser mantenidas en un registro o directorio publico convenientemente, en efecto, se asume que cualquier usuario externo puede tener acceso a este tipo de claves, mientras que las claves privadas S_A y S_B son mantenidas en secreto por cada uno de los usuarios. Las claves públicas y secretas pueden ser aplicadas a manera de función a cualquier mensaje como pronto se verá a continuación. Para ello, es preciso definir los conjuntos $\mathcal{M}$ y $\mathcal{C}$, siendo $\mathcal{M}$ el conjunto de todos los mensajes admisibles, (entendiéndose un mensaje M como una secuencia o cadena de caracteres alfanuméricos concatenados, por lo que un mensaje podría ser una simple cadena de bits finita) y siendo $\mathcal{C}$ el conjunto de todas las cadenas de texto alfanumérico resultado del cifrado y asociadas a un $\mathrm{M} \in \mathcal{M}$, sin embargo como los conjuntos $\mathcal{M}$ y $\mathcal{C}$ están definidos en un mismo alfabeto, podemos contemplar un conjunto $\mathcal{D}$ formado por todas la secuencias concatenadas posibles de los elementos del alfabeto, por lo que siendo así entonces las funciones $P_A : \mathcal{M} \to \mathcal{C}$ y $S_A : \mathcal{C} \to \mathcal{M}$, asociadas a un usuario A, podrían ser definidas de manera más amplia $P_A : \mathcal{D} \to \mathcal{D}$ y $S_A : \mathcal{D} \to \mathcal{D}$, de esta manera las funciones implementadas en el algoritmo de encriptación, deben conservar las siguientes características:

- Las funciones $P_A : \mathcal{D} \to \mathcal{D}$ y $S_A : \mathcal{D} \to \mathcal{D}$, han de ser inyectivas, para la función P_A esto sería: dado $\mathrm{M}_1, \mathrm{M}_2 \in \mathcal{D}$ con $\mathrm{M}_1 \neq \mathrm{M}_2$ entonces $P_A(\mathrm{M}_1) = \mathrm{C}_1 \neq \mathrm{C}_2 = P_A(\mathrm{M}_2)$, con $\mathrm{C}_1, \mathrm{C}_2 \in \mathcal{D}$. Por lo que $\mathrm{M}_1, \mathrm{M}_2, \mathrm{C}_1, \mathrm{C}_2$ serian elementos de Σ^* correspondientes en una máquina de Turing.

- Las funciones P_A y S_A corresponden a permutaciones dentro del conjunto $\mathcal{D}$.

- Con el fin de preservar la solides del algoritmo y la seguridad, se asumirá las funciones $P_A(.)$ y $S_A(.)$ son eficientemente computables dadas las respectivas

claves P_A y S_A. Es decir, son fáciles de calcular siempre que se tengan respectivamente cada una de las claves asociadas.

- Las funciones $P_A : \mathcal{D} \rightarrow \mathcal{D}$ y $S_A : \mathcal{D} \rightarrow \mathcal{D}$ son funciones inversas asumidas como de dirección única con trampa. Por lo cual para todo $\mathrm{M} \in \mathcal{D}$

  $\mathrm{M} = S_A(\mathrm{C}) = S_A(P_A(\mathrm{M}))$ y $\mathrm{M} = P_A(\mathrm{C}) = P_A(S_A(\mathrm{M}))$

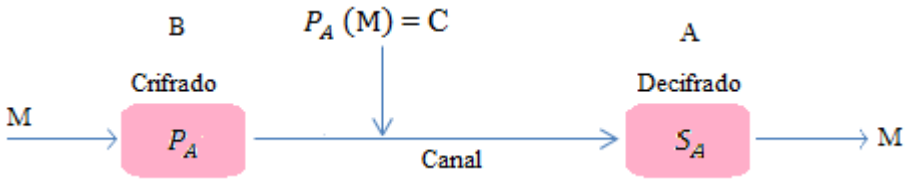
- Las funciones P_A y S_A que operan en el algoritmo deben ser resistentes a las colisiones, es decir: Si $\mathrm{M} \in \mathcal{M}$, deberá ser altamente improbable encontrar o calcular un $\mathrm{M}' \in \mathcal{D}$ con $\mathrm{M}' \neq \mathrm{M}$ tal que $P_A(\mathrm{M}) = P_A(\mathrm{M}')$. Análogamente para S_A, si $\mathrm{C} \in \mathcal{D}$ deberá ser altamente improbable encontrar o calcular un $\mathrm{C}' \in \mathcal{D}$ con $\mathrm{C} \neq \mathrm{C}'$ tal que $S_A(\mathrm{C}) = S_A(\mathrm{C}')$ y que permita calcular el mensaje original M. *Nótese que el hecho de que el algoritmo sea resistente a colisiones, es una consecuencia directa del contrareciproco de la definición de inyectividad relacionada en el primer punto*.

- Nótese que una de las propiedades asociadas a las funciones P_A y S_A que se sustrae del primer punto, es que si se define $P_A : \mathcal{M} \rightarrow \mathcal{C}$, como se pretendía en un principio, entonces la cardinalidad de $\mathcal{M}$ y $\mathcal{C}$ debe cumplir la siguiente relación: $card(\mathcal{M}) \leq card(\mathcal{C})$. Por este motivo y con el propósito de dificultar ataques por criptoanálisis, algunos algoritmos, construyen $\mathcal{M}$ y $\mathcal{C}$ en alfabetos distintos, de tal manera que se mantenga: $card(\mathcal{M}) < card(\mathcal{C})$. Sin embargo para nuestro caso el algoritmo RSA no contempla este caso.

El escenario a considerar que describe el funcionamiento es el siguiente: Supongamos que el usuario B tiene como propósito enviar un mensaje M al usuario A, y que sea indescifrable ante cualquier tercero que escudriñe el canal de comunicación. En este sentido, B procede primeramente a obtener la clave pública P_A de A, posteriormente B hace corresponder un el mensaje M con un texto ininteligible C (tarea que se realiza al operar la

función P_A (M)), el cual viaja a través del canal. Una vez C =P_A (M) es recibido por A, el usuario, aplica su clave secreta S_A sobre C, por lo cual y en virtud a que P_A y S_A son funciones inversas se obtendría S_A (C) = S_A $\left(P_A\,(\mathrm{M})\right) = \mathrm{M}$, obteniendo así el mensaje original. El esquema que describe con detalle el proceso es el siguiente.

Figura 1. Esquema de funcionamiento del cifrado



Fuente. El Autor

8.2.2. Estructuración del algoritmo RSA y fundamentos

Como se mencionó anteriormente, este criptosistema se inicia mediante la generación de una dupla (*P, S*) de claves específicas para cada usuario, la cual está integrada por una clave pública y una clave secreta. El algoritmo a ejecutar para un usuario que implemente el RSA se puede estructurar de la siguiente forma:

1. Se procede a seleccionar de manera aleatoria dos números primos p y q grandes, con $p \neq q$, asegurándose además de que en dicha selección se escojan primos de aproximadamente el mismo tamaño, pero lo menos próximos entre sí. Esto con el fin de maximizar la solides del cifrado ante ataques por factorización.

2. Se calcula $n = p.q$ el cual hará de modulo dentro de la exponenciación modular por el que se definirán las funciones *P* y *S*.

3. Se procede a calcular la función $\phi(n)$ de Euler, la cual y para nuestro caso particular estaría dada por $\phi(n) = (p-1)(q-1)$, seleccionando a su vez un entero *e* tal que $1 < e < \phi(n)$, siendo a su vez *mcd* $(e\ ,\ \phi(n)) = 1$.

4. Se procede a calcular *d* como inverso multiplicativo de *e* modulo $\phi(n)$, es decir el único entero *d*, tal que $1 < d < \phi(n)$ y $e.d \equiv 1\ (mod\ \phi(n))$.

5. Se publica se comparte a través de un directorio la dupla $P = (e, n)$ siendo esta la *clave pública* del usuario.

6. Se mantiene en secreto el valor de *d*, siendo el par $S = (d, n)$ la *clave secreta o privada* del usuario.

Seguidamente se implementan las funciones P y S asignando a $\mathbb{Z}_n$ como valor $\mathcal{D}$ de dominio. Así pues para cifrar y enviar un mensaje $\mathrm{M} \in \mathbb{Z}_n$ asociado con la clave pública $P = (e, n)$ del respectivo usuario, se procede a calcular $P(\mathrm{M}) = \mathrm{M}^e (mod\ n) = \mathrm{C}$. Por otro lado para descifrar un mensaje C asociado a la clave secreta $S = (d, n)$ el usuario procede a calcular M a través de la siguiente función $S(C) = \mathrm{C}^d (mod\ n) = \mathrm{M}$.

A continuación y para finalizar esta parte, se hará una pequeña observación de algunos de los pasos tratados anteriormente, estas observaciones garantizan el desarrollo de los puntos más importantes del RSA, para ello se empezará introduciendo un algoritmo ampliamente conocido el algoritmo de Euclides, además de ello se presentará un teorema fundamental para la solución de ecuaciones lineales modulares:

Algoritmo de Euclides: Este algoritmo es un método que no solo sirve para calcular el máximo común divisor de dos número, sino que también puede ser implementado para computar enteros *x* y *y* de la forma $d = mcd\ (a, b) = ax + by$ (y que por lo tanto nos servirá para calcular *d* como inverso de la ecuación en congruencias $e.d \equiv 1(mod\ \phi(n))$). El algoritmo consiste específicamente en encontrar el $mcd(a, b)$, siendo $a > b > 0$ y

basándose en la existencia de q_i y r_i (los cuales forman una cantidad finita de enteros positivos menores que *b*) tales que cumplen con el siguiente cómputo:

$a = q_1 b + r_1$ donde $0 < r_1 < b$;
$b = q_2 r_1 + r_2$ donde $0 < r_2 < r_1$;
$r_1 = q_3 r_2 + r_3$ donde $0 < r_3 < r_2$;
.
.
.
$r_{k-3} = q_{k-1} r_{k-2} + r_{k-1}$ donde $0 < r_{k-1} < r_{k-2}$;
$r_{k-2} = q_k r_{k-1} + r_k$ donde $0 < r_k < r_{k-1}$;
$r_{k-1} = q_{k+1} r_k + 0$;

Nótese que implementar el algoritmo con cada una de las operaciones anteriores implicaría el almacenamiento de cada uno de los r_i, por lo cual no sería muy eficiente en cuanto al uso del espacio en memoria cuando los valores de *a* son muy grandes en comparación a los de *b*, pues esto implicaría un número considerablemente grande de r_i a almacenar. Teniendo en cuenta esto, y siempre y cuando no se requieran los valores intermedios, el algoritmo puede ser expresado para calcular *mcd (a,b)* de manera más formal y eficiente así:

Input: *a , b enteros positivos mayores que 0*
Output: *mcd (a,b)*

Funcion *mcd (a,b) {*

Si $b = 0$ entonces:
Retorne *a*;

En otro caso:
El resultado es *mcd (b, a(mod b)*);
}

Teorema 1: Supongamos que la ecuación en congruencias dada por $ax \equiv b\ (mod\ n)$ es soluble. Sea x_0 una solución cualquiera de dicha ecuación y sea *mcd* (a , n) = *d* el máximo

común divisor entre *a* y *n*. Entonces, $ax \equiv b\ (mod\ n)$ posee exactamente *d* soluciones distintas modulo n, dadas por $x_k = x_0 + k(n/d)$ con *k = 0,1,2, ...(d* -1*)*.

Ahora sea $n > 1$ tomando b=1, y puesto que *mcd* (a , n) = 1, entonces por el teorema anterior se tiene que la ecuación en congruencias dada por $ax \equiv 1\ (mod\ n)$ posee exactamente una única solución. Obteniéndose así el siguiente corolario:

Corolario 1: Sea $n > 1$ y sean *a* y *n*, tales que *mcd* (a , n) = 1, entonces la ecuación en congruencias dada por $ax \equiv 1\ (mod\ n)$ tiene una única solución.

Por último, es posible obtener como resultado directo del teorema chino del residuo el siguiente corolario:

Corolario 2: Sean $n_1, n_2, \dots, n_m$ una secuencia de números relativamente primos a pares, y sea $n = n_1 n_2 \dots\ n_m$, entonces, para cualquier *x* y *a* enteros tenemos que $x \equiv a\ (mod\ n)$ si y solo si $x \equiv a\ (mod\ n_k)$ para *k = 1, 2, ... , m*.

Como se puede apreciar en el punto 3 que se describe en el algoritmo RSA, se había procedido a seleccionar *e* tal que 1 < e < $\phi(n)$, con *mcd* (e , $\phi(n)$) = 1, siendo así el inverso *d*, que se describe en la ecuación en congruencias e.d≡ 1(mod ϕ(n)) con 1< d < ϕ(n) presentada en el punto 4 del algoritmo RSA, tendría su existencia garantizada en virtud al corolario 1. Nótese que al disponer de la clave pública de un usuario la cual se describe en el punto 5 del RSA por el par $P = (e, n)$ (y que supone es conocida por todos los usuarios), el algoritmo de Euclides proporcionaría al usuario una forma eficiente de calcular *d* como inverso de *e* módulo ϕ(n), es decir, nos permitiría encontrar la solución *d*, de la ecuación en congruencias e.d≡ 1(mod ϕ(n)), esto desde que dicha ecuación en congruencias sea equivalente a solucionar la ecuación diofantica $e.d + \phi(n).y = 1$ siendo *d* y *y* variables enteras a calcular.

Por otro lado, para probar la exactitud del algoritmo RSA, es necesario probar que las funciones tratadas al principio y definidas por:

$$\left.\begin{aligned} P(\mathrm{M}) &= \mathrm{M}^{e}(mod\ n) = \mathrm{C} \\ S(C) &= \mathrm{C}^{d}(mod\ n) = \mathrm{M} \end{aligned}\right\}$$

Con $\mathrm{M} \in \mathbb{Z}_n$ son funciones inversas, es decir que satisfacen las relaciones $\mathrm{M} = S_A\,(P_A\,(\mathrm{M}))$ y $\mathrm{M} = P_A\,(S_A\,(\mathrm{M}))$ descritas en la parte de *Funcionamiento y Aspectos Centrales* del presente trabajo. Probaremos se establecerá ese resultado amanera del siguiente teorema:

Teorema 2 (Precisión del algoritmo): Las ecuaciones propias del algoritmo RSA descritas por $P(\mathrm{M}) = \mathrm{M}^{e}(mod\ n)$ y $S(\mathrm{M}) = \mathrm{M}^{d}(mod\ n)$ representan funciones inversas, esto es satisfacen las relaciones $\mathrm{M} = S(P(\mathrm{M}))$ y $\mathrm{M} = P(S(\mathrm{M}))$

Prueba: Desde que se tengan las ecuaciones $P(\mathrm{M}) = \mathrm{M}^{e}(mod\ n)$ y $S(\mathrm{M}) = \mathrm{M}^{d}(mod\ n)$, para todo $\mathrm{M} \in \mathbb{Z}_n$ se tiene que $S(P(\mathrm{M})) = P(S(M)) = \mathrm{M}^{ed}(mod\ n)$. Luego, siempre que *e* y *d* sean inversos multiplicativos modulo $\phi(\mathrm{n})$, tendremos que $e.d - 1 = k.\ \phi(\mathrm{n})$ es decir $ed = 1 + k(p-1)(q-1)$, para algún entero *k*, siendo *p* y *q* los factores primos de *n*, esto es $n = p.q$. Seguidamente, si $M \equiv 0\ (mod\ p)$, entonces $\mathrm{M}^{ed} \equiv 0\ (mod\ p)$, por lo que: $\mathrm{M}^{ed} \equiv \mathrm{M}\ (mod\ p)$, análogamente si $M \equiv 0\ (mod\ q)$ entonces $\mathrm{M}^{ed} \equiv \mathrm{M}\ (mod\ q)$, luego por corolario 2 se tiene que $\mathrm{M}^{ed} \equiv \mathrm{M}\ (mod\ n)$, por lo tanto $S(P(\mathrm{M})) = P(S(M))$. Ahora si $M \not\equiv 0\ (mod\ p)$, se tiene que:

$$\begin{aligned} \mathrm{M}^{ed} &\equiv \mathrm{M}^{1+k(p-1)(q-1)}\ (mod\ p) \\ &\equiv \mathrm{M}(\mathrm{M}^{(p-1)})^{k(q-1)}\ (mod\ p) \\ &\equiv \mathrm{M}((\mathrm{M}\ mod\ p)^{(p-1)})^{k(q-1)}\ (mod\ p) \\ &\equiv \mathrm{M}(1)^{k(q-1)}\ (mod\ p) \quad ; \end{aligned}$$

(lo cual se deduce del Teorema de Fermat pues $(\mathrm{M}\ mod\ p)^{(p-1)} \equiv 1\ (mod\ p)$ ya que $\mathrm{M}\ (mod\ p) \in \mathbb{Z}_n$ y *p* primos).

Así pues, $\mathrm{M}^{ed} \equiv \mathrm{M}(mod\ p)$, análogamente se obtiene que $\mathrm{M}^{ed} \equiv \mathrm{M}(mod\ q)$, luego por corolario 2 se tiene que $\mathrm{M}^{ed} \equiv \mathrm{M}\ (mod\ n)$, por lo tanto $S(P(\mathrm{M})) = P(S(M)) = \mathrm{M}$. Lo cual prueba que las funciones *S* y *P*, así definidas en el algoritmo RSA son funciones inversas.

8.2.3. Aspectos sobre la complejidad del RSA

El aspecto central del criptosistema RSA consta fundamentalmente en el algoritmo que lo determina, en este orden de ideas analizar la complejidad servirá para establecer la eficiencia del mismo. En efecto, para empezar el análisis consideremos el tamaño de la entrada del algoritmo la cual se establecerá por n con $n \in \mathbb{Z}$, en este sentido, se deberá enfocar el análisis en la complejidad asociada a cada una de las funciones generadoras de las claves tanto pública (P) como privada (S), y las claves respectivas generadas (esto es los pares $P = (e, n)$ y $S = (d, n)$). Para iniciar, se empezara por tratar con la cantidad o número de dígitos de los enteros que trataremos en función a una base decimal, en este caso para la entrada *n* y *e,* el número de dígitos es *log n* y *log e* respectivamente (en el desarrollo de este análisis se optó por implementar logaritmos en base diez, sin embargo cabe mencionar que la base logarítmica con que se analice el problema no inducirá variaciones en la complejidad, pues si $O(\log \mathrm{M})$ representa la complejidad para computar un cierto M entero de *m= log* M dígitos, entonces por la propiedad de cambio de base logarítmica tendríamos que $\log_b \mathrm{M} = (\log_k \mathrm{M})/(\log_k \mathrm{b})$ por lo que $\log_b \mathrm{M} = \alpha \log_k \mathrm{M}$ siendo $\alpha = 1/(\log_k \mathrm{b})$ una constante, así pues $O(\log \mathrm{M}) = O(\alpha \log_k \mathrm{M}) = O(\log_k \mathrm{M})$ por lo cual la complejidad permanecería invariante).

Para continuar con el análisis, se ha de tener en cuenta dos aspectos básicos asociado a la complejidad el primero se relaciona a la suma o resta, en efecto, este aspecto nos dice que dados dos números cuales quiera G y H enteros positivos de g = *log* G y h = *log* H dígitos decimales (y en general cualquier base) respectivamente, la complejidad computacional para calcular G + H o G - H en $\mathbb{Z}$ está dada por $O(g)$, lo cual resulta trivial al considerar la propiedades de O, asi pues se señalará dentro del contexto criptográfico que G + H o G – H

posee una complejidad $O(log\ \mathrm{G})$. Por otro lado el segundo aspecto a tener presente también es una consecuencia directa de las propiedades de O, y hace referencia al producto, en efecto, al considerar G y H, tenemos que la complejidad computacional de G.H $\in \mathbb{Z}$ esta dada por $O(g.h)$ o bien $O((log\ \mathrm{G}).(log\ \mathrm{H}))$, seguidamente nótese que si $g \geq h$, entonces g es una cota superior del par $\{\ g, h\ \}$ y por lo tanto si $h = O(g)$, tendríamos que la complejidad de G.H estaría dada por $O(g^2)$ es decir $O((log\ \mathrm{G})^2)$.

Por otro lado, recordemos que la división Euclidea de G entre H con G,H $\in \mathbb{Z}$, requiere, el computo de los enteros Q y R, con R $\in \{1,2, \dots, (H-1)\}$ siendo R y Q el residuo de dividir G entre H y Q el cociente respectivamente, tendremos siempre que G = Q.H + R con R< H, de esta forma si $q = log\ \mathrm{Q}$ y $r = log\ \mathrm{R}$ representan el número de dígitos decimales de Q y R, entonces en conformidad a lo que se ha tratado hasta el momento, la complejidad computacional de ejecutar una división Euclidea, dada por G = Q.H + R seria $O(q.h)$ es decir $O((log\ Q).(log\ \mathrm{H}))$, así pues siempre que G > H, si K es una cota superior del conjunto {G, H} entonces la complejidad del realizar una división Euclidea estaría dada por $O((log\ \mathrm{K})^2)$. Así pues, para una multiplicación modular cualquiera G.H(mod N) mientras GH(mod W) represente el resto de dividir GH por W, con W $\in \mathbb{Z}$ siendo $w = log\ \mathrm{W}$ el número de dígitos decimales de W, se tendrá que la complejidad computacional asociada a G.H(mod W) estaría dada por $O((log\ \mathrm{W})^2)$.

En conformidad a lo tratado hasta ahora, una ecuación exponencial modular, cualquiera dada por $\mathrm{G}^{\mathrm{H}}(mod\ \mathrm{W})$ con H > 0, necesitaría al menos $log\ \mathrm{H}$ multiplicaciones modulares o bien esto seria $log_2\ \mathrm{H}$ multiplicaciones binarias modulares en caso se considerará las representaciones binarias de H, G y W (recuérdese que $h = log\ \mathrm{H}$, representa el número de dígitos decimales de H), luego puesto que cada multiplicación modular tiene una complejidad de $O((log\ \mathrm{W})^2)$, entonces la complejidad de un exponencial modular dada por $\mathrm{G}^{\mathrm{H}}(mod\ \mathrm{W})$ está dada por $O(w^2.h)$ o bien $O((log\ \mathrm{W})^2.(log\ \mathrm{H}))$.

Así pues, si se considera que al aplicar una clave pública en el RSA se implementa la ecuación exponencial modular dada por $P(\mathrm{M}) = \mathrm{M}^{e}(mod\ n)$, siendo $1 < e < \phi(n)$, entonces en conformidad a lo anterior, la complejidad de aplicar la clave pública sobre un $\mathrm{M} \in \mathbb{Z}$, estaría dada por $O((log\ n)^2.(log\ e))$, sin embargo como $1 < e < \phi(n) < n$, entonces tendríamos que n es una cota superior de *e*, y por tanto $log\ e = O(log\ n)$, por lo cual $O((log\ n)^2.(log\ e))$, se reduce a $O((log\ n)^2)$, de esta manera la complejidad computacional de aplicar la clave pública sobre M, esta dada por $O((log\ n)^2)$.

Por otro lado y para finalizar, si se considera que para aplicar una clave secreta el RSA implementa la ecuación exponencial modular dada por $S(C) = \mathrm{C}^{d}(mod\ n)$, para un $\mathrm{C} \in \mathbb{Z}$, entonces tendríamos que la complejidad computacional de tal evento estaría dada por $O((log\ n)^2.(log\ d))$, sin embargo teniendo en cuenta que dentro del el algoritmo RSA los valores del tamaño la clave privada *d* son menores pero muy cercanos al de *n*, (es decir son comparables o semejantes) entonces $(log\ d) \sim (log\ n)$, por lo cual la complejidad computacional quedaría reducida a $O((log\ n)^2.(log\ n))$, es decir $O((log\ n)^3)$.

8.2.4. Métodos clásicos de factorización

Como puede apreciarse, la gran parte de la seguridad que ofrece el criptosistema RSA expuesto anteriormente, reposa esencialmente en la dificultad de factorizar enteros largos. En efecto, nótese que una vez teniendo *n* (el cual es posible obtenerlo del par que representa la clave pública $P = (e\ , n)$) si se dispone de un método eficiente para descomponer el modulo *n* en sus factores primos *p* y *q*, se podría posteriormente encontrar el modulo $\phi(n) = (p-1)(q-1)$ de la ecuación en congruencias $e.d \equiv 1\ (mod\ \phi(n))$ dada en el punto 4 de la descripción del algoritmo, en este sentido y mientras dicha ecuación sea equivalente a la ecuación diofantica dada por $e.d + \phi(\mathrm{n}).y = 1$ siendo *d* y *y* variables enteras a calcular, entonces un atacante podría romper fácilmente la seguridad proporcionada por el RSA implementando el par *p* y *q*, para resolver la ecuación anterior y

obtener la clave privada $S = (d\,, n)$ de la misma manera en que el usuario legitimo la obtuvo, lo cual podría realizarse fácilmente mediante el algoritmo de Euclides o el algoritmo de Euclides extendido, así pues si factorizar un entero largo es fácil, entonces romper RSA será fácil. Sin embargo, lograr descomponer un entero largo en sus factores primos resulta extremadamente dificultoso incluso para cualquier maquina en la actualidad basada en un diseño determinista de máquina de Turing, en resumidas cuentas el problema solo puede ser resuelto en tiempo exponencial mediante los sistemas de cómputo actuales (independientemente de su poder de procesamiento). *Por lo tanto, cualquier intento por romper el criptosistema debe estar enfocado en dos aspectos: el primero es centrar la solución del problema de la factorización de enteros largos mediante un algoritmo eficiente, es decir un algoritmo que permita desarrollar en un tiempo polinomial tal factorización; y el segundo aspecto es desarrollar un sistema de cómputo no determinista, y con capacidad de procesar simultáneamente los varios estados de las posibles soluciones al problema y en la cual pueda ejecutarse dicho algoritmo*. En efecto, y como se verá mucho más adelante el algoritmo de Shor, y las cualidades de algunas máquinas no determinísticas que se empiezan a desarrollar recientemente pueden dar una solución eficiente a dicho problema.

Seguidamente, se mostrara de manera resumida algunos algoritmos de factorización, sin embargo cabe mencionar que dichos algoritmos no son eficientes cuando se trata de enteros largos; en este sentido y el "mejor" tiempo de factorización conocido hasta el momento implementado para una máquina de Turing determinista, es presentado por la criba del Cuerpo de Números, el cual nos ofrece una solución al problema en un tiempo exponencial asintótico de tipo $exp[(c + o\,(1))((logn)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}]$.

Antes de mostrar cada uno de estos algoritmos se procederá a hacer un conjunto de consideraciones válidas y generalmente implementadas dentro de los criptositemas RSA de calve publica, estas consideraciones pueden catalogarse como un conjunto de aspectos

complementarios al algoritmo para explotar la complejidad del mismo y producir una mayor solides en su cifrado.

- El generador de secuencias aleatorio, debe presentar un par $(p\,,q)$ de primos grandes (usualmente de 1024 bits de longitud como mínimo, pudiendo ser incluso de 10^{200} dígitos), con $p \neq q$ tal que $n = p.q$. En este sentido se tomaran tales p y q que no estén lo suficientemente próximos entre sí.

- En conformidad al punto anterior, y en virtud a la gran variedad de algoritmos probabilísticos existentes que permiten constatar con un alto grado de certeza si un número es primo o compuesto, se evitaran dentro del esquema de cifrado asimétricos la selección de módulos *n* de tipo $n = p$ con *p* primo (en efecto, esto es uno de los primeros pasos a constatar en un ataque por criptoanálisis a los algoritmos de cifrado de clave pública como el RSA), de esta manera y por ser mucho más eficiente comprobar si un número es primo que tratar de factorizarlo, se recomienda complementar los criptosostemas RSA por herramientas que permitan aplicar de manera predeterminada un test de primalidad para asegurarse de que en efecto *n* pueda ser descompuesto realmente de una manera no trivial. Dentro de estos métodos probabilísticos se encuentran los test de primalidad Lehmann o el método de Miller-Rabin el cual resulta ser mucho más eficiente que el primero.

- Finalmente y en contraste al primer punto, queda la posibilidad de que n tenga un único factor primo elevado a una potencia mayor que 1. En este caso cabe resaltar que existen métodos capaces de verificar de manera eficiente si *n* es una potencia perfecta, es decir si *n* puede ser expresado mediante la forma x^m con $m > 1$ y x, m enteros. Por lo cual lo *n* seleccionados para implementar el RSA no serán en ningún caso potencias perfectas de ningún primo.

Teniendo en cuenta estos tres aspectos, cualquier algoritmo de factorización a implementar para el desarrollo de un criptoanálisis, ha de partir de la suposición de que la elección de tal

n se realizó cumpliendo cada uno de estos puntos durante el proceso de cifrado. Siendo así, se procederá a continuación a presentar algunos métodos de factorización conocidos y sus respectivas objeciones, incluyendo entre los cuales la criba general del cuerpo de números, siendo este último el que proporciona un "mejor" tiempo en la solución del problema.

Algoritmo de división por tentativa: El funcionamiento de este algoritmo radica en dividir *n* entre todo número primo *p*, tal que $2 \leq p \leq \sqrt{n}$, si el resultado de la división resulta ser un número entero, entonces *p* se clasifica como un factor de *n*. Obsérvese que este método requeriría en el peor de los casos ejecutar $\sqrt{n}/2$ operaciones (divisiones), por lo cual la complejidad de algoritmo seria de $O(2^{\log n})$ siendo $\log n$ el tamaño de la entrada. Se puede referenciar de la misma forma el método de *Monte Carlo*, o el método $p-1$ *de Pollard*, entre otros, sin embargo, todos estos métodos se caracterizan por ser notablemente lentos en el proceso de factorización, resultando así completamente ineficientes cuando se trata de números enteros demasiado grandes como los tratados en el RSA.

Método de Fermat: El método consiste en representar *n* mediante la forma $n = x^2 - y^2$ siendo $x, y \in \mathbb{Z}$, con $x, y \geq 1$. Así pues, $n = (x+y)(x-y) = a.b$ donde a y b serán los factores primos de *n*. Se inicia tomando $x_0 = \lceil \sqrt{n} \rceil$, es decir tomando el mínimo número entero no inferior a $\sqrt{n}$. Posteriormente se procede a comprobar si $y_0 = x_0^2 - n$ es un cuadrado perfecto, en caso así sea entonces se ha logrado una factorización de *n*, en caso contrario se procede a calcular $x_{k+1} = x_k + 1$ y usando $y_{k+1} = x_{k+1}^2 - n$ se puede calcular el siguiente y_k. Repitiendo este proceso, se habrá dado con una factorización de *n* en cuanto se encuentre un y_k que sea cuadrado perfecto de un entero positivo. Nótese que el peor caso se alcanzaría cuando la entrada *n* es un numero primo (pues de ser así, el polinomio dado por $y_k = x_k^2 - n$ seria irreducible en $\mathbb{Z}_n[x]$, por lo cual solo podría descomponerse de manera trivial, es decir $n = n.1$), así pues se ejecutar el algoritmo en el peor de los casos requeriría un total de $O(n)$ pasos, por lo tanto para una entrada *n*, de tamaño $m = \log n$, se tendría por la linealidad de $O(n)$ en *n*, que el número de pasos o complejidad en función al tamaño de la entrada es $O(10^{\log n})$ (recuérdese que esto es

mientras *n* este en sistema decimal, por lo que $\log n$ designa el número de dígitos decimales de *n*, siendo así la complejidad para el algoritmo implementando *n* en sistema binario seria, $O(10^{\log_2 n})$), es decir la complejidad presentada por este método es de naturaleza exponencial, lo cual no haría de este un método eficiente para la factorización de números grandes (siendo apropiado solo para enteros $n \leq 10^6$).

Criba general de campo de números: En la actualidad, es uno de los algoritmos de factorización más complejos pero más eficiente que se puede implementar sistemas de cómputo actuales (basados en un diseño determinista de máquina de Turing). Este algoritmo radica en los siguientes pasos:

1) Dada una entrada *n*, se empieza expresando a esta de la forma $n = r^e + s$ para luego determinar un polinomio mónico e irreducible en $\mathbb{Z}[x]$, de la siguiente forma: se procede a seleccionar un grado *d* de extensión para el polinomio, con $d \in \mathbb{Z}^+$, luego de ello, sea $k \in \mathbb{Z}^+$ el minimal tal que $k.d \geq e$, siendo asi entonces tendríamos que $r^{kd} \equiv sr^{kd-e}(mod\ n)$, así pues, si $c = sr^{kd-e}$, y $m = r^k$, por lo cual $m^d \equiv c\ (mod\ n)$, lo cual nos induce a determinar el polinomio mónico dado como $f(X) = X^d - c\ \in \mathbb{Z}[x]$.

2) Sea α una raíz de $f(X)$, esto es $f(\alpha) = 0$, se procede a definir el cuerpo dado por $\mathbb{Z}[\alpha]$.

3) Se selecciona una base B-lisa (o también llamada B-uniforme), de tal forma que para enteros *a* y *b*, con $b \geq 1$ y $mcd(a, b) = 1$, entonces $a + mb$ y $N(a + \alpha b)$ son B-lisos (es decir, que pueden ser expresados como productos de un conjunto de primos pequeños menores o iguales a B), donde $N(a + \alpha b)$, representa la norma de $a + \alpha b$, definiéndose dicha norma como $N(a + \alpha b) = a^d - c(-b)^d$. Así pues: $|N(a + \alpha b)| = \prod_{p\ primo, p \leq B}\ p^{v_p}$ y $|a + mb| = \prod_{p\ primo, p \leq B}\ p^{w_p}$, para v_p , $w_p \in \mathbb{Z}^+$. (Este valor de B se determina experimentalmente).

4) Se procede a calcular los $a, b \in \mathbb{Z}$, del punto anterior, partiendo del supuesto de que se puede implementar la factorización prima de $N(a + \alpha b)$ para obtener la factorización de $a + \alpha b$ en elementos primos y unidades, esto es: $a + \alpha b = (\prod_{u \in U} u^{t_u}).(\prod_{g \in G} g^{v_g})$, con $t_u \in \mathbb{Z}$ y $v_g \in \mathbb{Z}^+$. Siendo U un conjunto predeterminado de generadores del grupo de unidades y G es el conjunto generador de los ideales primos de $\mathbb{Z}[\alpha]$ de normas primas menores o iguales B. Así pues, tendríamos que $a + \alpha b$ es G-liso y $a + mb$ es B-liso, es decir: $a + \alpha b$ y $a + mb$ están completamente factorizados en G y B respectivamente.

5) Se procede a construir una matriz de factores, por lo cual se procede a encontrar pares (a, b) más que la suma de los cardinales de la base de los factores, esto es $card(U) + card(G) + \pi(B)$, donde $\pi(B)$ representa el número de primos menores o iguales a B. Si tenemos una cantidad de (a, b) mayor que la suma anterior, entonces se podrá encontrar enteros $x(a, b) \in \{0,1\}$ (sin ser todos cero, por lo cual la combinación de pares a tratar seria linealmente dependiente), de manera que $\prod_{(a,b)} (a + \alpha b)^{x(a,b)} = [(\prod_{u \in U} u^{\bar{t}_u}).(\prod_{g \in G} g^{\bar{v}_g})]^2$, y además $\prod_{(a,b)} (a + mb)^{x(a,b)} = (\prod_{p\ primo, p \leq B} p^{\bar{w}_p})^2$, (nótese que al buscar los pares (a, b) dependientes implica la existencia de bases de factores comunes).

6) Al tener en cuenta los elementos del punto 1, se procede a considerar el homomorfismo de anillos establecido de la siguiente forma: $\varphi: \mathbb{Z}[\alpha] \to \mathbb{Z}/n\mathbb{Z}$, definido por: $\varphi(\alpha) = m\ (mod\ n)$. Siendo así, en conformidad al punto 4 se sigue que: Ec. A $(\prod_{u \in U} \varphi(u)^{t_u})(\prod_{g \in G} \varphi(g)^{v_g}) = \prod_{p\ primo, p \leq B} p^{w_p}\ (mod\ n)$, así pues del punto 5, y en conformidad al homomorfismo así definido se obtiene que: $[(\prod_{u \in U} \varphi(u)^{\bar{t}_u}).(\prod_{g \in G} \varphi(g)^{\bar{v}_g})]^2 = [(\prod_{p\ primo, p \leq B} p^{\bar{w}_p})^2 (mod\ n)]$, Obsérvese que podemos implementar el método de eliminación de Gauss módulo 2, al vector correspondiente de los exponentes de la Ec. A, para hallar los pares (a, b).

7) Luego del punto 6, podemos encontrar enteros y, z tal que $y^2 \equiv z^2 (mod\, n)$ por lo cual al computar(x, y) y calcular $mcd(n, y - z)$ se hallara el divisor de *n*, obteniendo así una factorización no trivial de *n*.

Es notable el alto grado de dificultad que este algoritmo posee ante los primeros mencionados, como también se puede observar, existen parámetros tales como la cota superior de la base de primos que bien hemos (señalado por B), el cual resulta bastante conveniente obtenerlo de manera experimental. En este aspecto, solo aquellos enteros que son lisos en cierto sentido puede ser implementados en el algoritmo, en consecuencia el valor esperado del número de enteros lisos en la secuencia de la base a elegir jugarían un papel importante en el análisis del tiempo de ejecución del algoritmo. Por otro lado y debido a la naturaleza intrincada del algoritmo, el análisis asintótico correspondiente a la complejidad, es más conveniente realizarlo de manera heurística, en este caso el cálculo de la complejidad se reduce a un problema de optimización, el cual consiste en determinar *¿cuán grande debe ser un cierto k para que* $P(x, k) \geq \frac{1}{2}$, *siendo* $P(x, k)$ *la probabilidad de que algún conjunto no vacío* $\mathfrak{X} \subset \{1,2, \dots [k]\}$ *exista siendo* $(\prod_{i \in \mathfrak{X}} s_i)$ *un cuadrado?*, con $x, k \in \mathbb{R}^+$y habiendo elegido predeterminadamente un conjunto aleatorio e independientes de enteros $s_1, s_2, \dots, s_{[k]}$ distribuidos uniformemente en $[1, x]$. En efecto, existe una solución a este problema, la cual reposa en forma de teorema y está previamente demostrado en uno de los trabajos más notables de Carl Pomerance, en el que ofrece una solución dada por $\lim_{x \to \infty} P(x,\ exp^{[(1+\varepsilon)\ \sqrt{2\,(\log x)(\log\log x)}]}) = 1$, para todo $\varepsilon > 0$. Así, pues, al considerar $n = r^e + s$, y un $\varepsilon > 0$ arbitrario pero fijo, se tendría al tomar los parámetros $d = (2/\beta)^{1/2}.(\log n / \log\log n)^{\ 1/3}$ y $B = exp^{[\beta\,(\log n)^{1/3}\,(\log\log n)^{2/3}}$, con $\beta = (8/9)^{1/3} + \varepsilon$, que el tiempo de ejecución estaría acotado por $exp^{[(64/9)^{1/3}\,(\log n)^{\ 1/3}\,(\log\log n)^{2/3}]}$ es decir por lo cual la complejidad del algoritmo estaría dada por $O(exp^{[((64/9)^{1/3}+o(1)).(\log n)^{\ 1/3}.(\log\log n)^{2/3}]})$. Este resultado puede constatarse

más detalladamente en el trabajo "*The number field sieve" de Carl Pomerance*[2] y *"The number field sieve" de A.K. Lenstra*[3], respectivamente.

8.2.5. Ejemplo Proceso de Cifrado RSA

A continuación y a modo ilustrativo se sintetizará mediante un ejemplo todo lo expuesto hasta el momento en lo concerniente al criptosistema RSA. En efecto, consideremos Σ un alfabeto el cual consta de 27 letras, en el cual se ha asignado los valores siguientes A=00, B=01, C=02, …, Z=26, [ ]=27 (para espacio en blanco).

1° Escojamos *p*=29 y *q*=31, siendo así entonces $n = p.q = 899$.

2° Se procede a calcular por $\phi(n) = (p-1)(q-1) = 840$, seleccionando a su vez un entero e tal que $1 < e < \phi(n)$, siendo a su vez *mcd* (e , $\phi(n)$) = 1. En este caso, al seleccionar $e = 37$, se puede observar que claramente cumple con esta condición.

3° Se halla d tal que $e.d \equiv 1\ (mod\ \phi(n))$, con $1 < d < \phi(n)$, es decir se resolverá la ecuación diofantica dada por $37d - 840.y = 1$ siendo d y y las variables a calcular, en efecto, este paso puede realizarse implementando el algoritmo de Euclides (o bien su versión extendida) contemplando $\phi(n)/e$, obteniéndose al final la ecuación dada por $1 = 37.(-227) + 840.(10)$, es decir $d_0 = -227$ y $y_0 = 10$ son una solución particular de dicha ecuación, luego las soluciones generales estarían dadas por $d = -227 + 840t$ y $y = 10 - 37t$ donde t es un entero cualquiera. Siendo así y considerando un valor de d positivo, tendríamos que para $t = 1$, el primer valor positivo para d es $d = 613$. Por lo tanto esta será nuestra clave a implementar.

[2] *The number field sieve. En este documento se demuestra heurísticamente la complejidad algorítmica de la criba general del cuerpo de número.*
[3] *The number field sieve. Se suministra una demostración más detallada del problema de la factorización.*

4° Se designa entonces como claves públicas y privadas los pares dados por $P = (37{,}899)$ y $S = (613{,}899)$, respectivamente. Manteniéndose *S* en secreto por el respectivo usuario.

Supongamos ahora que se desea cifrar un cierto mensaje M, implementando las claves antes deducidas, y supongamos que se desea encriptar bloques de 2 letras en bloques de 3 letras (es decir $27^2 < 899 < 27^3$).

5° Sea M: "SENADO" el texto plano a encriptar, entonces al implementar el alfabeto Σ se obtiene el siguiente cifrado:

S→19, E→04, N→13, A→00, D→03, O→15.

Los bloques a cifrar son: (19,04) (13,00) (03,15). Ahora expresemos cada bloque como un número en base 27 (número de letras del alfabeto), esto es:

$$(19{,}04) = 19 \times 27^0 + 04 \times 27^1 = 127$$
$$(13{,}00) = 13 \times 27^0 + 00 \times 27^1 = 13$$
$$(03{,}15) = 03 \times 27^0 + 15 \times 27^1 = 408$$

6° Luego se procede a obtener el criptograma C implementando la igualdad dada por $P(\mathrm{M}) = \mathrm{M}^e (mod\ n) = \mathrm{C}$, de la siguiente manera:

$$127^{37} (mod\ 899) = 234 \ \rightarrow c_1$$
$$13^{37} (mod\ 899) = \ 208 \ \rightarrow c_2$$
$$408^{37} (mod\ 899) = \ 222 \rightarrow c_3$$

Seguidamente se expresa cada c_i en base 27, esto teniendo presente que se obtendrán 3 componentes o letras por bloque.

$$234 = 18 \times 27^0 + 08 \times 27^1 + 00 \times 27^2 \Leftrightarrow (18{,}08{,}00) \ \Leftrightarrow (\mathrm{RIA})$$

$208 = 19 \times 27^0 + 07 \times 27^1 + 00 \times 27^2 \Leftrightarrow (19{,}07{,}00) \Leftrightarrow (\text{SHA})$
$222 = 06 \times 27^0 + 08 \times 27^1 + 00 \times 27^2 \Leftrightarrow (06{,}08{,}00) \Leftrightarrow (\text{GIA})$

Así pues el criptograma que se mostrara es: RIASHAGIA

7° Para descifrar el mensaje, se implementa la igualdad $S(C) = \mathrm{C}^d(mod\ n) = \mathrm{M}$. Así pues realizando el proceso inverso sobre cada uno de los c_i se obtendría que:

$234^{613}(mod\ 899) = 127 \rightarrow m_1$
$208^{613}(mod\ 899) = 13 \rightarrow m_2$
$222^{613}(mod\ 899) = 408 \rightarrow m_1$

Lo cual expresado en base 27, seria:

$127 = 19 \times 27^0 + 04 \times 27^1 \Leftrightarrow (19{,}04) \Leftrightarrow (\text{SE})$
$13 = 13 \times 27^0 + 00 \times 27^1 \Leftrightarrow (13{,}00) \Leftrightarrow (\text{NA})$
$408 = 03 \times 27^0 + 15 \times 27^1 \Leftrightarrow (03{,}15) \Leftrightarrow (\text{DO})$

Obteniéndose de esta forma el texto plano que se presentó al inicio M: “SENADO”.

8.3. EL ALGORITMO DE SHOR COMO SOLUCIÓN A LA FACTORIZACIÓN

Como bien se ha establecido en el apartado anterior, la fortaleza de la seguridad del RSA como sistema de llave publica, radica única y exclusivamente en la dificultad que requiere en encontrar los factores primos de un entero largo, en este sentido, también se mostró que dado un par (p, q) de primos fuertes (o largos) tales que $n = p.q$ siendo n el modulo correspondiente las funciones P y S, entonces el problema de dicha factorización es resuelto en los sistemas de procesamiento actuales en un tiempo asintótico dado por: $O(exp^{[((64/9)^{1/3}+o(1)).(\log n)^{1/3}.(\log\log n)^{2/3}]})$, (esto independientemente del poder computacional que se disponga, siempre que esta pueda ser representada mediante una máquina de Turing determinista). Para tener una idea de la complejidad del problema, un numero de 512 bits (es decir del orden de $\log(2^{512}) = 154$ dígitos decimales) podría tomar tiempo cercano a los 30.000 años, suponiendo un poder de computo de 1 millón de instrucciones por segundo, mientras que para una clave de 1024 bits el tiempo de computo estimado sería de unos 300.000.000.000 años, de esta manera y en vista de que no hay suficiente poder de procesamiento actual, dicha hazaña sería una tarea computacionalmente impracticable. Sin embargo, y a pesar de estos desafíos, se han podido factorizar números de hasta 768 bits (véase reto RSA-768[4]) en un tiempo eficiente de 2 años implementando una red de procesamiento en paralelo y algunas técnicas matemáticas, y que en principio se calculaba un tiempo de computo del orden de los 200.000.000 años mediante una maquina con poder de procesamiento de 1 millón de operaciones por segundo y de unos 2000 años si se habría implementado la misma red de procesamiento sin ayuda de las técnicas modernas de optimización. Seguidamente, se han realizado algunos avances en el tema de la factorización prima de números grandes, en referencia a ello vale la pena traer a colación un caso interesante en expuesto en durante una conferencia en el "*Día Internacional de la Seguridad de la Información DISI*" realizada en el 2007, en la cual se pudo factorizar un

[4] El hecho que se haya factorizado el RSA-768 no quiere decir que las claves RSA no sirvan, sino que, si nosotros decidimos proteger una información con RSA-768, alguien podría acceder de manera ilícita a esa información en el plazo de tiempo que ha tardado el experimento (o incluso en menos si se cuenta con una mayor infraestructura de supercomputación).

numero de 640 bits (el mismo número dado en el reto RSA-640 resuelto en el 2005) en sólo algunos minutos y desde un simple notebook, un reto para el cual se había estimado un tiempo de cálculo de poco más de 30 años mediante una unidad de procesamiento de unos 2,2 GHz.

Así pues, aunque existen muchas maneras de vulnerar RSA sin recurrir a la factorización, se puede asegurar que el desarrollo de un método eficiente que permita lograr una descomposición prima de enteros largos en un tiempo polinomial, haría débil toda clave del RSA, pasando así a ser de este un sistema de cifrado obsoleto. En este orden de ideas, se han desarrollado algunos avances significativos en el campo de la computación, como lo han sido las computadoras cuánticas y que a su vez ha permitido la implementación de nuevas puertas lógicas para la realización de cálculos más complejos en tiempos mas eficientes, esto basándose en el principio de superposición del estado de una partícula, siendo así y a diferencia de un ordenador clásico (el cual solo permite operar sobre un conjunto de estados de bits {0,1} procesando solo un estado a la vez) un ordenador con estas características permitiría operar sobre una combinación de los estados {0,1} pudiendo procesar distintos valores al mismo tiempo, así pues mientras que en el paradigma clásico un bit solo puede ser 0 ó 1, un qbit (el cual es la versión cuántica del bit) puede tomar dos estados ortogonales a la vez como lo serian 0,1 y 1,0, por lo que si se dispone de un registro de 3 qbit entonces habrían 8 estados o valores distintos posibles que pueden ser tomados a la vez durante el computo gracias al principio de superposición, en contraparte un ordenador clásico contemplaría estos 8 estados, mas sin embargo el computo se haría sobre solo uno de los 8 bits a la vez; de esta manera según mayor sea el número de qbits disponibles se podría aumentar notablemente la capacidad de cálculos posibles en tiempos exponencialmente más cortos. Por tal motivo, el desarrollo de este tipo de máquinas implicaría la resolución en un tiempo eficiente de todo un conjunto de problemas que en un principio y desde el punto de vista de la computación clásica son intratables, como lo es el caso de la factorización prima de números grandes empleados por el RSA, o bien el problema de los logaritmos discretos, naturalmente este desarrollo seria paralelo al diseño

de nuevos algoritmos y programas que permitan su operación. Los primeros computadores de este tipo fueron desarrollados a principios del 2001 utilizando bases de 5 y 7 qbit, en el primer caso se puso a prueba el potencial de estas máquinas ejecutando a través de un simple paso un algoritmo de búsqueda de orden (implementado parcialmente en el algoritmo de Shor para la factorización de enteros y que veremos más adelante) y que en un ordenador clásico habría llevado una cantidad notable de iteraciones, en el segundo caso se implementó por primera vez un computador de 7 qbits para la factorización del número 15 a través de la ejecución del algoritmo de Shor, dando el resultado esperado. Hoy en día, se ha desarrollado computadores de este tipo de hasta 12 qbits, los cuales implementan un conjunto de 12 átomos para su funcionamiento, en efecto y para entender de manera más amena la importancia de estas investigaciones, considérese que eventualmente los computadores convencionales operan a razón de los 10^9 operaciones en coma flotante por segundo (gigaflops), un ordenador de solo 30 qbits trabajaría en el orden de los 10×10^{12} operaciones en coma flotante por segundo (es decir unos 10 teraflops), es decir ejecutarían alrededor de más de 10000 veces más operaciones en solo un segundo que cualquier ordenador convencional.

En conformidad a lo aquí planteado, el algoritmo de Shor, aprovecha las cualidades de un computador de este tipo y explota otros principios como los algoritmos de búsqueda, el principio de la superposición, y la serie de Fourier para lograr la factorización de cualquier entero largo en un tiempo de ejecución eficiente. En efecto, para iniciar con el estudio del algoritmo se procederá a introducir algunos aspectos fundamentales para la construcción y funcionamiento del mismo.

8.3.1 Algunos aspectos fundamentales

A diferencia de los algoritmos clásicos ejecutados en máquinas convencionales (basadas en un diseño de máquina de Turing determinista) y que operan sobre un conjunto de estados o bits {0,1}, este algoritmo opera a nivel de qbits (siendo este la unidad básica de

información cuántica del sistema y que bien se definirá posteriormente), los cuales representan los estados cuánticos posibles de la(s) partículas empleadas para la realización del cómputo. De esta forma y a diferencia de los estados {0,1} de los sistemas clásicos, la noción del qbit permite extender los estados que pueden presentarse en un sistema convencional. Para realizar tal tarea es necesario definir el qbit de manera consistente como la estructura más fundamental a implementar en el algoritmo, para lograr un mejor entendimiento es necesario introducir de previamente un espacio vectorial, en el cual estarán inmersos todos los estados posibles de la partícula, en efecto, el espacio más apropiado para trabajar es el *espacio n-dimensional de Hilbert* (*simbolizado por* $\mathcal{H}_n$), este es definido como un espacio vectorial n-dimensional sobre el cuerpo de los números complejos $\mathbb{C}$ dotado a su vez de un producto interno. Así pues, un estado físico de la partícula en el sistema será representado como un vector $\vec{v} \in \mathcal{H}_n$ para un espacio de Hilbert n-dimensional, o equivalentemente un estado físico también podrá ser representado por su conjugado Hermitiano (un ejemplo sencillo de espacio de Hilbert n-dimensional seria $\mathbb{C}_n$ considerando a su vez como producto interno el siguiente operador: $g = \langle\ ,\ \rangle$: $\mathcal{H}_n \to \mathbb{C}_n$ defindo por $\langle \vec{v}, \vec{w} \rangle = \sum_{i=1}^{n} v_i . \overline{w}_i$ siendo $\vec{v} = (v_1, v_2 \dots, v_n)$ y $\vec{w} = (w_1, w_2 \dots, w_n)$ con $\vec{v}, \vec{w} \in \mathbb{C}_n$ y siendo $\overline{w}_i$ el conjugado de w_i). Este espacio, tiene una base ortonormal canónica dada por el conjunto de vectores $\{e_1$=(1,0, …,0), e_2=(0,1,0,…,0),…, e_i=(0,0,…,1,…0), …, e_n=(0,0,…1)$\}$, es decir, es posible expresar cada vector $\vec{v} \in \mathcal{H}_n$ como una combinación lineal de dicha base (nótese además que todo espacio de Hilbert n-dimensional es isomorfo a $\mathbb{C}_n$, y en efecto es fácil ver que tal isomorfismo es proporcionado directamente por el producto interno que lo define). Para cualquier vector $\vec{v} \in \mathcal{H}_n$, es común representarlo mediante la notación de Dirac, en la cual se cada vector $\vec{v}$ es escrito como $|\ \vec{v}\ \rangle$ denotándose este como el *vector ket* de $\vec{v}$, así mismo se denota el vector adjunto o hermitico de $\vec{v}$ (esto es $\vec{v}^* = (\vec{v})^t$, es decir la transpuesta del vector conjugado) como el *vector bra* de $\vec{v}$, denotándose este como $\langle\ \vec{v}\ |$. Así pues, con esta convención podrá escribirse el producto interno (el cual está definido naturalmente por $\langle \vec{v}, \vec{w} \rangle = \vec{w}^* . \vec{v}$) como $\langle \vec{v}\ |\ \vec{w} \rangle$. De acuerdo a esta notación, podrá también escribirse la base canónica de $\mathbb{C}_n$, como:

$\{ | 0 \rangle, | 1 \rangle, | 2 \rangle, .., | i \rangle, \ldots, | n-1 \rangle \}$

Donde

$$| 0 \rangle = \begin{pmatrix} 1 \\ 0 \\ . \\ . \\ . \\ 0 \\ . \\ . \\ 0 \end{pmatrix}, \; | 1 \rangle = \begin{pmatrix} 0 \\ 1 \\ . \\ . \\ . \\ 0 \\ . \\ . \\ 0 \end{pmatrix}, \; | 2 \rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ . \\ . \\ 0 \\ . \\ . \\ 0 \end{pmatrix}, \ldots., \; | i \rangle = \begin{pmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 1 \\ . \\ . \\ 0 \end{pmatrix}, \ldots, |n-1\rangle = \begin{pmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 0 \\ . \\ . \\ 1 \end{pmatrix}$$

Es decir cada vector ket dado por $| i \rangle$, está representado canónicamente como un vector columna con un 1 en la (i-esima + 1) fila (o posición) y siendo 0 en el resto de las posiciones, para $i = 0,1,2, \ldots, n-1$. De igual forma también podrá escribirse la base canónica de $\mathbb{C}_n$, como la base dada por:

$\{ \langle 0 |, \langle 1 |, \langle 2 |, \ldots, \langle i |, \ldots, \langle n-1 | \}$

Dónde:

$\langle 0 | = (1, 0,0,0 \ldots, 0, \ldots, 0)$

$\langle 1 | = (0,1,0,0, \ldots, 0, \ldots, 0)$

$\langle 2 | = (0,0,1,0, \ldots, 0, \ldots, 0)$

.

.

$\langle i | = (0,0,0,0, \ldots, 1, \ldots, 0)$

.

.

$\langle n-1| = (0,0, \ldots, 0, \ldots, 1)$

Es decir cada vector bra dado por $\langle i|$ está representado canónicamente por un vector fila con un 1 en la (i-esima + 1) columna (o posición) y siendo 0 en el resto de las posiciones,

para $i = 0,1,2,\dots,n-1$. Así pues, $\langle\, i \mid j \rangle = \delta_{ij}$ con $\delta_{ij} = 1$ si $i = j$ y $\delta_{ij} = 0$ si $i \neq j$, de esta manera un vector ket cualquiera del espacio de Hilbert $|\, \psi \,\rangle$ podrá ser expresado como una combinación lineal de la base ortonormal de vectores ket mediante la siguiente manera $|\, \psi \,\rangle = \alpha_0 |\, 0 \,\rangle + \alpha_1 |\, 1 \,\rangle + \alpha_2 |\, 2 \,\rangle + \cdots + \alpha_i |\, i \,\rangle + \cdots + \alpha_{n-1} |\, n-1 \,\rangle$ y $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}$ complejos, siendo cada uno de estos la *i-esima* coordenada del vector $|\, \psi \,\rangle$ con respecto a su base canónica. De esta manera si, $n = 2$ tendríamos que un vector ket $|\, \psi \,\rangle$ del espacio $\mathcal{H}_2$ podrá ser expresado como una combinación lineal de la base canonica $\{|\, 0 \,\rangle, |\, 1 \,\rangle\}$ como $|\, \psi \,\rangle = \alpha_0 |\, 0 \,\rangle + \alpha_1 |\, 1 \,\rangle$. El vector $|\, \psi \,\rangle$ como en cualquier espacio vectorial puede expresarse mediante diferentes bases ortonormales o no, así pues si se considerara para $n = 2$ una base de vectores kets ortonormales dada por $\{|\, x \,\rangle, |\, y \,\rangle\}$, definiendo los vectores de dicha base como $|\, x \,\rangle = \frac{1}{\sqrt{2}}(|\, 0 \,\rangle + |\, 1 \,\rangle)$ y $|\, y \,\rangle = \frac{1}{\sqrt{2}}(|\, 0 \,\rangle - |\, 1 \,\rangle)$, se tendría que el vector $|\, \psi \,\rangle$ podrá ser expresado como una combinación lineal de los vectores ket de dicha base de la siguiente manera: $|\, \psi \,\rangle = \frac{1}{\sqrt{2}}(\alpha_0 + \alpha_1).|\, x \,\rangle + \frac{1}{\sqrt{2}}(\alpha_0 - \alpha_1).|\, y \,\rangle$, siendo los escalares $\frac{1}{\sqrt{2}}(\alpha_0 + \alpha_1)$ y $\frac{1}{\sqrt{2}}(\alpha_0 - \alpha_1)$ la primera y segunda coordenada del vector $|\, \psi \,\rangle$ con respecto a la base $\{|\, x \,\rangle, |\, y \,\rangle\}$. Nótese que se podría hacer exactamente los mismos procedimientos con los vectores bra de la base canónica y expresar $\langle\, \psi \,|$ como una combinación lineal de los vectores bra dados por la base $\{\langle\, x|, \langle\, y|\}$ definiéndose esta vez $\langle\, x \,| = \frac{1}{\sqrt{2}}(\langle\, 0 \,| + \langle\, 1 \,|)$ y $\langle\, x \,| = \frac{1}{\sqrt{2}}(\langle\, 0 \,| - \langle\, 1 \,|)$. De manera más general, se puede establecer o asociar a cada vector ket $|\, \psi \,\rangle$ con su dual, es decir con la transpuesta del vector conjugado, lo que sería equivalente a asociar el ket $|\, \psi \,\rangle$ con su bra $\langle\, \psi \,|$, de la siguiente manera: $|\, \psi \,\rangle = (\langle\, \psi \,|)^*$ y $\langle\, \psi \,| = (|\, \psi \,\rangle)^*$, esto permite representar el vector bra dado por $\langle\, \psi \,|$ como una combinación lineal de la base canónica de vectores bra con los conjugados de las coordenadas del vector $|\, \psi \,\rangle$, es decir, podrá expresarse el vector bra a través de $\langle\, \psi \,| = \bar{\alpha}_0 \langle\, 0 \,| + \bar{\alpha}_1 \langle\, 1 \,| + \cdots + \bar{\alpha}_i \langle\, i \,| + \cdots + \bar{\alpha}_{n-1} \langle\, n-1 \,|$ donde cada $\bar{\alpha}_0, \bar{\alpha}_1, \dots, \bar{\alpha}_{i,} \dots, \bar{\alpha}_{n-1}$ son los conjugados complejos de $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}$ las coordenadas *i-esimas* del vector ket en la base canónica. Por lo que podrá escribirse los vectores $\langle\, \psi \,|$ y $|\, \psi \,\rangle$ en forma matricial como: $\langle\, \psi \,| = (\bar{\alpha}_0, \bar{\alpha}_1, \dots, \bar{\alpha}_{i,} \dots, \bar{\alpha}_{n-1})$ es decir como una matriz

de $1 \times n$ de los conjugados complejos, y respecto a la base canónica formada por vectores bra.

Mientras que $|\,\psi\,\rangle$ se podrá expresar como una matriz de $n \times 1$ de los elementos complejos $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}$ respecto a la base canónica de vectores ket, expresándose entonces como:

$$|\,\psi\,\rangle = \left(\bar{\alpha}_0, \bar{\alpha}_1, \ldots, \bar{\alpha}_{i,} \ldots, \bar{\alpha}_{n-1}\right)$$

Así pues en conformidad a esta notación el producto interno de un vector $|\,\psi\,\rangle$ consigo mismo, puede ser descrito como:

$$\langle\psi\,|\,\psi\rangle = \left(\bar{\alpha}_0, \bar{\alpha}_1, \ldots, \bar{\alpha}_{i,} \ldots, \bar{\alpha}_{n-1}\right) \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ . \\ . \\ . \\ \alpha_i \\ . \\ . \\ \alpha_{n-1} \end{pmatrix} = |\alpha_0|^2 + |\alpha_1|^2 + \cdots + |\alpha_{n-1}|^2$$

Donde $|\alpha_i|^2$ es la suma del cuadrado de la parte real más el cuadrado de la parte imaginaria del complejo α_i, más exactamente esto es: $|\alpha_i|^2 = \bar{\alpha}_i . \alpha_i = [\Re e(\alpha_i)]^2 + [\Im m(\alpha_i)]^2$.

Por último, y con el fin de complementar las ideas anteriores, y que permitirá no solo conceptualizar de una manera más exacta el qbit como unidad elemental para el algoritmo de Shor, sino también que permitirá operar los estados individuales de los mismos para obtener el estado conjunto (o estado compuesto) de un sistema formado por *N* qbits, se procederá a establecer un operador entre matrices llamado *producto tensorial*, denotado por $\otimes$, en efecto sean A y B dos matrices de $m \times n$ y $p \times q$ respectivamente, dadas por:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \quad \text{y} \quad B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}$$

Se define el producto tensorial de A $\otimes$ B, como la matriz de tamaño $mp \times nq$ dada por:

$$A \otimes \mathrm{B} = \begin{pmatrix} a_{11}\mathrm{B} & a_{12}\mathrm{B} & \cdots & a_{1n}\mathrm{B} \\ a_{21}\mathrm{B} & a_{22}\mathrm{B} & \cdots & a_{2n}\mathrm{B} \\ a_{31}\mathrm{B} & a_{32}\mathrm{B} & \cdots & a_{3n}\mathrm{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathrm{B} & a_{m2}\mathrm{B} & \cdots & a_{mn}\mathrm{B} \end{pmatrix}$$

Donde, los elementos a_{ij}B, con $1 \leq i \leq m$ y $1 \leq j \leq n$ son una submatriz cuyas entradas son el producto de a_{ij} con todos los elementos de la matriz B. A manera de ejemplo, si se considera el producto tensorial de los vectores $|0\rangle$ y $|1\rangle$ de $\mathcal{H}_2$ se tendría que este estaría dado por una matriz de columna de 4×1 de la siguiente forma:

$$|0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.\begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0.\begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Una vez presentado toda la estructuración anterior, se procede a definir la unidad elemental de operación del algoritmo de Shor.

Definición (Qbit): Se define un qbit, como la unidad básica de información, el cual puede ser representado como un vector $|\psi\rangle \in \mathcal{H}_2$ del espacio 2-dimensional de Hilbert, expresándose este como una combinación lineal (o superposición) de los vectores de la base canónica $|0\rangle$ y $|1\rangle$ dada por $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, donde $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Siendo $|\alpha_0|^2$ y $|\alpha_1|^2$ las probabilidades de que dicho qbit ocupe el estado $|0\rangle$ y $|1\rangle$ respectivamente (es decir, los elementos de $\alpha_0, \alpha_1 \in \mathbb{C}$ son contenedores de información de fase).

Una vez presentada esta definición, se podrá extender este concepto a sistemas con más de un qbits. En efecto, considérese un sistema compuesto por *n* qbits, donde cada uno de los cuales puede tener dos estados, siendo así y a diferencia de sistemas de los bits clásicos implementados en máquinas deterministas (en donde el estado de dicho sistema puede ser descrito de manera completa por *n* estados) la descripción completa del sistema de *n* qbits se realizaría con $2^n - 1$ números complejos, de manera más precisa el estado de dicho sistema es representado por un vector en el espacio 2^ndimensional de Hilbert ($\mathcal{H}_{2^n}$) sobre el campo de los complejos, con una base de 2^n vectores de estados $|\, e_i \,\rangle$ con $i = 1,2, \dots, 2^n$, así pues si $|\, \psi \,\rangle$ representa el estado de dicho sistema, entonces $|\, \psi \,\rangle = \sum_{i=1}^{2^n} (\alpha_i |\, e_i \,\rangle)$, donde las amplitudes α_i satisfacen $\sum_{i=1}^{2^n} |\alpha_i|^2 = 1$ siendo $|\alpha_i|^2$ la probabilidad de observarse (o medir) el vector de estado $|\, e_i \,\rangle$. Así pues podremos caracterizar cada estado $|\, \psi \,\rangle$ de cualquier qbit o sistema, con dos vectores 2^n-dimensionales: el vector de amplitud de probabilidad (el cual es un vector complejo), y la probabilidad de estado (el cual es un vector de números reales), los cuales estarían definidos por $(\alpha_1, \alpha_2 \dots, \alpha_{2^n})$ y $(|\alpha_1|^2, |\alpha_2|^2, \dots |\alpha_{2^n}|^2)$ respectivamente.

Por otro lado y antes de discutir el algoritmo de Shor, se procederá a introducir otro concepto, siendo este quizás el de mayor importancia para el desarrollo de dicho algoritmo, se trata de la *trasformada de Fourier* la cual permitirá como se verá más adelante a encontrar el periodo de una cierta función (la cual como veremos consiste en una ecuación exponencial modular) para lograr de esta manera una factorización prima eficiente de un entero largo, este sentido la transformada de Fourier puede aplicarse en espacios de dimensiones diversas e incluso de dimensión infinita, sin embargo y para el propósito que aquí se presenta se introducirá una definición de la transformada de Fourier sobre el espacio de Hilbert 2^n dimensional, lo cual al ser una potencia de 2 simplificara un poco las operaciones realizadas dentro de la misma y para mayor conveniencia del algoritmo de Shor que se presentara posteriormente.

Transformada Cuántica de Fourier: Se define como transformada cuántica de Fourier al operador lineal que actúa sobre los vectores de la base ortonormal $\{ | 0 \rangle, | 1 \rangle, | 2 \rangle, .., | j \rangle, \ldots, | k \rangle, \ldots, | N-1 \rangle \}$ del espacio vectorial $\mathcal{H}_N$, con $N = 2^n$ transformándolos de la siguiente manera: $| j \rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{i2\pi jk/N} | k \rangle$ con $i = \sqrt{-1}$. Transformando así un estado $| \psi \rangle = \sum_{j=0}^{N-1} \alpha_j | j \rangle \rightarrow | \varphi \rangle = \sum_{k=0}^{N-1} \beta_k | k \rangle$ siendo cada amplitud de probabilidad β_k la transformada de Fourier discreta de la amplitud de probabilidad α_j, $0 \leq j \leq N-1$, con $\beta_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \alpha_j e^{i2\pi jk/N}$.

Para efectos prácticos, obsérvese que al tomar la representación binaria de los respectivos enteros j y k formadas por: $j = j_0 2^{n-1} + j_1 2^{n-2} + j_2 2^{n-3} + \cdots + j_{n-2} 2^1 + j_{n-1} 2^0$ y $k = k_0 2^{n-1} + k_1 2^{n-2} + k_2 2^{n-3} + \cdots + k_{n-2} 2^1 + k_{n-1} 2^0$, entonces la transformada de Fourier puede ser reescrita como:

$$| j_0 j_1 \ldots j_{n-1} \rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k_0=(0,1)} \sum_{k_1=(0,1)} \ldots \sum_{k_{n-1}=(0,1)} e^{i2\pi j \sum_{m=0}^{n-1} k_m . 2^{-m}} | k_0 k_1 \ldots k_m .. k_{n-1} \rangle$$

Luego como $| k_0 k_1 \ldots k_m .. k_{n-1} \rangle = |k_0\rangle \otimes |k_1 \rangle \otimes \ldots \otimes |k_{n-1} \rangle$ entonces la transformada puede ser expresada de la siguiente manera:

$$| j_0 j_1 \ldots j_{n-1} \rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k_0=(0,1)} \sum_{k_1=(0,1)} \ldots \sum_{k_{n-1}=(0,1)} \bigotimes_{m=0}^{n-1} e^{i2\pi j k_m 2^{-m}} | k_m \rangle$$

Esto es:

$$| j_0 j_1 \ldots j_{n-1} \rangle \rightarrow \frac{1}{\sqrt{2^n}} \bigotimes_{m=0}^{n-1} \{ \sum_{k_m=(0,1)} e^{i2\pi j k_m 2^{-m}} | k_m \rangle \}$$

Donde el bit k_m puede tomar solo los valores {0,1}. Luego puesto que $e^{i2\pi j k_m} = 1$ si $k_m = 0$, entonces la transformada de Fourier se reduce a:

$$| j_0 j_1 \dots j_{n-1} \rangle \rightarrow \frac{1}{\sqrt{2^n}} \otimes_{m=0}^{n-1} \{| 0\rangle + e^{i2\pi j\, 2^{-m}} . | 1\rangle\}$$

Pero $| 0\rangle + e^{i2\pi j\, 2^{-m}} . | 1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ e^{i2\pi j\, 2^{-m}} \end{pmatrix} = \begin{pmatrix} 1 \\ e^{i2\pi\,(j\,/2^m)} \end{pmatrix}$. Así pues, la transformada podra ser escrita como:

$$| j \rangle = | j_0 j_1 \dots j_{n-1} \rangle \rightarrow \frac{1}{\sqrt{2^n}} \begin{pmatrix} 1 \\ e^{i2\pi\,(j\,/2^0)} \end{pmatrix} \otimes \begin{pmatrix} 1 \\ e^{i2\pi\,(j\,/2^1)} \end{pmatrix} \otimes \dots \otimes \begin{pmatrix} 1 \\ e^{i2\pi\,(j\,/2^{n-1})} \end{pmatrix}$$

Obsérvese que los vectores $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ y $\begin{pmatrix} 0 \\ e^{2\pi i\,/2^m} \end{pmatrix}$, forman una matriz de 2×2 dada por $R_m = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i\,/2^m} \end{pmatrix}$ la cual es una matriz que representa una transformación unitaria y reversible, asociando dicha matriz a una puerta lógica controlada para el desplazamiento de fase dentro del circuito de cómputo para la transformada de Fourier.

Habiendo presentado los aspectos fundamentales como el paralelismo (aquella propiedad de los sistemas cuánticos para representar los estados 0 y 1 o incluso varios estados simultáneamente), la noción de qbit, y la transformada de Fourier, se procede a detallar el algoritmo de factorización prima de Shor.

8.3.2. El algoritmo de Shor como solución al problema

El algoritmo que se presenta en este apartado basa su eficiencia computacional en las propiedades intrínsecas de las partículas, en este sentido el algoritmo de Shor intenta encontrar un divisor *p* (primo) de un entero largo *N*. Como se verá más adelante, el algoritmo reduce esta tarea a un algoritmo de búsqueda de un factor capaz de satisfacer

cierta ecuación exponencial modular. Para ello, el algoritmo de Shor se estructura esencialmente en dos partes: la primera se consiste en reducir el problema de la descomposición prima en encontrar el orden asociado a una función exponencial modular; la segunda parte se enfoca en implementar los conceptos previamente expuestos a través de un subprograma, dando especial importancia a la transformada de Fourier para hallar el periodo relacionado a la ecuación modular exponencial planteada en la primera parte. Sea *n* un entero largo a factorizar, el algoritmo a Shor se estructura de la siguiente manera siguiente:

Primera Parte: Reducción a búsqueda del orden

1. Inicio: Selecciónese un número entero $1 < a < n - 1$ aleatorio.

2. Calcúlese el $mcd(a, n)$, (lo cual puede realizarse de manera eficiente en un tiempo polinomico de mediante el algoritmo extendido de Euclides).

2.1. Si $mcd(a, n) > 1$, entonces este es un factor no trivial de *n*, por lo tanto, devuelva como factor $mcd(a, n)$.

2.2. Sino, entonces salte al ***subprograma*** (segunda parte) y hállese el periodo r de la función dada por: $a^x (mod\ n) = f(x)$. (es decir, es el menor entero r tal que $f(x + r) = f(x)$ o lo que bien seria equivalente a hallar el orden de $a\ (mod\ n)$, esto es el menor entero positivo r tal que $a^r \equiv 1 (mod\ n)$).

3. Si $2 \nmid r$ entonces regrese al paso 1.

4. Si $a^{r/2} \equiv \pm 1 (mod\ n)$ entonces regrese al paso 1.

5. Calcule $mcd(a^{r/2} \pm 1, n)$ mediante el algoritmo de Euclides extendido y pruebe si son factores, si es así devuélvalos como factores de n. Sino regrese al paso 1. Fin

Segunda Parte: Subprograma (Cálculo del orden mediante el periodo)

1. Se procede a iniciar con un par de qbits o registros (entrada y salida), obtiene poniendo el primer registro en una superposición de estados uniformes que represente el número $x(mod\ q)$, con q potencia de 2, donde $n^2 < q = 2^n < 2n^2$, (Este paso es relativamente fácil, ya que todo lo que requiere es poner cada bit en el primer registro en la superposición $\frac{1}{\sqrt{q}}(|\ 0\rangle + |\ 1\rangle)$, lo cual se puede hacer implementado la transformación de Hadamard[5]), esto dejaría la maquina en el estado

$$\frac{1}{\sqrt{q}}\sum_{j=0}^{q-1}|\ j\rangle|\ 0\rangle$$

2. Se procede a construir la función $f(x)$ como función cuántica e impleméntese en el estado anterior, es decir se computa $a^x(mod\ n)$ en el segundo registro que se dispone y se obtiene una superposición de los 2^n valores de la función $f(x)$ para cada uno de los posibles 2^n valores que pueda tener el argumento x. Obteniéndose como resultado:

$$\frac{1}{\sqrt{q}}\sum_{j=0}^{q-1}|\ j\rangle|\ f(j)\rangle\ \ es\ decir\ \frac{1}{\sqrt{q}}\sum_{j=0}^{q-1}|\ j\rangle\left|a^j(mod\ n)\ \right\rangle$$

3. Se aplica la transformada cuántica de Fourier al primer registro, obteniéndose el siguiente estado de maquina:

$$\frac{1}{q}\sum_{j=0}^{q-1}\sum_{k=0}^{q-1}e^{i2\pi jk/q}\ |\ k\rangle|\ f(j)\rangle\ \ osea\ \frac{1}{q}\sum_{j=0}^{q-1}\sum_{k=0}^{q-1}e^{i2\pi jk/q}\ |\ k\rangle\left|a^j(mod\ n)\right\rangle$$

4. Realice una observación del estado de máquina, es decir una medición sobre los estados $|\ k\rangle$ y $\left|a^j(mod\ n)\right\rangle$, para $0 \le j \le q-1$. Este paso produce una salida

[5] La transformada de Hadamard es un caso particular de la transformada discreta de Fourier.

concreta dada por $|k\rangle|a^m(mod\ n)\rangle$ con $0 \leq m < r$. Obteniéndose fácilmente como probabilidad de medir dicho estado el resultado siguiente:

$$P(|k\rangle|a^m(mod\ n)\rangle) = \left| \frac{1}{q} \sum_{j:\, a^j \equiv a^m (mod\ n)} e^{i2\pi jk/q} \right|^2$$

Luego como r es el orden buscado de la congruencia y puesto que podemos escribir j como $j = b.r + m$, esta probabilidad puede ser reescrita así:

$$P(|k\rangle|a^m(mod\ n)\rangle) = \left| \frac{1}{q} \sum_{b=0}^{(q-m-1)/r} e^{i2\pi(b.r+m)k/q} \right|^2$$

5. Transfórmese k/q a una fracción irreducible, y extráigase el denominador resultante r' el cual será primer candidato a ser el periodo r.

6. Evalúese si $f(x + r') = f(x)$, de ser cierto retorne como factor $mcd(a^{r'/2} \pm 1, n)$, fin del programa.

7. Si al evaluar r' se tiene que $f(x + r') \neq f(x)$, entonces obténgase valores más próximos a k o múltiplos de r' para obtener más candidatos a r, una vez se tenga estos nuevos candidatos r'' proceda a evaluar nuevamente si $f(x + r'') = f(x)$, de ser así retorne como factor $mcd(a^{r''/2} \pm 1, n)$ y fin del programa.

8. Si el paso (7) falla, regrese al paso (1) correspondiente al programa central *Reducción a búsqueda del orden.*

8.3.3. Observaciones generales y complejidad algorítmica

Como se puede apreciar, el algoritmo consta básicamente de dos partes, la primera puede ser ejecutada en un computador clásico, esta primera parte se encarga de reducir el problema a la búsqueda de un elemento r por el cual se pueda calcular posteriormente los factores de n dados por $mcd(a^{r/2} \pm 1, n)$. En efecto y para ver esto más detalladamente, obsérvese que el paso (3) de la primera parte del algoritmo, se restringe el valor del orden de la ecuación en congruencias $a^r \equiv 1(mod\, n)$) a un r par, ahora para garantizar que el $mcd(a^{r/2} \pm 1, n)$ es un factor de n, detallemos el siguiente teorema:

Teorema 3: Supóngase que n es un número compuesto con $\log_2 n$ bits de longitud. Sea r el menor entero par tal que $a^r \equiv 1(mod\, n)$ para algún $0 \leq a \leq n$ (esto es el orden de a $(mod\, n)$). Entonces $mcd(a^{r/2} \pm 1, n)$ existe y además es un factor no trivial de n, teniendo esto una complejidad de $O((\log n)^3)$.

Prueba: Sea $a^r \equiv 1(mod\, n)$, esto es $a^r - 1 = k.n$ para un cierto $k \in \mathbb{N}$, siendo así entonces $a^r - 1 \equiv 0(mod\, n)$ y ademas se tendría que $n|(a^r - 1)$, luego pues que r es par, se tendría que $a^r - 1$ es reducible a $(a^{r/2} + 1)(a^{r/2} - 1)$, luego $n|(a^{r/2} + 1)(a^{r/2} - 1)$, seguidamente como por hipótesis r es el menor entero tal que $a^r \equiv 1(mod\, n)$, entonces $n \nmid (a^{r/2} - 1)$, (pues de lo contrario $r/2$ sería un entero tal que $a^{r/2} \equiv 1(mod\, n)$, lo cual iría en contra de la hipótesis de que r es el menor entero con tal característica), es decir $a^{r/2} \not\equiv 1(mod\, n)$. De igual forma se tiene que $n \nmid (a^{r/2} + 1)$, esto es $a^{r/2} \not\equiv -1(mod\, n)$. Siendo así entonces, n debe tener un factor común entre $(a^{r/2} - 1)$ y $(a^{r/2} + 1)$, por lo tanto $mcd(a^{r/2} \pm 1, n)$ existe y si n es un producto de dos primos (tal como se maneja en el cifrado RSA) entonces dicho número es un factor no trivial de n. Seguidamente nótese que la tarea de computar $mcd(a^{r/2} - 1, n)$ o $mcd(a^{r/2} + 1, n)$ puede ser resuelta aplicando el algoritmo de Euclides, en este sentido cada división Euclidea se realizaría en $O((\log n)^2)$ pasos, siendo asi estos pasos tendrían que realizarse en el peor caso unas

$O(\log n)$ veces, por lo cual aplicar el algoritmo de Euclides para calcular $mcd(a^{r/2}-1, n)$ una vez teniendo r, tomaría un tiempo de computo dado por $O((\log n)^3)$. ■

Así pues, desde los pasos (2) hasta el (5) de la primera parte del algoritmo, tomaría un total de $O((\log n)^3)$ una vez pueda calcularse eficientemente el orden de la ecuación exponencial modular.

Por otro lado nótese que la parte que contiene el subprograma, es la parte fundamental para la resolución del problema de la factorización prima en tiempo de ejecución eficiente, esta segunda parte basa su rapidez en el *paralelismo cuántico*[6] para la obtención del factor r que satisfaga la ecuación exponencial en congruencias dada por $a^r \equiv 1(mod\, n)$ considerandose a r como el menor entero con dicha propiedad. En este sentido, toma ventaja del *paralelismo* para crear en un solo paso una superposición de todos los valores de estados posibles de una función y realizar el cómputo, para ello se inicia expresando la entrada como una combinación lineal de los vectores bases $\{|\,0\rangle, |\,1\rangle, \ldots, |\,q-1\rangle\}$ tal como se realizó en el paso (1) del subprograma, de esta forma el paralelismo a implica a su vez un numero de puertas lógicas representadas mediante las operaciones sobre qbits y que pueden ser activadas instantáneamente en paralelo para la resolución del problema de la factorización, que en principio sería un problema intratable desde la óptica dela computación clásica. En efecto, para ilustrar un poco las ventajas de este principio en la resolución de problemas complejos, consideremos una función cualquiera que mapee valores del conjunto $\{0,1\}^n$ en $\{0,1\}^m$ (pudiendo ser $m \geq n$), esto es $f: \{0,1\}^n \to \{0,1\}^m$, piénsese en el problema de evaluar los valores de la función $f(x)$ para todos los argumentos posibles $x \in \{0,1\}^n$ con $n = 100$, en este sentido un algoritmo ejecutado en un ordenador clásico, necesitaría realizar un total de unas $2^{100} \approx 10^{33}$ operaciones o pasos a ejecutar, mientras que si se implementara el paralelismo cuántico, se podría crear una superposición que registre todos los valores posibles de la función, realizando así esta misma tarea que en un solo paso.

[6] Propiedad física intrínseca de las partículas atómicas que permiten realizan simultáneamente varias operaciones simultáneamente y sobre todas las combinaciones de una cierta entrada.

En este orden de ideas obsérvese también la gran importancia que juega para la segunda parte la implementación de la transformada de Fourier para la búsqueda del periodo (paso 3 del subprograma), por lo cual el orden de complejidad del subprograma utilizado radica básicamente en la complejidad asociada a la implementación de dicha transformada. En efecto, al aplicarla el problema es sintetizado solamente a encontrar una estimación $\omega = k/q$ tal que $e^{i2\pi j\omega}$ sea un valor propio y correspondiente a $|\, k\rangle$, la cual se contempla en el paso (5) del subprograma, seguidamente para elaborar tal estimación de k/q (en su mínima expresión) esta podría esperarse que incluso coincida con r en una sola ejecución, deberá tenerse en cuenta que la mejor estimación que ha satisfacer $\left|\frac{k}{q} - \frac{d'}{r'}\right| \leq \frac{1}{2q}$ siendo $r' < n < \sqrt{q}$, es decir la sucesión de valores dadas por k/q debe ser convergente a un d'/r' fijo, esto significa que dado un a asociado a un valor observado $|\, k\rangle|a^m(mod\ n)\rangle$ con $0 \leq m < r$ como el que se da en el punto (4) del subprograma, siempre se podrá encontrar un valor correcto de r con probabilidad cercana a 1. Para ver este hecho, es preciso definir previamente lo que es un *buen valor de k*, en este orden de ideas diremos que k es un buen valor si existe $\ell\epsilon\left[-\frac{r}{2}, \frac{r}{2}\right]$ tal que $rk \equiv \ell(mod\ q)$, siendo así entonces en existe un d tal que $-\frac{r}{2} \leq rk + dq = \ell \leq \frac{r}{2}$, luego al multiplicar por $\frac{1}{qr}$ se tiene que $-\frac{1}{2q} \leq \frac{k}{q} + \frac{d}{r} \leq \frac{1}{2q}$ es decir: $\left|\frac{k}{q} - \frac{d}{r}\right| \leq \frac{1}{2q}$, lo cual era lo que se quería ver. Así pues, si k es un buen valor entonces el r' hallado y que satisfaga $\left|\frac{k}{q} - \frac{d'}{r'}\right| \leq \frac{1}{2q}$ siempre dividirá a r, seguidamente la probabilidad de obtener ese buen valor de k puede ser obtenida directamente a través de la ecuación dada en el paso (4) del subprograma, es decir la probabilidad de obtener un buen k se puede obtener mediante

$$P(buen\ valor\ de\ k) = P(|\, k\rangle|a^m(mod\ n))) = \left|\frac{1}{q}\sum_{b=0}^{(q-m-1)/r} e^{i2\pi(b.r+m)k/q}\right|^2$$

Seguidamente nótese, que como $r' < n < \sqrt{q}$, entonces $(r')^2 < q$, por lo que podríamos reescribir $\left|\frac{k}{q} - \frac{d'}{r'}\right| \leq \frac{1}{2q}$ como $\left|\frac{k}{r} - \frac{d'}{r'}\right| \leq \frac{1}{2(r')^2}$, luego desde que $\frac{k}{r}$ sea un número racional y que $\frac{d}{r}$ sea una aproximación a $\frac{k}{r}$, entonces podrá implementarse iterativamente el cálculo de fracciones continuas para obtener explícita y eficientemente un par de enteros primos relativos tales que d' y r' tales que $\frac{d'}{r'} = \frac{k}{r}$, de esta manera el número r' se convertiría en nuestro candidato para el orden de la ecuación exponencial modular tratada en el algoritmo de Shor. En este orden de ideas, sería entonces de gran interés calcular la complejidad algorítmica que requeriría desarrollar el cálculo de estas fracciones continuas extendidas, en efecto para determinar tal complejidad se anunciará previamente el siguiente teorema:

Teorema 4: Sea $a_0, a_1, \ldots, a_N$ una secuencia de números positivos. Entonces $[a_o, a_1, \ldots, a_n] = k_n/r_n$ donde k_n y r_n son números reales definidos inductivamente por $k_0 = a_0$, $r_1 = 1$ y $k_1 = 1 + a_0 a_1$, $r_1 = a_1$, mientras que para $2 \leq n \leq N$ se define $k_n = a_n k_{n-1} + k_{n-2}$ y $r_n = a_n r_{n-1} + r_{n-2}$. Siendo k_j y r_j enteros positivos siempre y cuando a_j también lo sea.

Siendo asi entonces, para determinar la complejidad asociada a ejecutar la fraccion continua extendida que nos daran el mejor candidato r del orden buscado, se tendria que resolver la siguiente inquietud ¿Cuántos valores de a_n han de ser determinados para obtener una expansión de fracción continua para un número racional $z = \frac{k}{r} > 1$ donde k y r son primos relativos? En efecto, supóngase que $a_0, a_1, \ldots, a_N$ son enteros positivos, entonces de la definición de k_n y r_n dada en el teorema 4 se sigue que, k_n y r_n son secuencias crecientes. Por lo cual se tendría entonces que $k_n = a_n k_{n-1} + k_{n-2} \geq 2k_{n-2}$ análogamente para r_n se tiene que $r_n = a_n r_{n-1} + r_{n-2} \geq 2r_{n-2}$, luego $k_n \geq 2^{n/2}$, además también se tendría que $r_n \geq 2^{n/2}$. Así pues $p \geq q \geq 2^{N/2}$, y además $N = O(\log_2 p)$. De aquí se sigue que si $z = \frac{k}{r}$ es un numero racional, y si k y r son enteros de L bits de tamaño, entonces la fracción continua extendida para el cálculo de r seria computada usando un

total de $O(L^3)$ pasos, es decir $O((\log_2 k)^3)$, esto teniendo en cuenta que se requieren un total de $O(\log_2 k)$ operaciones elementales para dividir e invertir los elementos que integran el proceso, y considerándose que además cada uno de estos pasos implementa un total de $O((\log_2 k)^2)$ operaciones aritméticas, obteniéndose así la complejidad de esta pequeña parte del algoritmo de Shor.

Por otro lado, para calcular la complejidad de algoritmo de Shor, es importante a su vez determinar la complejidad computacional que requiere implementar la transformada de Fourier (o bien la inversa de esta, la cual en principio ha de tener la misma complejidad que aplicar dicha transformada), en efecto si se tiene en cuenta la expresión binaria, para cada estado de un qbit $|\, x\,\rangle$ cualquiera, tendríamos que este x puede ser representado como $x = x_0 2^{n-1} + x_1 2^{n-2} + x_2 2^{n-3} + \cdots + x_{n-2} 2^1 + x_{n-1} 2^0$, donde cada $x_j \in \{0,1\}$ para cada $j = 0,1, \dots, n-1$, asi pues la base de todos los posibles estados del qbit está dada por $|\, x\,\rangle = |\, x_0 x_1 \dots x_{n-1}\rangle = |\, x_0\,\rangle \otimes |\, x_1\,\rangle \otimes \dots \otimes |\, x_{n-1}\,\rangle$ además, la aplicar la transformada de Fourier, sobre el estado $|\, x\,\rangle$ nos quedaría que:

$$|\, x\,\rangle = |\, x_0 x_1 \dots x_{n-1}\,\rangle \rightarrow \frac{1}{\sqrt{2^n}} \bigotimes_{m=0}^{n-1} \{|\, 0\rangle + e^{i2\pi x\, 2^{-m}} .|\, 1\rangle\} \qquad \text{(Expresión. A)}$$

Donde la superposición del estado $|\, 0\rangle$ y $e^{i2\pi x\, 2^{-0}} .|\, 1\rangle$ corresponden a la salida del qbit 1, la superposición del estado $|\, 0\rangle$ y $e^{i2\pi x\, 2^{-1}} .|\, 1\rangle$ corresponden a la salida del qbit 2, y así sucesivamente. Ahora bien, nótese que en la expresión A, el primer término dado por $|\, 0\rangle + e^{i2\pi x\, 2^{-0}} .|\, 1\rangle$ requiere solamente una puerta de Hadamard[7], mientras que el segundo término dado por $|\, 0\rangle + e^{i2\pi x\, 2^{-1}} .|\, 1\rangle$, requiere una puerta de Hadamard y una puerta controlada para el desplazamiento de fase, es decir una puerta lógica asociada directamente por la matriz $R_m = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i\, /2^m} \end{pmatrix}$ establecida anteriormente cuando se trató la

[7] Es definida como una puerta lógica que opera sobre un único qbit, y que es representada mediante la matriz $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, la cual aplica una rotación de π sobre los ejes coordenados x y z.

transformada de Fourier, si continuamos con el tercer termino $|0\rangle + e^{i2\pi x\, 2^{-2}}.|1\rangle$, nótese que este requiere una puerta de Hadamard y dos puertas controladas para el desplazamiento de fase, sucesivamente cada término requiere una puerta lógica adicional de desplazamiento de fase a la del término anterior. De esta manera si se suma el número de puertas lógicas se tendría un total de $1 + 2 + 3 + \cdots + (n-1) + n$ puertas lógicas o pasos a ejecutar, esto es $1 + 2 + 3 + \cdots + (n-1) + n = n.(n+1)/2 = \frac{1}{2}n^2 + \frac{1}{2}n$, de manera que aplicar la transformada de Fourier requeriría de $\frac{1}{2}n^2 + \frac{1}{2}n$ pasos en total, por lo cual el orden de complejidad para esta parte del programa es $O(n^2)$, esto es $O(n^2) \propto$ número de qbits.

Resumiendo y para finalizar, tendríamos entonces que la parte clásica del algoritmo de Shor donde se calcula la exponenciación modular se realizaría en $O((\log n)^3)$, esto una vez se tenga el valor de r. Seguidamente, en el subprograma se tiene que el tiempo computacional de ejecución para el cálculo de r mediante fracciones continuas extendidas seria de $O((\log n)^3)$ (esto ya que $k < n$, por lo cual $log\ n$ seria una cota superior para el proceso), esto más la complejidad que representa implementar la transformada de Fourier, daría un total de $O((\log n)^3)$ para el coste del subprograma, así pues teniendo en cuenta el coste computacional de $O((\log n)^3)$ dado tanto en la parte clásica y como en el subprograma, tendríamos entonces así un tiempo de ejecución de $O((\log n)^3)$, para el algoritmo de Shor, es decir una solución al problema de la factorización prima en tiempo polinomico (en cuanto al tamaño de la entrada o modulo), lo cual representaría claramente la obsolencia del sistema de cifrado RSA ante este algoritmo de factorización.

Por último, se presentará el siguiente teorema el cual permitirá posteriormente medir el desempeño del algoritmo de Shor al obtener de manera sencilla la probabilidad de éxito asociada a un m fijo.

Teorema 5: Sea p un primo impar, y sea 2^d la potencia más grande de 2 que divida a $\phi(p^\alpha)$. Entonces el número entero 2^d divide el orden r de un $x\ (mod\ p^\alpha)$ (para un elemento x elegido al azar en $\mathbb{Z}^*_{p^\alpha}$) con una probabilidad de $1/2$.

Siendo así, se procederá a finalizar este apartado enunciando la siguiente proposición, la cual proporcionará la probabilidad de éxito del algoritmo de Shor, la cual bien puede ser refinada al tomar diversos valores de x aleatorios.

Proposición: Supongamos que $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_m^{\alpha_m}$ la factorización prima de un entero positivo compuesto. Sea x un elemento elegido aleatoria y uniformemente de $\mathbb{Z}_n^*$ y sea r el orden de x, modulo n, esto es $x^r \equiv 1\ (mod\ n)$. Entonces la probabilidad de que r sea par y $x^{r/2} \not\equiv -1\ (mod\ n)$ es por lo menos $1 - 1/2^m$, es decir:

$$P\{[2|r] \wedge [x^{\frac{r}{2}} \not\equiv -1\ (mod\ n)]\} \geq 1 - 1/2^m$$

Prueba: Vamos a probar que $P\{r\ es\ impar \vee [x^{\frac{r}{2}} \equiv -1\ (mod\ n)]\} \leq 1/2^m$. En efecto, en virtud al teorema chino del residuo, el seleccionar un x uniformemente aleatorio de $\mathbb{Z}_n^*$ es equivalente a seleccionar x_j independiente y uniformemente al azar de $\mathbb{Z}^*_{p_j^{\alpha_j}}$, y requiriendo que $x \equiv x_j\ (mod\ p_j^{\alpha_j})$ para cada j. Ahora bien, sea r_j el orden de $x_j\ (mod\ p_j^{\alpha_j})$. Sea 2^{d_j} la potencia más grande de 2 que divide a r_j y 2^d la potencia más grande de 2 que divida a r. Se probará entonces que para cualquier valor de r primo o bien $x^{\frac{r}{2}} \equiv -1\ (mod\ n)$, es necesario que d_j tome los mismos valores para cada valor de j. Así por el teorema 5, tendríamos entonces que la probabilidad de ocurrir el evento dado por $x^{\frac{r}{2}} \equiv -1\ (mod\ n)$ es a lo mucho$1/2^m$. Primero considérese que r es impar, siendo así entonces es sencillo ver que $r_j|r$ para todo valor de j, por lo tanto r_j es impar, asi que $d_j = 0$ para $j = 1,2, \dots k$. Ahora si se considera explícitamente el caso cuando r es par y $x^{\frac{r}{2}} \equiv -1\ (mod\ n)$, tendríamos entonces que $x^{\frac{r}{2}} \equiv -1 (mod\ p_j^{\alpha_j})$, así que $r_j \nmid (r/2)$. Por lo cual desde que $r_j | r$ se tendrá que $d_j = d$ para cualquier valor de d. Así pues del teorema 5 se deduce que $P\{r\ es\ impar \vee [x^{\frac{r}{2}} \equiv -1\ (mod\ n)]\} \leq 1/2^m$, y por lo tanto se tendría que $P\{[2|r] \wedge [x^{\frac{r}{2}} \not\equiv -1\ (mod\ n)]\} \geq 1 - 1/2^m$. ■

9. CONCLUSIÓNES

Como se puede observar, el algoritmo de factorización de Shor, a diferencia de los otros algoritmos clásicos como el de Fermat, el $p-1$ de Pollar, o incluso la criba general del cuerpo de números (el cual entrega $O(exp^{[((64/9)^{1/3}+o(1)).(\log n)^{1/3}.(\log\log n)^{2/3}]})$ como el mejor tiempo de ejecución desde el punto de vista de la computación clásica); el algoritmo de Shor plantea una solución al problema de factorización de números enteros largos en un tiempo de ejecución polinomio (en el tamaño de la entrada), estableciendo así una complejidad algorítmica para el mismo de solo $O((log\, n)^3)$ siendo n el entero largo a factorizar y $log\, n$ el tamaño de la entrada, desarrollando además esta tarea con una probabilidad de éxito de $1-1/2^m$, donde m responde al número de factores primos distintos de n, ofreciendo además la posibilidad de desarrollar refinamientos de dicha probabilidad mediante la evaluación sucesiva de otros enteros a escogidos aleatoriamente para la ecuación exponencial modular. En este sentido, el algoritmo de Shor aprovecha algunos principios de la física moderna de partículas como lo son el principio de la superposición, el paralelismo cuántico, y algunos métodos matemáticos como la transformación de Fourier, el periodo de funciones, transformada de Hadamard, los espacios $\mathcal{H}_n$ de Hilbert, y otros para construir algunos conceptos fundamentales como el qbits y los estados del mismo, para lograr así una solución óptima al problema de la factorización de enteros grandes. En este orden de ideas, el algoritmo de Shor plantea una serie de implicaciones importantes en la criptografía de llave pública para los sistemas basados en el problema de la descomposición prima como lo es el RSA, ya que una vez encontrado los factores que componen el módulo (n) de las ecuaciones exponenciales en congruencias tratadas en el RSA, entonces bastaría con calcular la función $\phi(n)$ de Euler e implementar el algoritmo extendido de Euclides u otro para computar la clave privada diseñada por el sistema de cifrado, rompiendo así la seguridad del mismo.

Por otro lado, y sin embargo, a pesar de las implicaciones que tendría para el RSA y de otros sistemas de cifrado clásico, el algoritmo de Shor requiere ser ejecutado en un sistema

de cómputo no convencional y que aún están en plena etapa de desarrollo, eso sumado a la gran dificultad que conlleva el desarrollo de tales sistemas (la cual y aunque se han desarrollado muchos avances en el tema, aumenta en proporción al número de qbits que estas manejen), hacen que estas máquinas estén a un par de décadas de ser ampliamente comercializadas, siendo esta la única razón por la cual tanto el sistema de cifrado RSA y en general muchos otros sistemas de llave publica clásicos sigan siendo ampliamente seguros ante ataques por algoritmos de factorización entera.

10. CRONOGRAMA

Ítem	FEBRERO DE 2014				MARZO DE 2014				ABRIL DE 2015				MAYO DE 2015			
Revisión de la literatura																
Análisis de la Información																
Redacción del Documento Final																
Revisión Personal del Documento																
Entrega del Material Intelectual																

11. PRESUPUESTO Y RECURSOS

11.1. RECURSO HUMANO

En el presente trabajo se recibirá la asesoría en cuanto a forma y aspectos generales a través del Ingeniero y M.Sc. Deivis Eduard Ramírez Martínez, de igual manera se tendrá la participación del Ingeniero y Matemático Juan Carlos Montaño (autor del presente material monográfico), además del Matemático y M.Sc. Pedro Jairo Vega (como asesor central en los temas referentes al componente teórico y matemático del tema a tratar).

11.2. PRESUPUESTO

ITEMS	CANTIDAD	COSTO UNITARIO	COSTO TOTAL
Computadores	1	$ 1.500.000,00	$ 1.500.000,00
Servicio de Internet	1 x 6 meses	$ 55.000,00	$ 330.000,00
Impresora Epson L200	1	$ 350.000,00	$ 350.000,00
Transporte	20	$ 1.700,00	$ 34.000,00
Recarga Tinta	4	$ 15.000,00	$ 60.000,00
Resma de Papel	3	$ 8.000,00	$ 24.000,00
Subtotal			$ 2.298.000,00
Gastos Imprevistos (15%)			$ 344.700,00
TOTAL			$ 2.642.700,00

12. REFERENCIAS BIBLIOGRAFÍCAS

1. SABOGAL ROZO, Esther Angélica, Módulo de Proyecto de Seguridad Informática Universidad Nacional Abierta y a Distancia (UNAD). 2013.

2. ICONTEC. Norma Técnica Colombiana NTC 1486. Documentación. Presentación de Tesis, Trabajos de Grado y Otros Trabajos de Investigación. Instituto Colombiano de Normas Técnicas y Certificación (INCONTEC). 2008.

3. SHOR, Peter W. (AT&T Research). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. Cornell University Library, Journal reference: SIAM J.Sci.Statist.Comput. 1995.

4. RIGOLIN, Gustavo and RIEZNIK, Aníbal A. Introduction to Quantum Cryptography. Revista Brasileira de Ensino de Fisica, v. 27, n. 2005.

5. NIELSEN A., Michael – CHUANG L., Isaac. Quantum Computation and Quantum Information. Cambridge University Press. 2000.

6. Dan C. Marinescu and Gabriela M. Marinescu. From Classical to Quantum Information. 2010.

7. Yu. I. Manin. Classical Computing, Quantum Computing, and Shor's Factoring Algorithm. Max–Planck–Institut fur Mathematik, Bonn, Germany. 1999.

8. Matthew E. Briggs. An Introduction to the General Number Field Sieve. Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Master of Science in Mathematics. 1998.

9. Amalia Duch. Análisis de Algoritmos. Barcelona. Marzo de 2007

10. Gustavo Rigolin e Andres Anibal Rieznik. Introduction to Quantum Cryptography. Instituto de Fisica Gleb Wataghin, Universidade Estadual de Campinas, Campinas, SP, Brasil. 2005.

11. Peter Shor. Introduction to Quantum Algorithms. Proceedings of Symposia in Applied Mathematics. 2000.

12. Tomas Recio. La Columna de Matemática Computacional. La Gaceta de la RSME, vol. 5.3. (2002).

13. Peter Stevenhagen. The Number Field Sieve. Algorithmic Number Theory MSRI Publications Volume 44, 2008.

14. Carl Pomerance. The Number Field Sieve. Proceedings of Symposia in Applied Mathematics. Vol. 48. 1994.

15. A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, J.M. Pollard. The Number Field Sieve. Bellcore, 435 South Street, Morristown, NJ 07960.

16. Castiblanco Caro Alejandro, Salazar Amaya Lissette Carolina, Amador Donado Siler. Criptoanálisis del Algoritmo RSA y la Complejidad de la Factorización de Números Semiprimos. Escuela Colombiana de Carreras Industriales y Fundación Universitaria de Popayán.

17. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to algorithms, Third Edition. Massachusetts Institute of Technology. 2009.

18. Donald R. Byrkit, Anthony J. Pettofrezzo. Introducción a la Teoría de los Números. Editorial Prentice Hall International. New Jersey. 1972.

19. Lucena López Manuel José. Criptografía y Seguridad en Computadores. Versión 4-0.8.1. Universidad de Jaén. 3 de Marzo de 2010.