Generación de números Primos

Universidad Católica San Pablo
Kevin Jhomar Sanchez Sanchez

ÍNDICE

I. Introducción

Dentro del mundo de la criptografia hay números muy especiales, llamados numeros primos, y es en este punto donde comienza nuestra investigacion.

Mostrare como se genera un numero primo fuerte y veremos como es que se prueba que un numero n es un numero primo.

II. Números primos

II-A. Definición

Un número primo es un número natural mayor que 1, que tiene únicamente dos divisores distintos: él mismo y el 1.

En el campo de teoría de los números hay una parte donde se especializa solo en el estudio de los números primos, en estas ramas se estudian básicamente todas sus propiedades aritméticas.

Algo interesante de ver es que si se consideran números individuales, los primos parecen estar distribuidos aleatoriamente, pero la distribución «global» de los números primos sigue leyes bien definidas.

NOTA: El número 1, por convenio, no se considera ni primo ni compuesto, y la propiedad de ser primo se denomina **primalidad**.

II-B. Generación de primos fuertes

Un número primo fuerte es un número con las siguientes propiedades.

Propiedades

1. p es un número primo.
2. $p-1$ tiene factores primos grandes.
   Es decir, $p = a_1 q_1 + 1$ para algún entero a_1 y un primo grande q_1.
3. $q_1 - 1$ tiene factores primos grandes.
   Es decir,$q_1 = a_2 q_2 + 1$ para algún entero a_2 y un primo grande q_2.
4. $p+1$ tiene factores grandes.
   Es decir $p = a_3 q_3 - 1$ para algún entero a_3 y un primo grande q_3.

Ejemplo. Si q_1 = 11, a_1 = 2, entonces lo introducimos a $p = a_1 q_1 + 1$ y resolviendo p = 2*11 + 1 = 23.

II-C. La Primalidad

La primalidad consiste en determinar, de forma aleatoria, números primos grandes. El problema surge, entre otras razones porque dentro del criptosistema RSA se necesita implementar, dos números primos p y q, de longitud grande (de mas de 100 dígitos).

En general, para resolver el problema se recurre a los test de primalidad y pseudoprimalidad, Un test de primalidad es un criterio para decidir si un numero dado es o no primo, que esta dentro de una matriz o dentro de un rango. [8]

III. Test de Pseudoprimalidad

III-A. Teorema de Tchebycheff

Esta ecuacion fue demostrado en primer lugar por Hadamard y de la Vallée Poussin en 1896 basandose en algunas propiedades de la funcion Zeta de Riemann.

Este teorema calcula la cantidad de números primos menores o iguales que x, $\pi(x)$, que es asintoticamente equivalente a $x/ln(x)$, es decir:

$$\lim_{n\to\infty} \frac{\pi(x)}{\frac{x}{ln(x)}} = 1$$

Cuadro grafico del algoritmo

x	π(x)	Li(x)-π(x)	θ(x)
10^6	78,498	129	-0.209
10^7	664,579	336	-0.1809
10^8	5,761,455	753	-0.1339
10^9	50,847,534	1,700	-0.0994
10^{10}	455,052,511	3,103	-0.0594
10^{11}	4,118,054,813	11,587	-0.0725
10^{12}	37,607,912,018	38,262	-0.0781
10^{13}	346,065,536,839	108,970	-0.0723
10^{14}	3,204,941,750,802	314,889	-0.0678
10^{15}	29,844,570,422,669	1,052,618	-0.0735
10^{16}	279,238,341,033,925	3,214,631	-0.0726
10^{17}	2,623,557,157,654,233	7,956,588	-0.0581
10^{18}	24,739,954,287,740,860	21,949,554	-0.05177
10^{19}	234,057,667,276,344,607	99,877,774	-0.0760
10^{20}	2,220,819,602,560,918,840	222,744,643	-0.0546

Figura 1. Tabla del teorema de Tchebycheff

Donde: x es el número de bits y $\pi(x)$ es la cantidad de primos que hay en el rango de x

III-B. *El Pequeño Teorema de Fermat*

Con este teorema se diferenciar a un numero primo de un número compuesto. [6]

Donde para todo p que es un número primo.

1. Si $MCD(a,p) = 1$, entonces $a^{(p-1)} \equiv 1(mod\ p)$
2. Si $r \equiv s(mod\ p-1)$, entonces $a^r \equiv a^s(mod\ p)$ para todos los enteros a. En otras palabras, cuando se trabaja a un numero primo p los exponentes pueden ser reducidos a $p-1$
3. De una forma particular $a^p \equiv a(mod\ p)$ para todo entero a

Ejemplo. Consideraremos el numero n = 91. Este numero es un pseudoprimo de base b=3, puesto que $3^{90} \equiv 1(mod\ 91)$; sin embargo ,91 no es pseudoprimo de base b = 2, puesto que $2^{90} \equiv 64(mod\ 91)$.

III-C. *Teorema de Mersenne*

Si $2^n - 1$ es primo, entonces n es primo. Si n es primo y $2^n - 1$, entonces $M_n = 2^n - 1$ se llama primo de Mersenne.

No se sabe si existen infinitos primos de Mersenne. El ultimo conocido es $M_{74207281} = 2^{74207281} - 1$ de 49 números primos.

IV. Test de Primalidad

Es un criterio para decidir si un numero dado es o no primo, que esta dentro de una matriz o dentro de un rango. El test de primalidad internamente tiene algoritmos que generalmente se dividen en dos ramas muy claras y estas son:

1. **Algoritmos deterministas**. son algoritmos completamente predictivos, donde si de conocen sus entradas, también conoceríamos su salida.
2. **Algoritmos probabilistas**. son algoritmos que basan sus resultados en la toma de algunas decisiones al azar, de tal forma que, en promedio, obtiene una buena solución al problema planteado para cualquier distribución de los datos de entrada, claro que se tiene un margen de error al momento de la generación de números primos.

IV-A. *Teorema de Wilson*

Este teorema(también llamado Congruencia de Wilson) se enuncia de la siguiente manera. [5]

Si p es un numero primo, entonces
$(p-1)! \equiv -1(mod\ p)$

El teorema de Wilson no se utiliza como test de primalidad en la práctica, ya que para calcular $(p-1)!\ mod\ n$ para un número n grande es costoso (computacionalmente hablando), y se conocen test más sencillos y rápidos. Gráficamente podemos verlo de esta manera.

Tabla de resto módulo *n*

n	$(n-1)!$	$(n-1)!\ \bmod\ n$
2	1	1
3	2	2
4	6	2
5	24	4
6	120	0
7	720	6
8	5040	0
9	40320	0
10	362880	0
11	3628800	10
12	39916800	0
13	479001600	12
14	6227020800	0
15	87178291200	0
16	1307674368000	0
17	20922789888000	16
18	355687428096000	0
19	6402373705728000	18
20	121645100408832000	0
21	2432902008176640000	0
22	51090942171709440000	0
23	1124000727777607680000	22

Figura 2. Tabla del teorema de wilson

Usando el teorema del Wilson, se tiene que para cada número primo p:

$1 \cdot 2 \cdots (p-1) \equiv -1(mod\ p)$

$1 \cdot (p-1) \cdot 2 \cdot (p-2) \cdots n \cdot (p-n) \equiv 1 \cdot (-1) \cdot 2 \cdot (-2) \cdots n \cdot (-n) \equiv -1(mod\ p)$

donde $p = 2n+1$. Esto se convierte en:

$$\prod_{j=1}^{n} j^2 \equiv (-1)^{n+1}\ (mod\ p)$$

o también

$(n!)2 \equiv (-1)^{m+1}\ (mod\ p)$

El teorema de Wilson ha sido utilizado para generar fórmulas para los primos, pero es demasiado lento como para tener valor práctico.

IV-B. *Criba de Eratostenes*

La criba de eratostenes es una algoritmo empleado para hallar números primos menores o igual a un n natural. [4]

Comenzando por el número 2 se tachan todos sus múltiplos, en la siguiente iteración se toma el siguiente menor numero de la criba y se tachan todos sus múltiplos y así hasta que el número primo al cuadrado sea mayor al n

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Figura 3. Criba de eratostenes, donde n=120

La complejidad Algorítmica de la criba es $O(N \ (log \ log \ N))$, esta complejidad es muy problemática al entender que para números mayores a 20 bits es muy lento(hablando desde un aspecto computacional).

IV-C. Criba de Atkin

Es un algoritmo diseñado por A.O.L. Atkin y Daniel J. Bernstein es rápido y moderno, empleado para hallar números primos menores o igual a un n natural.

En realidad es una versión mas optimizada de la criba de eratóstenes, pero a diferencia de eratóstenes, este algoritmo realiza trabajos preliminares y no tacha los múltiplos de los números primos como lo hace eratóstenes, sino que mas bien tacha los cuadrados de los primos múltiplos.

El algoritmo desde un inicio marca una liste de resultados, que normalmente esta compuesta por 2,3 y 5. También se crea una lista con cada entero positivo y cada casilla debe estar marcado desde un inicio como «no primo». Entonces para cada entrada de la lista de criba se realizan los siguientes pasos:

1. Si la entrada es un numero con resto 1, 13, 17, 29, 37, 41, 49 ó 53, se invierte tantas veces como soluciones posibles hay para $4x^2 + y^2$ = entrada
2. Si la entrada es un número con resto 7, 19, 31 ó 43, se invierte tantas veces como soluciones posibles hay para $3x^2 + y^2$= entrada.
3. Si la entrada es un número con resto 11, 23, 47 ó 59, se invierte tantas veces como soluciones posibles hay para $3x^2 - y^2$ = entrada con la restricción $x > y$.
4. Si la entrada tiene otro resto, se ignora.

Después de estos pasos se toma el menor de la lista de la criba y se toma al numero de la lista marcado como primo, después se incluyen el numero en la lista de resultados. Se eleva el número al cuadrado y se marcan todos los múltiplos de ese cuadrado como «no primos». Después hay que repetir estos últimos pasos. La complejidad computacional esta dada por

$$O(N/log \ log \ N)$$

con solo $N^{1/2+o(1)}$ bits de memoria, lo que lo hace ligeramente mejor que la criba de eratóstenes que su complejidad algorítmica es $O(N^{1/2}(log \ log \ N)/log \ N)$ bits de memoria.

De manera mas gráfica podemos verlo así

$$4x^2 + y^2$$

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

Figura 4. Tabla de ejemplo de la primera restricción del algoritmo

$$3x^2 + y^2$$

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

Figura 5. Tabla de ejemplo de la segunda restricción del algoritmo

$$3x^2 - y^2$$

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

Figura 6. Tabla de ejemplo de la tercera restricción del algoritmo

IV-D. Test de Solovay-Strassen

Supongamos que n es un entero positivo impar y queremos saber si n es un numero primo o compuesto. Para ello se eligen k enteros $0 < b < n$ aleatoriamente. Para cada uno de estos números b se calculan los valores de $b^{(n-1)/2}$ y de $\frac{b}{n}$. [2]

Si estos valores no son congruentes modulo n, entonces n es un numero compuesto y el test se detiene. en otro caso, se prueba el siguiente valor de b. Si los valores aleatoriamente calculados son congruentes para k valores de b, independientemente elegidos, entonces hay una probabilidad menor a 2^{-k} de que n no sea primo.

Si n es un numero entero par y b es un entero con $mcd(n, b) = 1$ y se verifica la congruencia anterior, entonces n se llama un pseudoprimo de Euler base b.

Donde:

$$\left(\frac{b}{n}\right) = \begin{cases} 0 \ si \ n \ divide \ a \ b \\ 1 \ si \ b \ es \ un \ residuo \ cuadratico \ modulo \ n \\ -1 \ si \ b \ no \ es \ un \ residuo \ cuadratico \ modulo \ n \end{cases} \quad (1)$$

Sea $a \in Z$ y p es primo, $p > 2$, diremos que a es un residuo cuadrático $(mod\ p)$, si y solo si existe $x \in Z$ tal que $x \equiv a(p)$.

Por lo que para cualquier número n que sea entero e impar, la probabilidad de error es de $(\frac{1}{2})^t$

IV-E. Teorema de Miller-Rabin

La primera propuesta fue dada por G. L. Miller, que desde un inicio se trataba de un algoritmo determinista de Riemann. Lo que hizo Oser Rabin fue modificar la propuesta de Miller y de esta modificación obtuvo un algoritmo probabilístico. [7]

Dado n un primo mayor a un numero impar del cual queremos saber si es primo o no. Sea m un valor impar tal que $n - 1 = 2^k m$ y a tal que $2 \leq a \leq n - 2$, Cuando se cumple:

$$a^m \equiv 1 \ mod \ n$$
$$\text{o}$$
$$a^{2m} \equiv -1 \ mod \ n$$

Para al menos un r entero tal que $1 \leq r \leq k - 1$, se considera que n es un probable primo; en caso contrario n no puede ser primo.

Por lo que para cualquier número n que sea entero e impar, la probabilidad de error es de $(\frac{1}{4})^t$

Si comparamos este algoritmo con el anterior(Solovay-Strassen), tendremos que Miller - Rabin es mucho mejor en varios aspectos, y las razones son que:

1. Solovay-Strassen es mucho mas caro(computacionalmente hablando) que Miller-Rabin.
2. También que la probabilidad de error de Solovay-Strassen ($(\frac{1}{2})^t$) es mucho mas alta que la de Miller-Rabin ($(\frac{1}{4})^t$)

IV-F. Algoritmo de Williams/Schmid

En 1979, Williams y Schmid propusieron el siguiente algoritmo, es efectivo y altamente recomendado para generar primos fuertes.

1. Encontrar dos primos P'' y P^+ del tamaño de bits elegido.
2. Calcular $R = -(P'')^{-1} mod\ P^+$ Por tanto $0 < r < P^+$, La inversa de P'' en modulo P^m puede ser calculada mediante el algoritmo extendido de Euclides.
3. Encontrar el menor A tal que
$$P' = 2AP''P^+ + 2RP'' + 1, y$$
$$P = 4AP^nP^+ + 4AP'' + 3$$
O encontrado P'
$$P = 2P' + 1$$
Hasta que sean primos

IV-G. Algoritmo de Gordon

En 1984, John Gordon y propuso un nuevo algoritmo para generar primos fuertes, este es un poco mas eficiente que el de Williams/Schmid, debido a que no necesita calcular un P' fuerte.

Algoritmo

1. Encontrar dos primos P'' y P^+ del tamaño de bits elegido.
2. Calcular el menor primo de la forma $P' = A''P'' + 1$ para algún entero A''
3. Dejar
$$P_0 = ((P^+)^{P'-1} - (P')^{P^+-1})(P'P^{++})$$
Notar que esto por el teorema de Fermat implicaría que $P' \equiv 1(mod\ P')$ y $P' \equiv -1(mod\ p+)$
4. Calcular el menor primo P de la siguiente manera
$$P_0 = +AP'P^+$$
para algun entero A.

IV-H. Espiral de Ulam

La espiral de Ulam, descrita por el matemático polaco-estadounidense Stanis law Marcin Ulam (1909-1984), es una forma de representación gráfica de números primos que muestra un patrón.

Todos los números primos, excepto el 2, son impares. Como en la espiral de Ulam algunas diagonales contienen números impares y otras contienen números pares, no sorprende ver como los números primos caen todos (salvo el 2) en diagonales alternas. Sin embargo, entre las diagonales que contienen números impares, unas contienen una proporción visiblemente mayor que otras de números primos.

Las pruebas que se han hecho hasta ahora confirman que: incluso si se extiende mucho la espiral, se siguen mostrando esas diagonales; El patrón se muestra igualmente aunque el numero central no sea 1 (en efecto, puede ser mucho mayor que 1). Esto significa que hay muchas constantes enteras b y c tales que la función:

$$f(n) = 4n^2 + bn + c$$

IV-I. Test de Lucas

M_p es primo si y solo si, $\frac{U_{2^p}}{U_{2^p-1}} \equiv 0(mod\ M_p)$

Con U_n = termino n-ésimo de la sucesión de Fibonacci Sucesión de Fibonacci:{1, 1, 2, 3, 5, 8, 13, 21,...}

IV-J. Test de Jacobi Sum

Es un test basada en la idea de congruencia de conjuntos, cada uno con una analogía a el teorema de Fermat. El tiempo de ejecución determinado por un entero n es

$$O((ln\ n)^{c\ ln\ ln\ ln\ n})$$

en operaciones de bits por algún constante c. Este es casi un algoritmo polinómico ya que el exponente $ln(ln(ln(n)))$ actúa como una una constante para un rango de valores n de interés. [3]

Ejemplo. Si $n \leq 2^{512}$, entonces $ln(ln(ln(n))) <$ 1,78.

La versión de Jacobi, usado en la practica es un algoritmo aleatorio que termina dentro de $O((ln\ n)^{c\ ln\ ln\ ln\ n})$ pasos con una probabilidad de al menos $1 - (\frac{1}{2})^k$ por cada $k \geq 1$ y siempre tienes una respuesta correcta.

Un inconveniente del algoritmo es que no produce un «certificado» que permitiría a la respuesta verificarse en el tiempo mucho más corto que ejecuta el propio algoritmo.

El test de Jacobi sum es, de hecho, práctico en el sentido de que la primalidad de los números que son varios cientos de dígitos decimales larga se pueden manejar en tan sólo unos minutos en un equipo. Sin embargo, la prueba no es tan fácil de programar como la prueba de Miller-Rabin probabilístico, y el código resultante no es tan compacto. Los detalles del algoritmo son complicados .

V. Conclusión

Para mi conclusión terminaré escogiendo un test pero antes de eso tengo que presentar los dos posibles candidatos que me han atraído mucho, estos son el test de Jacobi Sum y el teorema de Miller-Rabin.

El test de Jacobi Sum, me interesa mucho porque la complejidad algorítmica que traza este algoritmo que es de:

$$O((ln\ n)^{c\ ln\ ln\ ln\ n})$$

Esto significa que la prueba de Jacobi sum resultante es bastante eficiente y logra los resultados de forma muy rápida a comparación de los otros algoritmos que hemos visto.

Pero cabe recalcar como ya hemos dicho anterior mente que el test de Jacobi Sum no proporciona un certificado de que el n(el número a probar) sea un numero primo al 100 %, por lo que la única manera de comprobar computacionalmente es volverlo a realizar el mismo proceso de comprobación varias veces para tener un rango mas alto de probabilidad de que es un número primo.

Bueno, pero desde el punto de vista del teorema de Miller - Rabin parece ser mucho más rápido que el de Jacobi sum, pero mejor veamoslo en la siguiente tabla:

$\log_2 N$	Jacobi sums	Rabin-Miller
64	0.06 s	0.06 s
128	0.16 s	0.15 s
256	2.09 s	0.75 s
512	37.6 s	5.02 s
1024	907 s	37.1 s

Figura 7. Test de Jacobi sum y Rabin Miller

Esta tabla nos presenta una progresión algorítmica muy significativa. Para ser honesto cuando descubrí esto me que de atónito ya que pensaba que Jacobi test era mucho mejor. [1]

Sucede que dentro de la complejidad algorítmica de Jacobi test $(ln\ n)^c$ podemos apreciar que c vendría a ser uno de los primeros problemas que se presentan para este algoritmo ya que ya sea cualquier base que tenga siempre se va a elevar c veces.

Para un entendimiento mejor de mi punto de vista esta esta gráfica:

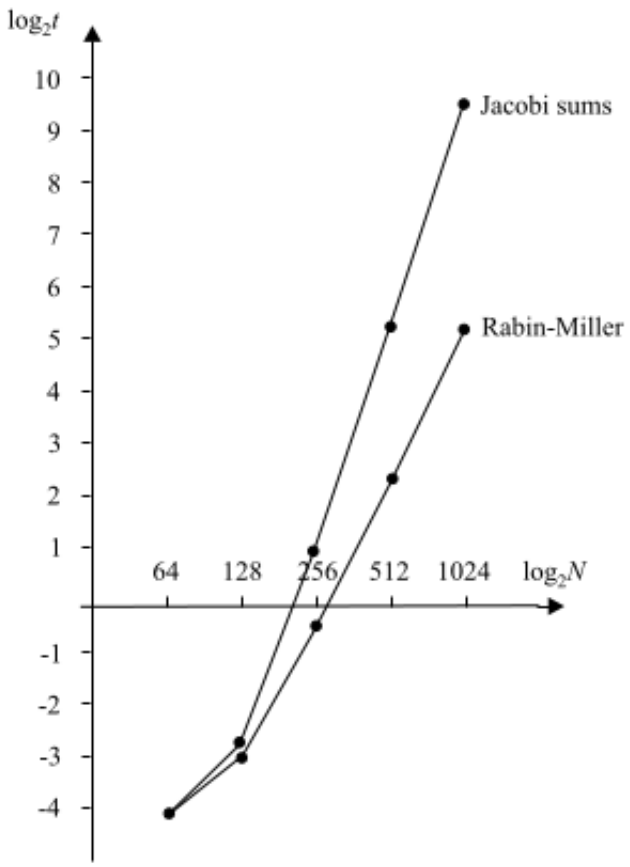

Figura 8. Comparaciones de tiempo de respuesta.

Desde esta gráfica comparativa podemos ver mucho mejor como es que la velocidad influyen en ambos algoritmos.

Sin duda me quedo con el algritmo de Miller - Rabin por razones muy obvias.

REFERENCIAS

[1] Andrzej Chmielowiec. Primality proving with gauss and jacobi sums. *Primality proving with Gauss and Jacobi sums*, 42, 72-75.

[2] Ashok K. Chandra George Markowsky. On the number of prime implicants. *On the number of prime implicants*, 1977.

[3] Shaft Goldwasser J. C. Lagarias Arjen K. Lenstra Kevin S. McCurley A. M. Odlysko. Proceedings of symposia in applied mathematics. *Cryptology and Computational Number Theory*, 42, 19-30.

[4] Marc Joye Pascal Paillier. Constructive methods for the generation of prime numbers. *Constructive Methods for the Generation of Prime Numbers*, 2001.

[5] William Stallings. Cryptography and network security. *Cryptography and Network Security*, 4to Edicion:245–251, 2005.

[6] Alfred J.Menezes Paul C.van Oorschot Scott A. Vanstone. Handbook of applied cryptography. *Introduction, Information security and cryptography*, 4to Edicion:133–149, 1996.

[7] Marc Joye Pascal Paillier Serge Vaudenay. Efficient generation of prime numbers. *Efficient Generation of Prime Numbers*, 2000.

[8] Marcela Wilder. Test de primalidad, aplicación a la criptografía. *Test de Primalidad, aplicación a la criptografía*, pages 44–46.

REFERENCIAS

[1] Andrzej Chmielowiec. Primality proving with gauss and jacobi sums. *Primality proving with Gauss and Jacobi sums*, 42, 72-75.

[2] Ashok K. Chandra George Markowsky. On the number of prime implicants. *On the number of prime implicants*, 1977.

[3] Shaft Goldwasser J. C. Lagarias Arjen K. Lenstra Kevin S. McCurley A. M. Odlysko. Proceedings of symposia in applied mathematics. *Cryptology and Computational Number Theory*, 42, 19-30.

[4] Marc Joye Pascal Paillier. Constructive methods for the generation of prime numbers. *Constructive Methods for the Generation of Prime Numbers*, 2001.

[5] William Stallings. Cryptography and network security. *Cryptography and Network Security*, 4to Edicion:245–251, 2005.

[6] Alfred J.Menezes Paul C.van Oorschot Scott A. Vanstone. Handbook of applied cryptography. *Introduction, Information security and cryptography*, 4to Edicion:133–149, 1996.

[7] Marc Joye Pascal Paillier Serge Vaudenay. Efficient generation of prime numbers. *Efficient Generation of Prime Numbers*, 2000.

[8] Marcela Wilder. Test de primalidad, aplicación a la criptografía. *Test de Primalidad, aplicación a la criptografía*, pages 44–46.