Trabajo de Investigación Factorización en base a Números Primos

Universidad Católica San Pablo
Kevin Jhomar Sanchez Sanchez

ÍNDICE

I. Introducción

Durante todo el semestre hemos visto y desarrollado varios algoritmos de criptografía, y uno de las primeras capas de seguridad que hemos visto (como en el RSA) es la generación de primos grandes y sus multiplicaciones, es de aquí que nace esta investigación que estaremos desarrollando para poder factorizar estos primos, veremos todo tipo de métodos y algoritmos que e podido recopilar.

II. RSA

El criptosistema RSA, llamdo asi por sus inventores R. Rivest, A. Shamir, and L. Adleman, es el mas extensamente usado como sistema de clave publica. Se puede utilizar para proporcionar para la encriptacion de mensajes y en la firma digital y su seguridad se basa en la obstinacion de la factorizacion de enteros. En esta seccion empezare a describir la encriptacion RSA, su seguridad y algunos problemas de implementacion.

II-A. ***El RSA***

Sean p y q números primos grandes (más de 1024 bits) y del mismo tamaño. Comenzaremos con el paso de seguridad dentro del RSA, ahora los multiplicaremos y obtendremos que $n = p \cdot q$ ademas de $\phi(n) = (p-1)(q-1)$. Dentro de estos dos pasos como ya dije antes se genera la seguridad la seguridad, porque si bien yo tengo que $p = 83$ y $q = 97$ entonces $n = 83 \cdot 97 = 8051$ y $\phi(n) = 82 \cdot 96 = 7872$. Dentro del algoritmo de RSA nuestro $\phi(n)$ nunca se muestra al público, solo se muestra el n y si quisieran hallar el $\phi(n)$ tendrían que descomponer el numero $n = 8051$, es aquí donde nos damos cuenta que factorizar números es difícil de cierta forma, ya que uno no va a operar con números pequeños sino con números de **1024 bits a más** que cuentan con 308 dígitos. Esto esto se detallara mejor en la siguiente sección.

Continuando con la generación de claves del RSA. Sean los enteros e y d generadores de las claves en RSA son llamados exponente de cifrado y exponente de decifrado, mientras que n es llamado el modulo. Nos serviran de mucha utilidad a la hora de querer generar la encriptacion y desencriptacion. Tienen que cumplir con $ed \equiv 1 \ (mod \ \phi(n))$, lo que significa que $ed = 1 + k\phi(n)$, Ahora si $mcd(m, p) = 1$ entonces por teorema de Fermat.

$$m^{p-1} \equiv 1 \ (mod \ p)$$

Ahora el aumento de ambos lados de esta congruencia a la potencia $k(q-1)$ y luego multiplicando ambos lados por m

$$m^{1+k(p-1)(q-1)} \equiv m \ (mod \ p)$$

Por otro lado, si $mcd(m, p) = p$, entonces esta última congruencia es válido desde cada lado, es congruente con 0 módulo p. Por lo tanto, en todos los casos

$$m^{ed} \equiv m \ (mod \ p)$$

Por ser el mismo argumento que

$$m^{ed} \equiv m \ (mod \ q)$$

Finalmente, desde p y q son primos distintos, sigue que:

$$m^{ed} \equiv m \ (mod \ n)$$

Por lo tanto,

$$c^d \equiv (m^e)^d \equiv m \ (mod \ n)$$

II-A1. ***Pseudo-Algoritmo para la generación de Claves:***
Para cada objeto que se crea en RSA le corresponde una clave privada

---

Por lo que para cada objeto le sigue:

1. Generar dos numeros grandes(y distintos) primos p y q, del mismo tamaño.
2. Calcular $n = pq$ y $\phi(n) = (p-1)(q-1)$
3. Escoger un entero e, tal que $1 < e < \phi(n)$, hasta que $mcd(e, \phi(n)) = 1$
4. Usando el algoritmo extendido de Euclides, calcular el unico d, tal que $1 < d < \phi(n)$,hasta que $ed \equiv 1(mod\ \phi(n))$
5. Donde e vendria a ser nuestra clave publica y d nuestra clave privada.

II-A2. ***Pseudo-Algoritmo para la encriptación y desencriptación:***
Supongamos que B encripta un mensaje para m para A, y A tiene que desencriptarlo

---

Por lo que para cada objeto le sigue:

1. Encriptación
   1.1. Optener las clave publica de A (n, e).
   1.2. Representar el mensaje como un numero m en el intervalo $[0, n-1]$.
   1.3. Calcular $c = m^e\ mod\ n$
   1.4. Enviar el texto cifrado c a A
2. Desencriptación
   2.1. Usar la clave privada d para recuperar el mensaje m, $m = c^d\ mod\ n$.

II-B. ***Seguridad de RSA***

La tarea se enfrenta un adversario pasivo es el de la recuperación de texto claro m a partir del correspondiente texto cifrado c, dada la información pública (n, e) del receptor previsto A. Esto es llamado el problema RSA (RSAP). No hay algoritmo eficiente conocida para este problema.

Un enfoque posible que un adversario podría emplear para resolver el problema RSA es primer factor n, y luego calcular $\phi(n)$ y d al igual que lo hizo en el anterior algoritmo. Una vez que d es obtenido, el adversario puede descifrar cualquier texto cifrado destinado a la A

Por otro lado, si un adversario podría de alguna manera calcular d, entonces podría obtener un factor n de manera eficiente de la siguiente manera. Primero que notamos es que $ed \equiv 1\ (mod\ \phi(n))$, donde hay un entero k que $ed - 1 = k\phi(n)$, Por lo tanto $a^{ed-1} \equiv 1\ (mod\ n)$ para todo $a \in Z_n^*$. Sea $ed - 1 = 2^s t$, donde t es un entero impar. Entonce se puede mostrar que existe un $i \in [1, s]$ donde $a^{2^{i-1}} \neq \pm 1\ (mod\ n)$ y $a^{2^i t} \equiv 1\ (mod\ n)$ por lo menos para todo $a \in Z_n^*$; Si a e i son enteros tales que $(a^{2^{i-1}} - 1, n)$ es un factor no trivial de n. Así, el adversario sólo tiene que seleccionar en repetidas ocasiones al azar un $a \in Z_n^*$ y verificar si un $i \in [1, s]$ satistafe la existencia de la propiedad, el número esperado de los ensayos antes de un factor no trivial de n se obtiene es de 2. Esta discusión establece lo siguiente.

El problema de calcular el descifrado RSA exponente d de la clave pública (n, e), y el problema de la factorización de n, son equivalentes computacionalmente.

Ademas que cuando la generación de claves RSA, es imperativo que el números primos p y q pueden seleccionar en tal manera que la factorización $n = pq$ es computacionalmente imposible.

III. Algoritmos de Factorización

III-A. ***MÉTODO DE FERMAT***

[1]El algoritmo de Fermat es un algoritmo de factorización especifico, es decir, el número a factorizar debe cumplir unos requisitos previos. [6] [5] [2]

Actualmente no es uno de los algoritmos más implementados y utilizados, de hecho apenas se usa salvo que se sepa previamente que el número que queremos factorizar tiene dos factores cercanos a la raíz cuadrada del propio número. No obstante, el algoritmo de Fermat contiene la idea principal en la que se basa algunos de los algoritmos más potentes que se conocen, como la criba cuadrática y el algoritmo de fracciones continuas.

El objetivo que persigue el algoritmo de Fermat consiste en descomponer un número en dos factores. Para hallar una descomposición en factores primos,debemos continuar factorizando sucesivamente cada uno de los factores que obtengamos hasta que todos ellos sean números primos.

III-A1. ***Conceptos teóricos y matemáticos****:* Este método se basa en la representación de un número natural impar como una diferencia de cuadrados:

$$n = a^2 - b^2$$

Dado esto lo podemos descomponer como $(a+b)(a-b)$, de esta forma obtenemos una factorización de n. Entonces de las preposiciones anteriores podemos representar a n, cuando $n = cd$ es una factorizacion de n, entonces:

$$n = \left(\frac{c+d}{2}\right)^2 - \left(\frac{c-d}{2}\right)^2$$

Entonces como n es impar, podemos decir que c y d también son impares, por lo que su semisuma y semidiferencia serán enteros.

Existe una variación del algoritmo de Fermat que funciona de una formas mas eficiente en determinadas situaciones.Se le atribuye al matemático belga Maurice Kraitchik (1882-1957) y se basa en el uso de congruencias.

III-A2. ***Descripción del Pseudo-Algoritmo****:* Se van tomando varios valores de a, hasta que $a^2 - n = b^2$ sea un cuadrado.

Entrada: Un numero impar n
Salida: Un factor del numero n

1. $a \leftarrow \sqrt{n}$
2. $b_{cu} \leftarrow a * a - n$
3. Mientras b_{cu} no sea cuadrado hacer:
   3.1. $a \leftarrow a + 1$
   3.2. $b_{cu} \leftarrow a * a - n$
4. Retorno $a - b_{cu}$ o $a + b_{cu}$

III-A3. ***Ejemplo y seguimiento****:*
Ejemplo. Factorizar $N = 2093713$ mediante el algoritmo de Fermat.

Solución. Partiremos de $x = [\sqrt{2093713}] + 1 = 1447$

x	$\sqrt{x^2 - N}$
1447	$\sqrt{96} = 9{,}79796$
1448	$\sqrt{2991} = 54{,}69$
1449	$\sqrt{5888} = 76{,}7333$
⋮	⋮
1462	$\sqrt{43731} = 209{,}12$
1463	$\sqrt{46656} = 216$

Ya hemos encontrado la forma de expresar nuestro número N como diferencia de dos cuadrados:

$$N = 1463^2 - 216^2 = (1463 + 216)(1463 - 216) = 1679 \cdot 1247$$

III-A4. ***Tiempo de Ejecución****:* La tabla tiempos respecto a los bits que mostrare a continuación es sacado del paper de investigación no es de mi autoria. [6]

Para la maquina, características:

1. Dos Xeon con cuatro núcleos de 6.6GHz cada uno
2. Memoria(RAM): 8.00GB

Bits	p	q	N	Tiempo Maquina 1	Tiempo Maquina 2
10	641	6700417	$2^{2^5} + 1$	13min, 34.309seg	1,320ms
12	16829	12467683	209818637207	22min, 39,720ms	2,473ms
14	64969	543391231	35303584886839	36h 59min, 31,000ms	1min, 50,206ms
16	7823323	975298717	7630076884576591	57h 37min, 42,000ms	2min, 48,382ms

III-B. ***MEJORA DE KRAITCHIK***

III-B1. ***Conceptos teóricos y matemáticos****:* [5]Para aplicar esta generalización del algoritmo de Fermat es suficiente con que N divida a $x^2 - y^2$ para encontrar un factor de N, es decir:

$$N | x^2 - y^2 = (x + y)(x - y)$$

si ahora N no divide ni a $x - y$ ni a $x + y$, entonces podemos concluir que $mcd(x+y) > 1$ y ya hemos encontrado un factor no trivial de N. por lo tanto, el objetivo será buscar números x e y tales que

$$x^2 \equiv y^2 (mod\ N)$$

$$x \neq \pm y (mod\ N)$$

Supongamos ahora que encontramos un conjunto de números $x_1, \cdots, x_n$, tales que $x_1^2 \equiv a_1 (mod\ N), \cdots, x_n^2 \equiv a_n (mod\ N)$ para ciertos e enteros $a_1, \cdots, a_n$. Si ahora un subconjunto $a_{i_1}, \cdots, a_{i_r}$. de $a_1, \cdots, a_n$ cumple que el producto $a_{i_1}, \cdots, a_{i_r}$ es un cuadrado, entonces

$$(x_{i1} \cdot x_i)^2 \equiv a_{ij} \cdots a_{i_r} (mod\ N)$$

y de esta forma tenemos la congruencia buscada $x^2 \equiv y^2 (mod\ N)$. Ahora esta congruencia puede o no satisfacer $x \neq y (mod\ N)$,lo cual determina finalmente si los números encontrados son útiles para la factorización de N.

III-B2. ***Ejemplo y seguimiento****:*

Ejemplo. Factorizar $N = 642537$

Solución. Definimos primero el polinomio $P(x) = x^2 - N$ y tomamos $x_0 = [\sqrt{N}] + 1$, en este caso tenemos $x_0 = 802$. Sea ahora $x_k = x_0 + k$, calculamos unos cuantos valores de $P(x_k)$, obteniendo la siguiente tabla:

k	X_k	$\mathbf{Q}(X_k)$
1	137	$168 = (2^3)(3)(7)$
2	138	$443 = (443)$
3	139	$720 = (2^4)(3^2)(5)$
4	140	$999 = (3^3)(37)$
5	141	$1280 = (2^8)(5)$

Ahora tenemos que $Q(139)Q(141) = 2^{12}3^25^2 = (2^6 \cdot 3 \cdot 5)^2 = 960^2$ es un cuadrado perfecto. Dicho de otra manera,

$$(139^2 - n)(141^2 - n) = 950^2$$

Modulo n es simple

$$(139 \cdots 141)^2 \equiv 960^2$$

y tenemos una solución a $x^2 \equiv y^2 \ (mod\ n)$, con $x \equiv (139)(141)(mod\ n) = 998$ y $y \equiv 960(mod\ n) = 960.Ahoracalcular$

$$(18601, 998 + 960) = 979,\ (18601, 998 - 960) = 19$$

y ahora ya tenemos los dos factores de n.

III-C. ALGORITMO POLLARD (p-1)

El algoritmo de factorización Pollard (p-1) puede ser usado eficientemente para buscar cualquier factor primo p de una composición de enteros n por cada $p - 1$. [6] [5] [1] [2] [4]

III-C1. ***Conceptos teóricos y matemáticos:*** Obtener un B que será un entero positivo, también un entero n se dice que debe ser un B suave o suave con respecto a un B obligado, si todo estos factores primos son $\leq B$.

La idea detrás de del algoritmo de Pollard (p-1) es el seguimiento. Donde sea B un valor suave obligado, donde también Q es el múltiplo menos común de todas las potencias de los primos$\leq B$ que son $\leq n$. Si $q^l \leq n$, entonces$l \ln q \leq \ln n$, así que $l \leq [\frac{\ln n}{\ln q}]$, por lo tanto

$$Q = \prod_{q \leq B} q^{[\ln n / \ln q]}$$

Donde el producto es distinto de $q \leq B$.Si p es un factor primo de n tales que $p - 1$ es B suave, entonces $p - 1|Q$, y consecuentemente para cualquier a se satisface $gcd(a, d) = 1$, el teorema de Fermat implica que $a^Q \equiv 1 \ (mod\ p)$. Por lo tanto si $d = gcd(a^Q - 1, n)$ entonces $p|d$. Es posible que $d = n$, en ese caso el algoritmo falla, sin embargo, esto es muy poco probable que ocurra si n tiene menos de dos de los factores primos grandes.

Nota: El B suave se selecciona sobre la base de la cantidad de tiempo que esta dispuesto gastar sobre el algoritmo de Pollard (p-1) antes de moverse sobre mas técnicas generales.

III-C2. ***Descripción del Pseudo-Algoritmo****:* Aplicaremos la teoría en la práctica.

Entrada: Un entero compuesto n que no es un primo a alguna potencia
Salida: Un factor d del número n

1. Escoger un numero B que será nuestro límite
2. Escoger un entero aleatorio a dentro del grupo $2 \leq a \leq n-1$
   Calcular $d \leftarrow gdc(a, n)$
   Si $d \leq 2$ entonces Retornar d
3. Para cada primo $q \leq B$ hacer:
   3.1. Calcular $l \leftarrow \left[\frac{ln(n)}{ln(q)}\right]$
   3.2. Calcular $a \leftarrow a^{q^l} \bmod n$
4. $d \leftarrow gcd(a-1, n)$
5. Si $d = 1$ o $d = n$, el algoritmo presenta fallos
   Sino Retornar d

III-C3. ***Ejemplo y seguimiento****:*

Ejemplo. Factorizar $N = 124786124891235$ mediante el método $p-1$ de Pollard.

Solución. En primer lugar determinamos los valores de las variables. En nuestro caso vamos a probar inicialmente con $B = 8$. Lo primero que haremos será calcular

$$m = mcm(1, 2, 3, ..., 8)$$

aunque en nuestro caso, como B se trata de un número muy pequeño, podemos tomar $m = B! = 40320$ porque claramente si $p-1|msm(1, 2, ..., 8)$ entonces también $p-1|8!$

A continuación, tomamos $a = 2$ y de este modo tenemos

$$2^{40320} - 1 \equiv 47614174037505 \ (mod\ 12478624891235)$$

y por ultimo calculamos

$$mcm(a^{m-1}, N)$$

$$mcm(47614174037505, 12478624891235) = 135$$

encontrando, de esta forma, un factor compuesto no trivial $p = 135$.

III-C4. ***Tiempo de Ejecución****:* La tabla tiempos respecto a los bits que mostrare a continuación es sacado del paper de investigación no es de mi autoria. [6]

Para la maquina, características:

1. Dos Xeon con cuatro nucleos de 6.6GHz cada uno
2. Memoria(RAM): 8.00GB

Bits	Tiempo
10	0ms
12	4ms
16	0ms
18	128ms
20	0ms
22	1,569ms
24	3,505ms

*III-D. **ALGORITMO RHO POLLARD***

[5] [1]El algoritmo Rho Pollard es uno de los usos especiales como algoritmo de factorización para encontrar factores de números pequeños dentro de una composición de enteros. [2] [4]

*III-D1. **Conceptos teóricos y matemáticos**:* Sea la función $f : S \to S$ tal que sea una función aleatoria, donde Ses un grupo finito de cardinalidad n. Sea x_0 un elemento aleatorio de S, y considerando la secuencia $x_0, x_1, x_2, ...$ definido por $x_{i+1} = f(x_i)$ para $i \geq 0$.Ya que S es infinito, la secuencia sera un ciclo, y consiste en una cola de la longitud esperada $\sqrt{\pi n/8}$ seguido por un ciclo interminable de la longitud $\sqrt{\pi n/8}$.

Un problema que surge en algunos trabajos criptoanaliticos, incluye la factorización de enteros y el problema logarítmico discreto, es de buscar buscar distintos indices i y j tales que $x_i = x_j$(una colisión se dice que a ocurrido).

Un método obvio para la búsqueda de colisiones es calcular el espacio x_i para $i = 0, 1, 2, ...$ y mirarlo por duplicados. El numero esperado de entradas que serán probados antes de detectar los duplicados es $\sqrt{\pi n/2}$. Este método requiere de O($\sqrt{n}$) en memoria y O($\sqrt{n}$) veces, asumiendo que x, esta guardado en una tabla de hash aquí que las nuevas entradas pueden añadirse en un tiempo constante.

*III-D2. **Descripción del Pseudo-Algoritmo**:* Aplicaremos la teoría en la práctica.

Entrada: Un entero compuesto n que no es un primo a alguna potencia
Salida: Un factor d del número n

1. $a \leftarrow 2$, $b \leftarrow 2$
2. Para $i = 1, 2, ...$ hacer:
   - 2.1. Calcular $a \leftarrow a^2 + 1 \bmod n$
     Calcular $b \leftarrow b^2 + 1 \bmod n$
     Calcular $b \leftarrow b^2 + 1 \bmod n$
   - 2.2. Calcular $d \leftarrow gcd(a - b, n)$
   - 2.3. Si $1 < d < n$ entonces Retorno d
   - 2.4. Si $d = n$ entonces termina el algoritmo con fallos

---

*III-D3. **Ejemplo y seguimiento**:*

Ejemplo. Buscar los fatores no triviales de $n = 455459$.
Solucion. Con la siguiente tabla de lista de valores de las variables a, b y d, miraremos cada iteración con paso 2 del algoritmo.

a	b	c
5	26	1
26	2871	1
677	179685	1
2871	155260	1
44380	416250	1
179685	43670	1
121634	164403	1
155260	247944	1
44567	68343	743

Por lo tanto hemos encontrado dos factores no triviales de 455459 que son 743 y $455459/743 = 613$

III-D4. ***Tiempo de Ejecución****:* El tiempo de ejecución a sido realizada con mi máquina, aqui se detallan las características de mi maquina:

1. Procesador: Intel Core i7-45100 CPU 2.00GHz $\times$ 4
2. Memoria(RAM): 7,7 GiB

Bits	Digitos	Tiempo
12	4	0.051000 ms
16	5	0.037000 ms
18	6	0.040000 ms
22	7	0.026000 ms
26	8	0.028000 ms
28	9	0.059000 ms
30	10	0.105000 ms

Yo se que los números de bit que e probado son pequeños pero sucede que aparir de 31 bits a mas, el numero de iteraciones incrementa demasiado y hace que se demore mucho, ademas que me devuelve como respuesta "Numero sin factorizaciónτ terminan mas o menos en 522.463000 ms

III-E. ALGORITMO DE FRACCIONES CONTINUAS

Como ya se introdujo anteriormente, el algoritmo de factorización mediante fracciones continuas fue descrito en 1931 por D.H.Lehmer y R.E Powers y fue posteriormente en 1975 cuando John Brillhart y Michael Morison desarrollaron un algoritmo de computadora basado en las ideas de Brillhart y Morison. A continuación se introducen varios conceptos necesarios para el desarrollo de este apartado. [6] [5]

III-E1. ***Conceptos teóricos y matemáticos****:* Se llama Fracción continua a una expresión del tipo:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}}$$

donde $a_0 \in Z, a_i \in N$ cuando $i \geq 1$. Se puede expresar la fracción continua de un número x como:

$$a_0, a_1, a_2, ...$$

Entonces, sea N el numero que queremos factorizar, en primer lugar debemos calcular el i-esimo reducido de $\sqrt{N}$y calcular $b_i \equiv m_i^2 (mod\ N)$. Según calculamos cada valor b_i descartaremos aquellos que no factoricen en la base de factores B escogida, esta base de números variara en función de N ya que dependiendo del tamaño del numero que queremos factorizar, sera suficiente con una base de factores pequeña o necesitaremos ampliarla.

Una vez que hayamos calculado suficientes números b_i buscaremos un producto de b_i cuyos factores aparezcan un numero para de veces, es decir, tendremos un conjunto $\{b_1, \cdots, b_k\}$ tal que $b_1 \cdots b_k = b^2$. De esta forma habremos encontrado una congruencia del tipo

$$\prod_{i=1}^{k} b_i \equiv \left(\prod_{i=1}^{k} m_1^2 \right) (mod\ N)$$

Por ultimo, para encontrar el factor de N se procederá como en el resto de algoritmos que buscan congruencias cuadráticas, calculando el máximo común divisor de N con la diferencia de los números cuyo cuadrado era congruente modulo N.

III-E2. ***Descripción del Pseudo-Algoritmo****:* Aplicaremos la teoría en la práctica.

Entrada: es un numero impar N, una cota C.
Salida: es p y q.

1. Iniciamos con: $m_0 = a_0 = \lfloor \sqrt{N} \rfloor, x_0 = \sqrt{N} - a_0$ y $b_0 \equiv a_0^2 \ (mod\ N)$
2. Para $i = 1, ..., r$ entonces
   2.1. $a_i = \lfloor 1/x_{i-1} \rfloor$, $x_i = (1/x_{i-1})$
   2.2. Si $i = 1$ entonces $m_i = a_0 a_1 + 1$ y $b_i \equiv m_i^2 \ (mod\ N)$
   sino $m_i = a_i m_{i-1} + m_{i-2}$ y $b_i \equiv m_i^2 \ (mod\ N)$
3. Descartar los b_i que tienen los factores primos mayores a C. De los restantes elegimos algunos valores b_i tales que sus factores primos aparezcan un numero par de veces, para encontrar una congruencia $\prod m_i^2 \equiv \prod p_i^2 \ (mod\ N)$

Nótese que x es racional si y solo si su fracción continua es finita.

III-E3. ***Ejemplo y seguimiento****:*
Ejercicio. Factorizar $N = 173131$ utilizando como base de factores $B = \{-1, 2, 3, 5, 7, 11, 13, 17\}$

Solución. En primer lugar calculamos una aproximación de la expresión de $\sqrt{N}$ como fracción continua

$$\sqrt{173131} \approx \{416, 11, 10, 1, 1, 2, 1, 2, 2, 1, 4, 1, 1, 3, 2, 2, 2, 3, 7, 13\}$$

y a continuación calculamos unos cuantos valores b_i

$[a_0, \cdots, a_i] = \frac{m_i}{n_i}$	$b_i \equiv m_i^2 (mod\ N)$	$b_i \equiv p_1^{e1} \cdots p_m^{e_m}$
$[416] = 416$	$-75 \equiv 416^2 (mod\ N)$	$-75 = -1 \cdot 3 \cdot 5^2$
$[416, 11] = \frac{4577}{11}$	$78 \equiv 4577^2 (mod\ N)$	$78 = 2 \cdot 3 \cdot 13$
$[416, 11, 10] = \frac{46186}{111}$	$-455 \equiv 46186^2 (mod\ N)$	$-455 = -1 \cdot 5 \cdot 7 \cdot 13$
$[416, 11, 10, 1] = \frac{50763}{122}$	$365 \equiv 50763^2 (mod\ N)$	$365 = 5 \cdot 73$
$[416, 11, 10, 1, 1] = \frac{96949}{233}$	$-258 \equiv 96949^2 (mod\ N)$	$-258 = -1 \cdot 2 \cdot 3 \cdot 43$
$[416, 11, 10, 1, 1, 2] = \frac{244661}{588}$	$457 \equiv 244661^2 (mod\ N)$	$457 = 457 \cdot 23$
$[416, 11, 10, 1, 1, 2, 1] = \frac{341610}{821}$	$-271 \equiv 341610^2 (mod\ N)$	$-271 = -1 \cdot 271$
$[416, 11, 10, 1, 1, 2, 1, 2] = \frac{927881}{2230}$	$261 \equiv 927881^2 (mod\ N)$	$261 = 3^2 \cdot 29$
$[416, 11, 10, 1, 1, 2, 1, 2, 2] = \frac{2197372}{5281}$	$-507 \equiv 2197372^2 (mod\ N)$	$-507 = -1 \cdot 3 \cdot 13^2$

En este punto observamos que $(-75 \cdot -507) = (-1)^2 \cdot 3^2 \cdot 5^2 \cdot 13^2$ o lo que es lo mismo $(-75 \cdot -507) = (-1 \cdot 3 \cdot 5 \cdot 13)^2$ por lo tanto tenemos la siguiente congruencia:

$$416^2 \cdot 2197372^2 \equiv 148203 \ (mod\ N) - 1 \cdot 3 \cdot 5 \cdot 13 \equiv 172936 \ (mod\ N)$$

de este modo calculamos ahora

$$mcd(148203 - 172936, 173131) \equiv 24733 mcd(148203 + 172936, 173131) \equiv 7$$

por lo que ya hemos encontrando dos factores no triviales de N

$$173131 \equiv 7 \cdot 24733$$

III-E4. ***Tiempo de Ejecución****:* La tabla tiempos respecto a los bits que mostrare a continuación es sacado del paper de investigación no es de mi autoría. [6]

Para la maquina, características:

1. Dos Xeon con cuatro núcleos de 6.6GHz cada uno
2. Memoria(RAM): 8.00GB

Bits	Tiempo	Cota	Precisión
12	0ms	500	200
14	468ms	750	600
16	300ms	750	600
18	3,532ms	1500	1750
22	18,906ms	3000	3000

III-F. MÉTODO DE DIXON

Ésta es una técnica probabilista de factorización, por lo que no tenemos ninguna garantía de que logremos encontrar algún factor de N, por lo menos en un periodo determinado de tiempo. Este método es utilizado para factorizar números con menos de 25 dígitos. [6] [3]

III-F1. ***Conceptos teóricos y matemáticos****:* El método de John D. Dixon fue publicado en 1981 y se basa en una idea muy simple. Si podemos encontrar $x, y \in Z$ con $x \neq \pm y \ (mod\ N)$ tales que $x^2 \equiv y^2 \ (mod\ N)$ entonces $N|(x-y)(x+y)$ pero N no es divisible por $(x-y)$ ni $(x+y)$, por consiguiente $mcd(x+y, N)$ es un factor no trivial de N (lo mismo ocurre con $mcd(x-y, N)$).

El método usa una base $B = p_1, ..., p_b$ que es un pequeño conjunto de primos sucesivos empezando con el 2. Primero se eligen aleatoriamente algunos valores x_i y se calculan $z_i \equiv x_i^2 \ (mod\ N)$. Cada valor de z i es factorizado de la siguiente forma:

$$z_i = p_1^{\alpha 1 i} p_2^{\alpha 2 i} \cdots p_b^{\alpha b i}.$$

Se eliminan aquellos valores de z_i tales que en su factorización los factores no pertenezcan a los valores en la base B. Para cada i, se considera el vector

$$a_i = (\alpha_1 \ (mod\ 2), \cdots, \alpha_{bi} \ (mod\ 2)) \in (Z_2)^b$$

Procuramos encontrar un subconjunto de a_k tal que la suma $(mod\ 2)$ sea el vector $(0, \cdots, 0)$, pero si no se logra encontrar tal subconjunto entonces se toma una base B más grande y se vuelve a realizar el procedimiento, una vez que se encontró dicho subconjunto se podrá calcular

$$\prod Z_k \equiv \prod x_k^2 \ (mod\ N)$$

Esto origina la congruencia deseada $x^2 \equiv y^2 \ (mod\ N)$, que (esperamos) nos llevará a la factorización de N con lo que tenemos un $50\,\%$ de probabilidad de encontrar un factor no trivial de N.

III-F2. ***Descripción del Pseudo-Algoritmo****:* Aplicaremos la teoría en la práctica.

Entrada: Un entero compuesto n. Sea $a \in Q_n \ (1 \leq a \leq n-1)$
Salida: Cuatro raíces cuadradas de a modulo n

1. Escoger un B suave
2. Sea $S = \phi$
3. Mientras $|S| \leq r$
   - 3.1. Escoger un x aleatorio dentro de Z_n^*
   - 3.2. Calcular $y = x^2 \ (mod\ n)$
   - 3.3. Si y es B suave, es decir $y = p_1^{e1} \cdots p_r^{er}$,
     1) Añadir $x^2 = p_1^{e1} \cdots p_r^{er}$ a S
   - 3.4. Encontrar un subconjunto T de los vectores de exponentes que se suma a 0 en módulo 2, utilizando eliminación gaussiana
   - 3.5. Usando T encontramos a, b tal que $a^2 \equiv b^2 \ (mod\ n)$
   - 3.6. Dividir n con $mcd(a-b, n)$

III-F3. ***Ejemplo y seguimiento****:*

Ejemplo. Supongamos que se desea factorizar $N = 2881$ tomando $B = -1, 2, 3, 5, 7$.
Soluciónn. Calculamos $\lfloor\sqrt{N}\rfloor = 53$, $\lfloor\sqrt{2N}\rfloor = 75$, $\lfloor\sqrt{3N}\rfloor = 92$, $\lfloor\sqrt{4N}\rfloor = 107$, $\lfloor\sqrt{5N}\rfloor = 120$, $\lfloor\sqrt{6N}\rfloor = 131$, $\lfloor\sqrt{7N}\rfloor = 142$, $\lfloor\sqrt{8N}\rfloor = 151$, $\lfloor\sqrt{9N}\rfloor = 161$, $\lfloor\sqrt{10N}\rfloor = 169$ y $\lfloor\sqrt{11N}\rfloor = 178$. Se eliminan los valores de z_i

$z_i \equiv x^2(\ \text{mód}\ N)$	$z_i \equiv p_1^{\alpha 1 i} \cdots p_b^{\alpha b i}$
$-72 \equiv 53^2\ (mod\ N)$	$-72 \equiv -1 \cdot 2^3 \cdot 3^2$
$2744 \equiv 75^2\ (mod\ N)$	$2744 \equiv 2^3 \cdot 7^3 \cdot 3^2$
$-179 \equiv 92^2\ (mod\ N)$	$-179 \equiv -1 \cdot 179$
$-75 \equiv 107^2\ (mod\ N)$	$-75 \equiv -1 \cdot 3 \cdot 5^2$
$-5 \equiv 120^2\ (mod\ N)$	$-5 \equiv -1 \cdot 5$
$-125 \equiv 131^2\ (mod\ N)$	$-125 \equiv -1 \cdot 5^3$
$-3 \equiv 142^2\ (mod\ N)$	$-3 \equiv -1 \cdot 3$
$-247 \equiv 151^2\ (mod\ N)$	$-247 \equiv -1 \cdot 13 \cdot 19$
$-8 \equiv 161^2\ (mod\ N)$	$-8 \equiv -1 \cdot 2^3$
$-249 \equiv 169^2\ (mod\ N)$	$-249 \equiv -1 \cdot 3 \cdot 83$
$-7 \equiv 178^2\ (mod\ N)$	$-7 \equiv -1 \cdot 7$

tales que en su factorización los factores no pertenezca a los valores de la base B, por consiguiente tenemos ocho factorizaciones, las cuales producen los siguientes vectores.

$$\begin{aligned} a_1 &= (1,1,0,0,0) \\ a_2 &= (0,1,0,0,1) \\ a_3 &= (1,0,1,0,0) \\ a_4 &= (1,0,0,1,0) \\ a_5 &= (1,0,0,1,0) \\ a_6 &= (1,0,1,0,0) \\ a_7 &= (1,1,0,0,0) \\ a_8 &= (1,0,0,0,1) \end{aligned}$$

Claramente $a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 = (0,0,0,0,0)\ (mod\ 2)$. Por lo que la congruencia que se obtiene es:

$$(75 \cdot 107 \cdot 120 \cdot 131 \cdot 142 \cdot 161 \cdot 178)^2 \equiv (2^3 \cdot 3 \cdot 5^3 \cdot 7^2)^2\ (mod\ 2881)$$

simplificando

$$404^2 \equiv 69^2\ (mod\ 2881)$$

finalmente se calcula

$$mcd(404 - 69, 2881) = 67$$

se obtiene un factor no trivial de N.

III-F4. ***Tiempo de Ejecución****:* La tabla tiempos respecto a los bits que mostrare a continuación es sacado del paper de investigación no es de mi autoria. [6]

Para la maquina, características:

1. Dos Xeon con cuatro núcleos de 6.6GHz cada uno
2. Memoria(RAM): 8.00GB

Bits	Tiempo	Cota	Precisión
10	132ms	100	120
12	252ms	500	220
14	2,849ms	750	425
16	8,881ms	750	1000
18	34,006ms	750	1100
20	6min, 34,657ms	1000	1150
22	7min, 44,286ms	1000	1350
24	27min, 258ms	1250	1750

III-G. ***CRIBA CUADRÁTICA***

Carl Pomerance, un matemático estadounidense especializado en teoría de números, propuso en 1981 un algoritmo llamado criba cuadrática que extendida las ideas de Dixon y de Kraitchik. La criba cuadrática era el algoritmo de factorización más rápido hasta que se descubrió la criba general del cuerpo de números en torno a 1993. No obstante, la criba cuadrática sigue siendo incluso más rápida que la criba general del cuerpo de números cuando se trata de números menores de 110 dígitos aproximadamente. [5] [1] [2]

Estudiando la criba de Eratóstenes, Pomerance se dio cuenta de que podía escoger un intervalo de números y señalar aquellos que habían sido marcados más de una vez al aplicar a dicho intervalo la criba de Eratóstenes, encontrando así números que tenían varios factores primos pequeños.

Con el objetivo de aplicarlo en la criba cuadrática, se planteó el hecho de que si realizamos la división de un número N por todos los números p de un intervalo y al final obtenemos 1, entonces ese número N factoriza sobre dicho intervalo. Si además el intervalo está acotado por B, tenemos un número B-suave. Con la idea planteada hasta ahora, observamos que no estamos consideran do potencias de primos, por lo que aplicando lo anterior a un número como por ejemplo un número como $63 = 3^2 \cdot 7$, no obtendríamos 1, sino 3. Para evitar esto, bastaría con dividir el número N por la mayor potencia posible del número primo.

Todo esto permite encontrar de un modo rápido qué números son B-suaves en un intervalo determinado. La criba cuadrática funciona eficientemente porque cuando tomamos módulo p a la hora de ejecutar el algoritmo, los múltiplos de dicho número p aparecen en posiciones concretas del intervalo.

III-G1. ***Conceptos teóricos y matemáticos****:* Supongamos un entero n que es factorizada, Sea $m = [\sqrt{n}]$, y considerando el polinomio $q(x) = (x+m)^2.Notamosque$

$$q(x) = x^2 + 2mx + m^2 - n \equiv x^2 + 2mx$$

Que es pequeño(relativo a n) si x es pequeño en valor absoluto. El algoritmo de la criba cuadrática escoge $a_i = (x+m)$ y verifica si $b_i = (x+m)^2 - n$ es un p_t suave. Notemos que $a_i^2 = (x+m)^2 \equiv b_i(modn)$. Notar también que si un primo p divide a b_i entonces $(x+m)^2 \equiv n(modp)$, y por lo tanto n es un residuo cuadrático modulo p. Entonces el el factor base necesita solo contener esos primos p para cada símbolo de Legendre$(\frac{n}{p})$ es 1. Ademas, desde b_i puede ser negativo, -1 es incluido en el factor base.

III-G2. ***Descripción del Pseudo-Algoritmo****:* Comenzaremos con el algoritmo

---

Entrada: Un entero compuesto n que no es un primo a alguna potencia
Salida: Un factor d del número n

1. Escoger un factor base $S = \{p_1, p_2, p_3, ..., p_t\}$ donde $p_1 = -1$ y p_j para $(j \geq 2)$ es el $(j-1)^{avo}$ primo de p para cualquier n es un residuo cuadrático modulo p.
2. Calcular $m \leftarrow [\sqrt{n}]$
3. (Recoger $t+1$ pares (a_i, b_i), los x son escogidos en el orden $0, \pm 1, \pm 2...$)
   Establecer $i \leftarrow 1$. Mientras $i \leq t+1$ hacer
   3.1. Calcular $b = q(x) = (x+m)^2 - n$ y usando el test de la división por elementos en S para cualquier b es
   3.2.
   3.3. $i \leftarrow i+1$
4. Usando álgebra lineal fuera de Z_2, se busca un grupo no basio $T \subseteq \{1, 2, ..., t+1\}$ hasta que $\sum_{i \in T} v_i = 0$
5. Calcular $x \leftarrow \prod_{i \in T} a_i \bmod n$
6. Para cada j,$1 \leq j \leq t$ hacer
   Calcular $l_j \leftarrow (\sum_{i \in T} e_{ij})/2$
7. Calcular $y \leftarrow \prod_{j=1}^{t} P_j^{l_j} \bmod n$
8. Si $x \equiv \pm y (mod\ n)$ entonces buscar un subgrupo no basio $T \subseteq \{1, 2, ..., t+1\}$ hasta que $\sum_{i \in T} v_i = 0$ y después ir al paso 5.(En el caso improbable de que tal subconjunto T no exista, reemplazar algunos de los pares(a_i, b_i) por pares nuevos y pasar al paso 4)
9. Calcular $d \leftarrow gcd(x-y, n)$ y Retornar d

III-G3. ***Ejemplo y seguimiento****:* **Ejemplo.** Factorizar $N = 87463$
Solución. En primer lugar vamos a calcular los parámetros M y B que determinaran el tamaño del intervalo de criba y de la base de factores respectivamente.

$$B = \left\lfloor \left(e^{\sqrt{ln(N) ln(ln(N))}} \right)^{\frac{\sqrt{2}}{4}} \right\rfloor = 6$$

$$M = \left\lfloor \left(e^{\sqrt{ln(N) ln(ln(N))}} \right)^{\frac{3\sqrt{2}}{4}} \right\rfloor = 264$$

De este modo ya tenemos el intervalo de criba $[-264, 264]$. Para formar la base de factores, necesitamos encontrar cinco números primos p que cumplan $(\frac{N}{p} = +1)$ por lo que calculamos el valor del símbolo de Legendre por la definición para unos cuantos. Es necesario tener en cuenta que, tanto -1 como 2, siempre van a formar parte de nuestra base de factores F.

p	3	5	7	11	13	17	19	23	29
$\left(\frac{N}{P}\right)$	1	-1	-1	-1	1	1	1	-1	1

Ya tenemos de esta forma nuestra base de factores

$$F = \{-1, 2, 3, 13, 17, 19, 29\}$$

A continuación debemos calcular para cada elemento p de F , los valores s_{p1} y s_{p2} como se indicó en la ecuación.

Los valores s_{p1} y s_{p2} son los que nos indica para cada factor de nuestra base F en qué momento empieza la progresión aritmética de números divisibles por p. Por ejemplo para $p = 13$ una de las progresiones comienza en

p	2	3	13	17	19	29
x	1	1,2	5,8	7,10	5,14	12,17
s_{p1}, s_{p2}	0	0,1	9,12	1,4	4,14	4,12

$d(a_9)$ y otra en $d(a_{12})$. A continuación se muestra una tabla que contiene el proceso de criba para un pequeño subintervalo de $[-M, M]$ como es $[0, 12]$.

Notar que $\lfloor N \rfloor = 295$. En la tabla no se muestran las potencias de cada factor primo pero será necesario almacenarlas, además de tenerlas en cuenta en la columna final.

i	$i + \lfloor \sqrt{N} \rfloor$	$d(a_i)$	-1	2	3	13	17	19	29	$d(a_i)$ final
0	295	-438	×	×	×					73
1	296	153			×		×			1
2	297	746		×						1
3	298	1341			×					1
4	299	1938		×	×		×	×		1
5	300	2537								1
6	301	3138		×	×					1
7	302	3741			×				×	1
8	303	4346		×						1
9	304	4953			×	×				1
10	305	5562		×	×					1
11	306	6173								1
12	307	6786		×	×	×			×	1

En este intervalo encontramos tres elementos que tienen la propiedad de ser B-lisos, es decir, factorizan completamente en nuestra base de factores F . Este proceso debería realizarse sobre todo el intervalo de criba, es decir, para $i \in [-264, 264]$. No obstante, vamos a aplicar a estos tres números B-lisos el proceso de obtención del vector de exponentes módulo 2. Una vez acabado el proceso de criba, debemos tener al menos $B + 1$ números

$d(a_i)$	Factorización	Vector de Exponentes	Vector de Exponentes (modulo 2)
153	$3^2 \cdot 17$	$[0, 0, 2, 0, 1, 0, 0]$	$[0, 0, 0, 0, 1, 0, 0]$
1938	$2 \cdot 3 \cdot 17 \cdot 19$	$[0, 1, 1, 0, 1, 1, 0]$	$[0, 1, 1, 0, 1, 1, 0]$
6786	$2 \cdot 3^2 \cdot 13 \cdot 29$	$[0, 1, 2, 1, 0, 0, 1]$	$[0, 1, 0, 1, 0, 0, 1]$

que factoricen completamente sobre la base de factores para tener así garantizado al menos un producto que sea cuadrado perfecto. En caso de no tener $B + 1$ elementos, se debe repetir el proceso anterior ampliando el intervalo describa.

A continuación construimos una matriz A cuyas columnas son los vectores de los exponentes módulo 2 de los números B-lisos y buscamos una solución para el sistema

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \cdot \bar{v} = \bar{O}$$

Una posible solución para el sistema anterior $\bar{v} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$ Por lo tanto, la solución involucra las cuatro primeras columnas, esto significa que el producto de los cuatro números $d(a_i) = (i + \lfloor N \rfloor)^2 - N$ asociados es un cuadrado, es decir

$$x^2 = (265 \cdot 278 \cdot 296 \cdot 307)^2 = 44816869024979257600$$

por otro lado, si tomamos

$$y^2 = (265^2 - N)(278^2 - N)(296^2 - N)(307^2 - N) = 182178565001316$$

tenemos dos números cuadrados que cumplen

$$x \neq y \ (mod \ N)$$

ya que

$$44816869024979257600 \equiv 10093 \ (mod \ N)$$

$$182178565001316 \equiv 10093 \ (mod \ N)$$

pero por otro lado

$$x \neq y \ (mod \ N)$$

donde tenemos

$$6694540240 \equiv 34759 \ (mod \ N)$$

$$13497354 \equiv 28052 \ (mod \ N)$$

Llegados a este punto, solo queda comprobar si x e y proporcionan un factor no trivial de N. Para ello calculamos

$$mcd(x + y, N) = mcd(6708037594, N) = 587$$

$$mcd(x + y, N) = mcd(6681042886, N) = 149$$

Por lo tanto concluimos que $N = 587 \cdot 149$

III-H. ***CRIBA GENERAL DE CAMPOS NUMÉRICOS***

La criba general de campos de números es un método complejo, comparte muchas similitudes con otros métodos basados en cribar. El nombre de este método es abreviado como (GNFS), pues en inglés se denomina“the general number field sieve”. Consiste de 5 pasos que se tienen que seguir. [5] [1]

Hoy en dia la criba general de campos numéricos se erige como el mas rápido de todos los que ya hemos visto anteriormente.

Heuristicamente su complejidad por factores de un entero n (que consta de $\lfloor log_2 n \rfloor + 1$) es de la forma

$$\left(\sqrt{\frac{64}{9}} + o(1) \right) (\ln n)^{\frac{1}{3}} (\ln \ln n)^{\frac{2}{3}}$$

III-H1. ***Conceptos teóricos y matemáticos:*** La criba general del cuerpo de números es uno de los algoritmos de factorización más complejos aunque también es uno de los métodos más rápidos a la hora de factorizar números enteros con un tamaño grande, considerando grande a partir de unos 110 dígitos. En este apartado vamos a dar una idea de los pasos en los que se basa dicho algoritmo, sin entrar en profundidad ya que el estudio completo de este método de factorización es muy extenso, se puede consultar en $The\ development\ of\ the\ number\ field\ sieve$ de $A.K.\ Lenstra$ y $H.W.\ Lenstra$.

El primer paso para factorizar un número N mediante la criba general del cuerpo de números consiste en encontrar un polinomio $f(x)$ que sea irreducible en $Z[x]$ y que además tenga una raíz m módulo N.

Para construir un polinomio en $Z_N[x]$ con una raíz m podemos hacerlo expresando el número N en base-m, es decir, expresamos N de la forma

$$N = \sum_{k=0}^{r} a_k m^k$$

y tomamos el polinomio

$$f(x) = \sum_{k=0}^{r} a_k x^k$$

suponemos que $f(x)$ y de este modo ya tenemos un polinomio que cumple los requisitos.

Los polinomios candidatos a ser utilizados en el algoritmo son muchos y no existe un método para determinar cuál será el que mejor funcionará al aplicar la criba general del cuerpo de números a cada número N.

El siguiente paso consiste en determinar el dominio sobre el que se va a aplicar el algoritmo. Para ello debemos especificar las distintas bases de factores que vamos a utilizar. Necesitaremos tres bases de factores: la base del factor racional, la base del factor algebraico y la base del carácter cuadrático. Para determinar el tamaño de las bases de factores lo haremos de forma empírica y dependerá del tamaño del número a factorizar.

La base del factor racional contendrá los números primos menores que un número w que representará la cota de la base. No obstante, la base del factor racional no almacenará solo dicha información, ya que cada número primo p se guardará junto al valor $p\ (mod\ m)$.

Por otro lado, la base del factor algebraico contendrá un lista con los pares (p, r) donde los números p son números primos y r es el menor número entero tal que $f(r) \equiv 0\ (mod\ p)$. El tamaño de la base del factor algebraico debe ser superior al de la base del factor racional.

Por último la base del carácter cuadrático tendrá una continuación de la base del factor algebraico, es decir, pares de números primos y las raíces pero con unos cuantos números p mayores que los anteriores. El tamaño de esta base de factores será inferior al de las anteriores.

El siguiente paso consiste en realizar una criba. Este es el cuello de botella del algoritmo ya que se realizan operaciones muy costosas en términos computacionales sobre dominios muy grandes, por lo tanto la mayor parte del tiempo del algoritmo se invierte en este paso.

El proceso de criba persigue el objetivo de encontrar pares de números (a, b) que cumplan

1. $mcd(a, b) = 1$
2. $a + bm$ tiene todos sus factores en la base del factor racional.
3. $(-b)^d f(\frac{a}{b})$ tiene todos sus factores en la base del factor algebraico.

Para ello, tomaremos b fijo y variaremos a en un intervalo $[-C, C]$ cuyo tamaño dependerá directamente del tamaño del número a factorizar. Si el valor escogido para C no es suficientemente grande, deberemos tomar un número C

superior. De esta forma calcularemos para los pares (a, b) los factores de $a + bm$ y de aquellos que factoricen en la base del factor racional, nos quedaremos con aquellos para los que $(-b)^d f(\frac{a}{b})$ tenga los factores en la base del factor algebraicoo.

Una vez hemos obtenido una lista de pares (a, b) que cumplen las propiedades requeridas, el objetivo es encontrar un subconjunto de la lista cuyo producto sea un número cuadrado. Nuevamente no necesitamos encontrar en la lista números cuadrados, nos basta con que el producto de varios de ellos sea un número cuadrado.

Este paso se puede llevar a cabo resolviendo un sistema de ecuaciones lineales, ya que incluso para matrices de un tamaño elevado, como el sistema solo contendrá un 1 en las posiciones de los números primos que aparezcan como factor con potencia impar y un 0 en las posiciones de los números primos que aparezcan como factor con potencia par o no aparezcan como factor, se podrá resolver de un modo relativamente eficiente.

Una vez se obtiene una solución al sistema, es decir, cuando tenemos números x e y cuyos cuadrados son congruentes módulo N, se procede como en el resto de algoritmos calculando $mcd(x - y, N)$ y $mcd(x + y, N)$ para ver si obtenemos un factor no trivial de N.

III-H2. ***Descripción del Pseudo-Algoritmo****:* Aplicaremos la teoría en la práctica.

Entrada: RFB(la base del factor racional), AFB(la base del factor algebraico), QCB(la base del caracter cuadrático), polinomio $f(x)$, raíz m de $f(x)(mod\ N)$, $rels = \{(a_0, b_0, \cdots, (a_t, b_t))\}$ de pares uniformes.
Salida: Matriz binaria M de dimension($\sharp$) $\times$ ($\sharp RFB + \sharp AFB + \sharp QCB + 1$).

1. Sea $M[i, j] = 0$
2. Para cada $(a_i, b_i) \in rels$
3. Si $a_i + b_i m < 0$ hacer $M[i, 0] = 1$
4. Para cada $(p_k, r_k) \in RFB$
   4.1. Sea l la potencia mas grande de p_k que divide a $a_i + b_i m$
   4.2. Si l es par, hacer $M[i, 1 + k] = 1$
5. Para cada $(p_k, r_k) \in AFB$
   5.1. Sea l la potencia mas grande de p_k que divide a $(-b_i)^{deg(f)} f(-\frac{a_i}{b_i})$
   5.2. Si l es par, hacer $M[i, 1 + \sharp RFB + k] = 1$
6. Para cada $(p_k, r_k) \in QCB$
   6.1. Si el simbolo de Legendre($\frac{a_i + b_i p_k}{r_k} \neq 1$), hacer $M[i, 1 + \sharp RFB + \sharp AFB + k] = 1$
7. Retorno M

III-H3. ***Tiempo de Ejecución****:* La tabla tiempos respecto a los bits que mostrare a continuación es sacado del paper de investigación no es de mi autoria. [6]

Para comenzar definamos **mips**. La notación **mips** significa millones de instrucciones por segundo ([43]) y es tomada como medida estándar para registrar la potencia de cómputo con que se cuenta. Un mips tiene como origen la potencia de cómputo que realizaba una DEC VAX 11/780. Un mipsy significa el número de instrucciones que se pueden realizar en un año con un poder de cómputo de un mips, es decir 31536×10^9instrucciones. Algunos datos interesantes son los mips que han sido utilizados para factorizar algunos números enteros.

Enseguida mostramos la estimación del poder de cómputo requerido para factorizar un número entero de la longitud respectiva pensando en usar la criba de campos numéricos general, ya que el número entero es aleatorio.

Dgitos	Bits	mipsy	Aprox
129	429		1000
154	512	29369.2	3×10^4
192	640	$2{,}990 \times 10^6$	3×10^6
231	768	$1{,}800 \times 10^8$	2×10^8
269	896	$7{,}331 \times 10^9$	7×10^9
308	1024	$2{,}198 \times 10^{11}$	2×10^{11}
346	1152	$5{,}150 \times 10^{12}$	5×10^{12}
385	1280	$9{,}835 \times 10^{13}$	9×10^{13}
423	1408	$1{,}580 \times 10^{15}$	9×10^{15}
462	1536	$2{,}189 \times 10^{16}$	2×10^{16}
500	1664	$2{,}666 \times 10^{17}$	3×10^{17}
539	1792	$2{,}898 \times 10^{18}$	3×10^{18}
577	1920	$2{,}849 \times 10^{19}$	3×10^{19}
616	2048	$2{,}558 \times 10^{20}$	3×10^{20}

IV. Comparación de Algoritmos

Hasta mediados de la década de 1990, los ataques fueron factoraje hecho usando un enfoque conocido como la criba cuadrática. El ataque contra RSA de 130 bits utiliza un algoritmo más reciente, Criba General de Campos Numéricos, y fue capaz de factorizar un número mayor que el de 129 bits a tan solo 20 % del esfuerzo de cálculo.

Veamos la siguiente tabla de tiempos respecto a los bits que mostrare a continuación es sacado del paper de investigación no es de mi autoria. [7]

Número de Dígitos decimales	Número Aproximado de bits	Prueba	MIPS years	Algoritmo
100	332	Abril 1991	7	Criba Cuadrática
110	365	Abril 1992	75	Criba Cuadrática
120	398	Junio 1993	830	Criba Cuadrática
129	428	Abril 1994	5000	Criba Cuadrática
130	431	Junio 1996	1000	Criba General de Campos Numéricos
140	465	Febrero 1999	2000	Criba General de Campos Numéricos
155	512	Agosto 1999	8000	Criba General de Campos Numéricos

Por lo tanto podemos decir que la Criba General de Campos Numéricos es mucho mas optimizado que la Criba cuadrática.

Ejemplos. Tiempo que corren algunos algoritmos conocidos son:

1. Algoritmo: Criba general de campos numericos.
   Tiempo: $L[n, 1/3, c_0 + o(1)]$, donde $c_0 = (\frac{64}{9})^1/3$
2. Algoritmo: Criba Cuadrática.
   Tiempo: $L[n, 1/2, 1 + o(1)]$
3. Algoritmo: Método de fracciones continuas.
   Tiempo: $L[n, 1/2, c_2 + o(1)]$, donde $c_2 = \sqrt{2}$

Para hacer un resumen algo mas especifico de funcionalidades de algoritmos

1. Criba Cuadrática: El algoritmo más rápido para factorizar números grandes de propósito general menores de 110 dígitos decimales (aproximadamente).
2. Criba General de Campos Numéricos: El algoritmo más rápido para factorizar números grandes de propósito general mayores de 110 dígitos decimales (aproximadamente).
3. Otros algoritmos: Son eficientes cuando los números de bits son pequeños.

V. Conclusión

Durante la investigación empecé a buscar cual era el mejor algoritmo, pero halle que en realidad depende mucho de tu manera de trabajo ya que si tu trabajas con un numero pequeño(de pocos bits) o limitado, es mejor utilizar el algoritmo de Fermat, pero si empezar a trabajar con algoritmos mucho más grandes y complejos, donde el trabajo de la factorización es la parte más escencial, entonces es mejor utilizar el método de Dixon o el de la Criba de Campos númericos.

Es interesante que estos algoritmos van mejorandose con el paso del tiempo, y aunque hoy por hoy el RSA de 1024 bits es estándar y se mantiene como un algoritmo muy eficiente, su seguridad se sigue centrando en la factorización de números. Llegará el día en que el RSA se rompa.

Para el desarrollo de este trabajo he necesitado el uso de algunos conceptos sobre teoría de números que están directamente relacionados con la base teórica de ciertos algoritmos. también he necesitado comprender ciertos apartados mas relacionados con criptografía que internamente están basados en algoritmos de factorización.

Referencias

[1] Scott A. Vanstone Alfred J.Meneses, Paul C.van Oorschot. Handbook of applied cryptography. *Handbook of Applied Cryptography*, pages 89–98.

[2] Scott A. Vanstone Alfred J.Meneses, Paul C.van Oorschot. Prime number. *A computational perspective*, pages 225–242.

[3] Eric Bach. Lecture 24: Dixon's algorithm. *Lecture 24: Dixon's Algorithm*, pages 1–3.

[4] Jose Raul Duran Diaz. Números primos especiales y sus aplicaciones criptográficas. *Números primos especiales y sus aplicaciones criptográficas*, pages 98–101.

[5] Urko Nalda Gil. Factorización. *Factorización*, pages 9–41.

[6] Leonel Sergio Carrasco Pérez. Factorización de enteros. *Factorización de Enteros*, pages 13–65.

[7] William Stallings. Criptography and network security. *Principles and Practices*, pages 276–277.

VI. ANEXOS

VI-A. Implementación del RSA

Antes que nada mostrare como es la organizacion de mi clase RSA.

```
1.  class RSA{
2.      private:
3.          ZZ e, d, N, ON;
4.          ZZ p, q;
5.          ZZ seed;
6.          bool gpass;
7.          int bits;
8.      public:
9.          string alf;
10.
11.         RSA(ZZ);
12.         RSA(ZZ, ZZ);
13.         vector<ZZ> getPublic(void);
14.         string _c(string);
15.         string _d(string);
16.         ZZ resto_chino(vector<ZZ>, vector<ZZ>);
17.
18.         string ZZtoString(const ZZ);
19.         ZZ StringtoZZ(string);
20.         string AddRightZero(string, int);
21.         string AddLeftZero(string, int);
22.
23.         string ZZtoStringBits(ZZ);
24.         ZZ StringtoZZBits(string);
25.
26.         bool Miller(ZZ);
27.         ZZ mulmod(ZZ, ZZ, ZZ);
28.         ZZ randomPrime(int);
29. };
```

Generación de aleatorios

Para esta sección uso una librería llamada openCV, la cual hace capturas capturas de fotos con la cámara web de mi computadora. Lo que hago en esta librería es tomar el arreglo de números que me da y juntarlos para posteriormente utilizarlos como generadores una semilla(seed), para luego usarlo para generar números primos.

```
1.  vector<ZZ> getCamera(){
2.      vector<ZZ> lseedcamera;
3.      ZZ sum = (ZZ)1, nm = (ZZ)0;
4.      VideoCapture cap;
5.      cap.open(0);
6.      if( !cap.isOpened() ){
7.          cout << Çould not initialize capturing..."<<endl;
8.          sum = (ZZ)0;
9.      }else{
```

```
10.             namedWindow( ”Generate Seed”, 1 );
11.             Mat frame;
12.             for(int i=0;i<1;i++){
13.                 cap >> frame;
14.                 if( frame.empty() ) break;
15.                 imshow(”Generate Seed”, frame);
16.         }
17.             Vec2b numeric;
18.             for(int i = 0; i < 3; i++){
19.                 for(int j = 0; j < 1; j++,nm=nm+2){
20.                     numeric = frame.at¡Vec2b¿(i, j);
21.                     if(numeric[0] != 0 and numeric[1] != 0){
22.                         sum *= numeric[0];
23.                         sum *= numeric[1];
24.                     }
25.                     if((i == (ZZ)0 and j == (ZZ)0 ) or (i == (ZZ)1 and
26.                     j == (ZZ)0) or (i == (ZZ)2 and j == (ZZ)0) ){
27.                         lseedcamera.push_back(sum);
28.                         sum = (ZZ)1;
29.                     }
30.                 }
31.             }
32.         }
33.         return lseedcamera;
34. }
```

Generación de Números Primos

```
1.  ZZ RSA::randomPrime(int n){
2.      ZZ nnumber, s, c, z, p, t;
3.      bool st = true, ct = true;
4.      string number, newnumber, num;
5.      int a, b, d, e;
6.      /*Paso A*/
7.
8.      ZZ seed = this-¿seed;
9.
10.     /*Paso B*/
11.     for (int i = 0; i ¡n; ++i){
12.         s = (seed & (ZZ(1)«3))»3;
13.         c = (seed & (ZZ(1)«2))»2;
14.         z = mod(s+c, (ZZ)2);
15.         seed = (seed<<1)|z;
16.     }
17.
18.     number = ZZtoStringBits(seed);
19.     number = number.substr(0, n);
20.     while(st){
21.         /*Paso C*/
```

```
22.             srand(time(NULL));
23.             p = mod((ZZ)rand(), (ZZ)n);
24.             t = mod((ZZ)rand(), (ZZ)n);//
25.             conv(b, p);
26.             conv(e, t);
27.             b = number[b]-'0';
28.             e = number[e]-'0';
29.
30.             /*Paso D*/
31.             num = number.substr(0, n/2);
32.             a = num[0]-'0';
33.             num = num.substr(1, num.size());
34.             conv(b, mod(ZZ(b + a),(ZZ)2) );
35.             num += '0'+b;
36.             newnumber += num;
37.
38.             num = number.substr(n/2, n);
39.             a = num[(num.size()-1)]-'0';
40.             num = num.substr(0,num.size()-1);
41.             conv(b, mod(ZZ(b + a),(ZZ)2) );
42.             num.insert(0, 1, b+'0');
43.             newnumber += num;
44.             //
45.             newnumber[0] = '1';
46.             number = newnumber;
47.             nnumber = StringtoZZBits(number);
48.             newnumber.clear();
49.             d = number[number.size()-1]-'0';
50.             if( (d & 1) )
51.                 if(Miller( nnumber)) st = false;
52.         }
53.         return nnumber;
54.     }
```

Test de Primalidad Criba de Eratostenes

```
1.  ZZ eratostenes(int bits){
2.          int u = 0;
3.          ZZ num, onum, oc, ic, a = ZZ(1);
4.          int n;
5.          /*Generamos los bits con puros unos*/
6.          for(int i = bits-1; i>=0; i–)
7.                  num = num | a<<i;
8.          conv(n,num);
9.          num = ZZ(1)<<n;
10.
11.         for(int i = 2; i < n and i*i < n; i++){
12.                 onum = (num & (1 << (i-1))) >> i-1;
```

```
13.             if(onum == 0){
14.                 for (int o = i+1; o < n; ++o){
15.                     onum = (num & ((ZZ)1 << (o-1))) >> o-1;
16.                     if(onum == 0){
17.                         if(mod( (ZZ)o, (ZZ)i) == 0){
18.                             num = num | ((ZZ)1 << (o-1));
19.                         }
20.                     }
21.                 }
22.             }
23.         }
24.
25.         for (int i = 2; i ¡n; ++i){
26.             onum = (num & ((ZZ)1 << (i-1))) >> i-1;
27.             if(onum == 0) cout << i << endl;
28.         }
29.
30.         conv(num, 2147483656);
31.         for (; num¿=0;) num = num/2;
32.
33.         return num;
34.  }
```

Test de Primalidad Probabilistico

```
1.  bool RSA::Miller(ZZ p){
2.      int z;
3.      if (p != 2 && mod(p, (ZZ)2)==0) return false;
4.
5.      int iteration = 5;
6.      ZZ s = p - 1;
7.      while (mod(s, (ZZ)2) == 0) s = s >> 1;
8.      for (int i = 0; i < iteration; i++){
9.          conv(z, (p - 1) + 1);
10.         ZZ a = ZZ( mod(ZZ(rand()), ZZ(z))), temp = s;
11.         ZZ mod = exp_mod(a, temp, p);
12.         while (temp != p - 1 && mod != 1 && mod != p - 1){
13.             mod = mulmod(mod, mod, p);
14.             temp = temp << 1;
15.         }
16.         if (mod != p - 1 && mod(temp, 2) == 0) return false;
17.     }
18.     return true;
19. }
```

VI-B. Implementación del Rompimiento del RSA

Bueno en esta sección presentare el código de implementación para factorizar, el algoritmo que usare será el de la criba cuadrática.

Este algoritmo consta de partes especificas, por lo tanto se han separado en funciones para que se entiendan uno por uno

```
1.  bool esPrimo(long);
2.  long jacobi(long ,long );
3.  long func_exp(long );
4.  long *HallarBase(long );
5.  void addMatriz(long ,long );
6.  void CalcularX();
7.  long *factorizar(long );
8.  void imprimirVector(long *, long );
9.  void imprimirMatriz(long , long );
10. long long mcd(long long , long long );
11. bool obtenerSolucion();
12. void LiberarMemoria();
```

Claro que hay funciones muy conocidas y hechas en clase que no se pasaran a explicar como es el caso del mcd $(Euclides)$, $factorizar$, exp $(Exponenciacion\ Rapida)$, $esPrimo(Algoritmo\ de\ Miller)$, $ImprimirVector$, $ImprimirMatriz$, $LiberarMemoria$.

Jocobi Test

```
1.  long jacobi(long a,long n){
2.  long a1 = 1,n1 = 0,s,j,sj,e ,p;
3.  long b=2;
4.  if(a==0 || a == 1) return a;
5.  else{
6.      e = func_exp(a);
7.      p = (long)pow(2,e);
8.      a1= (long)(a/p);
9.      if(mod(e, 2)==0) s=1;
10.     else
11.         if((mod(n-1, 8)==0) || ( mod(n-7, 8)==0)) s=1;
12.         else
13.             if(( mod(n-3, 8)==0) || ( mod(n-5, 8)==0)) s=-1;
14.     if((mod(n-3,4)==0) && (mod(a1-3,4)==0)) s=-s;
15.     n1= mod(n,a1);
16.     if(a1==1) return s;
17.     else return s*jacobi(n1,a1);
18.     }
19. }
```

Continuamos con la siguiente funcion

HallarBase - Funcion para encontrar la base de los primos

```
1.  long* HallarBase(long n){
2.      long i,j;
3.      VectorBase[0] = -1;
```

```
4.      VectorBase[1] = 2;
5.      for(i=3,j=2;i<MaxBase;i++)
6.          if(esPrimo(i)==true)
7.              if(jacobi(n,i)==1)
8.                  VectorBase[j++]=i;
9.      tamBase=j;
10. }
```

Continuamos con la siguiente funcion

CalcularX - Calcula los x+m

```
1.  void CalcularX(){
2.      long raiz=(long)sqrt(n);
3.      long i=raiz-MAxAleat;// i es el rango menor de la criba
4.      long j=raiz+MAxAleat;// j es el rango mayor de la criba
5.      long h=0,w=0,y;
6.      while(i!=j){
7.          long *factores = new long[tamBase+1];
8.          y=(i*i)-n;// (x+m)^2 - n
9.          if(y!=0){
10.             factores = factorizar(y);
11.             if(factores[0]==1){
12.                 VectorX[h]=i;
13.                 fx[h]=y;
14.                 addMatriz(factores,1,h);
15.                 h++;
16.             }
17.         }
18.     i++;
19.     }
20.     tamX = h;
21. }
```

Continuamos con la siguiente funcion

obtenerSolucion - Empezamos a generar la matriz que necesimos.

```
1.  bool obtenerSolucion(){
2.      int q=0;
3.      bool band=false;
4.      Sol=new long[tamX];
5.      long *aux2=new long[tamBase];
6.      int cont;
7.
8.      MatrizT = new long*[tamBase];
9.      for(int i = 0; i < tamBase; i++)
```

```
10.       MatrizT[i] = new long[tamX];
11.
12.       /*obtenemos la transpuesta*/
13.       for(long i=0; i<tamBase; i++)
14.       for(long j=0; j<tamX; j++)
15.       MatrizT[i][j]=Matriz[j][i];
16.       while(band==false||q!=100000){
17.             /*generar matriz aleatoria*/
18.             for(int i=0;i<tamX;i++)
19.                   Sol[i]=mod(rand(), 2);
20.             int aux=0;
21.             for(long i=0; i<tamBase; i++){
22.                   for(long j=0; j<tamX; j++)
23.                         aux+=MatrizT[i][j]*Sol[j];
24.                   aux2[i]=mod(aux, 2);
25.             }
26.             cont=0;
27.             /*verificamos que el vector final sea cero*/
28.             for(int i=0;i<tamBase;i++)
29.                   if(aux2[i]!=0)
30.                         cont++;
31.             if(cont==0){
32.                   band =true;
33.                   return true;
34.             }
35.       q++;
36.       }
37.       //si no se encuentra soluciones luego de 10000 numeros aleatorios retorna no hay solucion
38.       if(q<=100000) return false;
39. }
```