Test de primalidad para sistemas criptográficos

Miguel Ascaso Nerín

Trabajo de fin del grado de Matemáticas
Universidad de Zaragoza

Summary

0.1. Introduction

My main objective in this work was to present historical journey through the different cryptographic systems and problems that they solve or arise, finishing by presenting the latest development in the area of primality.

A review of both the origins of cryptography and the major cryptographic methods currently used is included, while highlighting the need for primes with a high number of digits to ensure insolubility. Both the RSA method, which was the first one to use asymmetric keys for encryption and decryption of messages, and the Diffie - Hellman method, based on the complexity of the discrete logarithm resolution, serve as examples for this purpose.

0.2. Primes

The concept of prime number and severability is particularly relevant:

Definition 0.2.1. *Given $m,n \in \mathbb{N}$ we may say m divides n when $\exists r \in \mathbb{N}$ such that $n = r \cdot m$.*

Definition 0.2.2. *Given $p \in \mathbb{N}$, $p \neq 1$ we may say p is **prime** if its only divisors are 1 and p.*

These common and universally known numbers hide behind their innocent appearance many unsolved problems. A review of the main theorems which represented a significant advance in this field of primality will be presented, such as:

Theorem 0.2.3 (Euclid)**.** *There are infinite prime numbers.*

And its important consequence:

Corollary 0.2.4. *There are infinitely large prime numbers.*

The problem of finding all prime numbers has been open for more than 2500 years and not few solutions have been proposed, none actually successful given the apparent randomness with which these famous numbers appear. However there are some trends worth noting:

Theorem 0.2.5 (of prime numbers)**.** *Being $\pi(x)$ the amount of prime numbers lower or equal to x then, when x grows, we have:*

$$\pi(x) \sim \frac{x}{\ln(x)}$$

This theorem allows us to begin the search for large primes with the certainty that we will find enough of them to establish secure cryptographic systems.

0.3. Algorithmic complexity

The biggest a number is, the more difficult it is to know if that number is prime, also, the larger it is the prime we seek, the more difficult it becomes to find it. The time it takes a computer to perform calculations is particularly relevant when trying to use on to find primes with a high number of digits. A study of the asymptotic behavior of the algorithms depending on the number entered will be presented.

Definition 0.3.1. *Given* $f : \mathbb{N} \to [0,\infty)$ *We note:*

$$O(f) = \{g : \mathbb{N} \to [0,\infty) \mid \exists c \in \mathbb{R},\ c > 0, n_0 \in \mathbb{N} \text{ such as } g(n) \leq cf(n)\ \forall n > n_0\}$$

Definition 0.3.2. *We say* f *is an algorithm solvable in polynomial time if there is a polynomial* p *such that* $f \in O(p)$.

Otherwise we may say that f *is an algorithm that requires exponential time.*

What really interests us is to see whether the algorithms are solvable in polynomial time using as a parameter the number of digits a number needs to be represented.

0.4. Primality test

Definition 0.4.1. ***Primality test*** *is the name given to a deterministic algorithm that decides whether a candidate to be prime actually is so, based on the fulfillment of certain properties by the candidate number.*

A review of the best primality tests at our disposal will be presented, along with its features, the numbers that the can be applied to and the results they offer. The test based on Fermat's little theorem, Miller - Rabin's test or Lucas - Lehmer's test are used for different purposes, as each of them has a series of advantages and disadvantages that will be presented in this work.

0.5. AKS

Finally, the AKS primality test and its features will be introduced, while doing the appropriate demonstration. The work will conclude with a small analysis of the impact that this result might have in finding large prime numbers.

Índice general

Scientia potentia est

Mucho antes de que el canciller inglés y filósofo Francis Bacon nos invitara a reflexionar con su frase 'El conocimiento es poder', incluso antes de que el hombre empezara a escribir, el valor de la información ya había sido puesto en relieve en multitud de ocasiones. Conocer la localización de un ejército enemigo o el número de efectivos del que dispone, conocer cuál es la época idónea para sembrar los cultivos y que estos germinen adecuadamente, conocer cual es la causa de una enfermedad y su posible tratamiento, conocer las fórmulas matemáticas que se imponen en el mundo, conocer marca la diferencia entre el éxito y el fracaso, la vida y la muerte, conocer nos predispone para actuar y evitar el miedo que genera el desconocimiento.

Este bien inmaterial tan preciado es un producto directo de la inteligencia, sin pensamiento no hay conocimiento. ¿Qué son las lineas de un libro si no hay nadie capaz de entenderlas?

Cada uno de nosotros tenemos la capacidad de recopilar información mediante nuestras experiencias y nuestros sentidos pero, si nos paramos a pensar, la mayor parte de la información que recibimos no viene de nosotros mismos si no de lo que los demás nos cuentan. Estamos al tanto de las protestas en Atenas, del deshielo de los casquetes polares, de los sucesos en la batalla del Alcoraz en el año 1096, de la forma casi esférica del planeta o del plan que tiene nuestro amigo el fin de semana, sin embargo no hemos sido testigos de ninguno de estos sucesos ni hemos hallado evidencias de su veracidad sin embargo, asimilamos toda esta información que nos ha sido trasmitida por diferentes **medios de comunicación**.

Para poder acumular toda la sabiduría de la que hoy disponemos ha hecho falta juntar y procesar toda la información recopilada por las personas a lo largo del tiempo y el espacio para luego trasmitirla a otras personas para que estas continúen añadiendo pequeños granitos de arena a esa inmensa montaña. Y es que esa es la principal característica de la información, que se puede trasmitir.

Es obligatorio realizar especial énfasis en el concepto de la comunicación y la necesidad, en la práctica, de establecer sistemas de llevarla a cabo. Desde la voz y el sonido hasta la transferencia de datos por internet pasando por las señales de humo, el telégrafo o la escritura el hombre ha ideado multitud de formas de transmitir información que han permitido ampliar enormemente la capacidad cognitiva de la sociedad.

Es de este proceso de transmisión del que nace el problema que de aquí en adelante vamos a tratar: ¿Es posible enviar información entre dos interlocutores de forma privada y segura frente a posibles espías?

Capítulo 1

Sistemas Criptográficos

1.1. Orígenes y objetivos de la criptografía

La información bien utilizada puede ser en ocasiones una herramienta muy valiosa y por lo tanto merecedora de atención y precauciones especiales. ¿Qué ocurre si queremos transmitir un mensaje de forma discreta y privada? La criptografía es la rama de las matemáticas encargada de la codificación de información que nace ante la necesidad de enviar información de forma secreta entre dos interlocutores ofreciendo ciertas garantías de que el mensaje no puede ser descifrado más que por el receptor escogido.

En la Biblia se cita un sistema de sustitución de letras llamado *Atbash* que se remonta al año 600 a.C.; en la *Iliada*, Homero nos cuenta como Belerofonte le entrega al rey Ióbates una carta cifrada por su homólogo Preto de Tirinto; en el siglo V a.C. los espartanos utilizaban un instrumento aproximadamente cilíndrico denominado *escítala* que servía para, enrollando una cinta con letras escritas alrededor de este aparato, escribir o leer un texto que variaba en función del diámetro de la escítala.

En el imperio romano Julio César necesitaba enviar mensajes de contenido bélico delicado y para ello utilizó el hoy denominado *cifrado César* que consiste en desplazar el alfabeto un número determinado de veces (la clave de cifrado-descifrado). Este método le permitió enviar sus órdenes de forma que aunque el mensajero fuese capturado por el enemigo éste no tendría forma de entender el contenido del mensaje.

Sin embargo estos método tienen muchos puntos débiles: hay muy pocas combinaciones posibles así que es factible probarlas todas, para que el receptor pueda descifrar el mensaje tiene que conocer la clave utilizada en la encriptación lo que obliga a enviar previamente un mensaje sin codificar o haberse reunido en persona emisor y receptor...

Por ejemplo, en la edad media el filósofo y científico árabe Al-Kindi realizó un concienzudo trabajo al analizar cuáles eran las letras y palabras que más aparecían en el Corán descubriendo así que en dicha obra existe una característica frecuencia de letras que se repetía en cualquier texto escrito en árabe. Nació de este modo el criptoanálisis y el denominado ataque por frecuencia utilizado para, sabiendo el idioma en el que está escrito el mensaje, intentar adivinar por la frecuencia de aparición de las letras del mensaje cifrado cual es el mensaje original.

Con el paso del tiempo se han ideado diversos sistemas criptográficos mas complejos que permiten la verificación de la identidad de los interlocutores y aseguran que ningún extraño ha accedido al contenido del mensaje original.

Con la entrada en escena de las matemáticas y especialmente los ordenadores como herramienta principal para codificar mensajes se ha tendido a transformar todo mensaje que se desee enviar en un mensaje numérico permitiendo una mayor manejabilidad de la información. Algunos de los tests más conocidos y utilizados son *advanced encryption standard (AES)* o el método *RSA*, los basados en el logaritmo discreto como *curvas elípticas*, *ElGamal* o *Diffie-Hellman*.

1.2. RSA

En 1977 Rivest, Shamir y Adleman (RSA) idearon el primer sistema criptográfico asimétrico (de clave pública) que utiliza un algoritmo cuyo funcionamiento es comparable a la siguiente situación:

Si Bob quiere entregarle un mensaje a Alicia pero nunca se han visto ni disponen de un medio seguro de comunicarse (y por lo tanto no han acordado ninguna clave secreta común) basta con que Alicia fabrique una caja con un cierre que se abre exclusivamente con una llave que ella misma posee. Alicia le manda a Bob la caja de seguridad abierta y cuando él la recibe, introduce el mensaje y la cierra de tal forma que solo Alicia, la propietaria de la llave, puede abrirla y acceder al mensaje que contiene.

Este algoritmo está muy extendido y es utilizado, por ejemplo, a la hora de realizar conexiones seguras a través de internet. No obstante, se trata de un método que requiere la realización de muchos cálculos por lo que en la práctica se utiliza sólo para intercambiar una clave que posteriormente se utilizará para comunicarse mediante otro sistema criptográfico simétrico (de clave privada).

Supongamos que Bob quiere enviar a Alicia un mensaje secreto que solo ella pueda leer, lo primero que ha de hacer es transformar mediante una identificación acordada el mensaje en un número natural m para poder manejarlo con las herramientas matemáticas de las que disponemos. Todos los mensajes, claves y códigos de los que vamos a hablar de aquí en adelante serán numéricos.

- Alicia elige dos números primos **p** y **q** distintos y muy grandes (de más de 100 dígitos).
- Calcula $n = pq$ y $\varphi(n) = (p-1)(q-1)$ ($\varphi(n)$ es la llamada función de Euler, que veremos más adelante).
- Escoge un entero positivo e menor y coprimo con $\varphi(n)$ (inversible en módulo $\varphi(n)$). Además Alicia calcula $d \in \mathbb{Z}_{\varphi(n)}$ que es el inverso de $e \mod \varphi(n)$
- Alicia manda a Bob un mensaje inseguro solamente con los números n y e. Llamamos a esta pareja de números (n,e) la clave pública de Alicia.
- Bob recibe dichos números y cifra su mensaje utilizando n y e para hallar su mensaje cifrado: $c \equiv m^e \mod n$
- Por último Bob envía c a Alicia sabiendo que su mensaje está cifrado de tal forma que solo Alicia es capaz de descifrarlo.
- Alicia calcula $c^d \equiv (m^e)^d \equiv m^{ed} \equiv m \mod n$

Luego Alicia recupera el mensaje original que Bob quería enviarle.

La seguridad de este algoritmo está basada en la dificultad de factorizar n como producto de primos cuando estos son muy grandes, el tiempo requerido por un ordenador actual para dicha tarea es inmenso por muy potente que sea su capacidad de cálculo.

Para realizar este proceso utilizamos en el primer paso dos números primos **muy grandes**. La seguridad del algoritmo reside en la magnitud de estos primos, cuanto más grandes, más complicado resulta factorizar su producto. Sin embargo es complicado encontrar números primos de gran tamaño. Dado un número de muchos bits, ¿cómo podemos estar seguros de que dicho número es o no primo? Al igual que se hace muy difícil descomponer n en primos, igualmente es difícil demostrar que no existe ninguna factorización no trivial de p. Para tratar este problema disponemos de los test de primalidad.

1.3. Diffie-Hellman

Otra alternativa interesante para intercambiar claves a distancia y de forma segura es el sistema de Diffie-Hellman.

En el método criptográfico inventado por Whitfield Diffie y Martin Hellman los interlocutores van a comunicarse por un medio no seguro de forma que aunque cualquier curioso tenga acceso a la información enviada, no podrá obtener la información que se quiere transmitir. Este sistema al igual que el RSA se suele utilizar para transmitir una clave que se pueda utilizar en un sistema simétrico.

Alicia quiere enviarle una clave secreta a Bob mediante el algoritmo Diffie-Hellman. Para ello Alicia escoge un número primo p, una clave pública g perteneciente al grupo multiplicativo de números enteros menores que p: $\mathbb{Z}_p^*$. Escoge también una clave secreta $a \in \mathbb{Z}_{p-1}$ siendo $\mathbb{Z}_{p-1}$ grupo abeliano con la suma esta vez, que sólo es conocida por Alicia, Bob por su parte escoge otra clave privada $b \in \mathbb{Z}_{p-1}$ y la guarda en secreto. Una vez se tienen todas las claves comienza el envío de información:

- Alicia calcula $A \equiv g^a \mod p$
- Alicia manda a Bob g, p y A
- Bob calcula $Z \equiv A^b \equiv (g^a)^b \equiv g^{ab} \mod p$ y $B \equiv g^b \mod p$
- Bob manda a Alicia B
- Por último Alicia calcula $Z \equiv B^a \equiv (g^b)^a \equiv g^{ab} \mod p$

Al final, Alicia y Bob obtienen Z que será su clave privada para futuras conversaciones. Pero, ¿pueden estar seguros de que nadie más ha podido calcular Z? Supongamos que Oscar ha interceptado todas las comunicaciones.

Alicia ha mandado por un canal inseguro g, p y $A \equiv g^a \mod p$, Oscar dispone pues de $g^a \mod p$, de p y de g, ¿le permite esto calcular a[1]?

Este problema es el denominado problema del logaritmo discreto.

$$a = logDis_g(A)$$

No se ha encontrado la manera de realizar, en general, en un tiempo aceptable este cálculo.

Cuando se escoge un número primo p lo suficientemente grande el número de operaciones necesarias para resolver esa ecuación se vuelve astronómico, cosa deseable para asegurar la privacidad de la información enviada. Por ejemplo, en el micro chip del DNI electrónico hay registrada información que nos permite realizar la firma digital mediante otro sistema criptográfico similar al RSA o Diffie-Hellman. Esta información no es otra cosa que una clave privada: dos números primos de gran tamaño que nos sirven como testigos de la veracidad de nuestra identidad ya que se suponen indescifrables precisamente por su gran longitud con lo que volvemos a encontrarnos con el reto de tener que encontrar un número primo de gran tamaño teniendo que recurrir de nuevo a los tests de primalidad.

[1] Análogo en el caso de querer encontrar b

Capítulo 2

¿Es primo?

2.1. Números primos

Hemos utilizado hasta aquí muchas veces el concepto de *número primo* pero convendría concretar un poco más a qué nos referimos con ese término. Para poder definir un número primo tenemos primero que definir qué es un divisor:

Definición 2.1.1. *Sean $m,n \in \mathbb{N}$ decimos que m divide a n cuando $\exists r \in \mathbb{N}$ tal que $n = r \cdot m$.*

Es equivalente decir que m **divide a** n o bien que m **es divisor de** n o bien m **es un factor de** n o bien n **es divisible por** m o bien $m|n$ o bien $n \equiv 0 \mod (m)$

Definición 2.1.2. *Sea $p \in \mathbb{N}$, $p \neq 1$ decimos que p es un **primo** si los únicos divisores que posee son 1 y p.*

Definición 2.1.3. *Un número $n \in \mathbb{N}$ se dice compuesto si no es primo.*

Definición 2.1.4. *Sea $p,q \in \mathbb{N}$, $p,q \neq 1$ decimos que p y q son **coprimos** si el único divisor común que poseen es el 1.*

Esta definición puede ampliarse para números en $\mathbb{Z}$ o cualquier otro dominio de integridad. De aquí en adelante consideraremos los números primos como números positivos y enteros o equivalentemente, el conjunto de los números primos $P \subset \mathbb{N}$

$\mathbb{N}$ es un conjunto infinito y podemos encontrar números naturales todo lo grandes que deseemos. Tal y como hemos visto, para poder aplicar con seguridad todos esos métodos criptográficos hacen falta dos números primos muy grandes, para encontrarlos debemos primero asegurarnos de que existen para lo que utilizaremos el antiguo (pero no por ello menos válido)

Teorema 2.1.5 (Euclides)**.** *Existen infinitos números primos.*

Demostración. Por reducción al absurdo: Supongamos que existe un número finito k de primos $p_1, p_2, ..., p_k \in \mathbb{N}$, definimos

$$Q_k = p_1 p_2 ... p_k + 1$$

por la definición anterior tenemos que $\forall i = 1, ..., k; p_i < Q_k$, como $Q_k \in \mathbb{N}$ pero según nuestra hipótesis Q_k no puede ser primo ya que es mayor que cualquiera de los números primos existentes luego se deduce que tiene que ser un número compuesto y por tanto $\exists i \in 1, ..., k$ tal que $p_i | Q_k$ pero entonces

$$p_i | p_1 p_2 ... p_k + 1$$

y esto implica que $p_i|1$ lo cual es absurdo ya que el único divisor de 1 es 1 y $p_i \neq 1$ por la definición de número primo.

Por tanto el conjunto de los números primos no puede tener tan solo k elementos y en consecuencia obtenemos que infinitos primos. □

Corolario 2.1.6. *Existen números primos infinitamente grandes.*

Queda claro que es posible encontrar primos lo suficientemente grandes como para aplicarlos en los métodos criptográficos anteriormente citados. Ahora bien, eso no excluye la posibilidad de que existan sólo primos muy alejados entre sí teniendo por ejemplo 20 dígitos de diferencia entre un primo y el siguiente. En dicho caso tendríamos un problema ya que seguramente sería sencillo descifrar las claves privadas de los sistemas criptográficos asimétricos basándose en la longitud de la clave pública. Por tanto, estamos obligados a estudiar la proporción de primos que encontramos conforme trabajamos con números más y más grandes.

La *criba de eratóstenes* consiste en escribir todos los números naturales hasta cierto n e ir tachando en esa lista todos los múltiplos de 2, luego de 3, luego del siguiente número que no se ha tachado, el 5, y así sucesivamente hasta que se llega a n. Obtenemos así todos los números primos menores o iguales que n. Este método evidencia que conforme avanzamos en la lista de los números naturales es más difícil encontrar un número primo.

A finales del siglo XVIII Gauss escribía en su cuaderno de notas la siguiente conjetura cuya demostración se demoró 100 años aproximadamente convirtiéndose en:

Teorema 2.1.7 (de los números primos)**.** *Sea $\pi(x)$ el número de primos menores o iguales a x entonces conforme x crece tenemos:*

$$\pi(x) \sim \frac{x}{\ln(x)}$$

Corolario 2.1.8. *La proporción de números menores o iguales a x que son primos es, cuando x es grande, aproximadamente*

$$\frac{1}{\ln(x)}$$

Queda confirmada la suposición de que los números primos disminuían en proporción conforme escogíamos magnitudes más grandes, sin embargo esta disminución es relativa ya que respecto al número de primos con un determinado número de cifras ocurre justo lo contrario como veremos a continuación:

La cantidad de números primos que podemos encontrar de n cifras es, según lo anterior, aproximadamente

$$M_n = \frac{10^n}{n\ln(10)} - \frac{10^{n-1}}{(n-1)\ln(10)} = \frac{(n-1)10^n - n10^{n-1}}{n(n-1)\ln(10)} = \frac{(9n-10)10^{n-1}}{(n^2-n)ln(10)}$$

y por tanto tenemos que

$$\lim_{n\to\infty} M_n = \infty$$

Ejemplo 2.1.9. *La cantidad de números primos que podemos encontrar de 100 cifras en base 10 es aproximadamente*

$$M_{100} = \frac{(900-10)10^{99}}{(100^2-100)ln(10)} \simeq 3{,}9042 \cdot 10^{97}$$

Unas $1{,}77 \cdot 10^{18}$ veces el número de partículas que se estima que hay en el universo.

Podemos considerar cualquier otra base y bastará sustituir el 10 de la fórmula anterior por la base deseada, no viéndose alterado el resultado. Es particularmente interesante la base 2 ya que estaríamos tratando con la longitud en bites de datos informáticos.

Ejemplo 2.1.10. *Podemos encontrar un ejemplo muy representativo en el DNI electrónico (ver [PDNI]). Como hemos comentado en el primer capítulo, en el micro chip que encontramos en nuestra tarjeta del DNI hay guardada una clave privada de 2048 bits, esto es, 2048 cifras en binario.*

Esta clave es ni más ni menos que el producto de dos números primos grandes. Aunque no debe ser así por motivos de seguridad, supondremos que se tratan de dos números de aproximadamente 1024 bits cada uno, ¿Cuántos números primos podemos obtener de ese tamaño?

$$\overline{M_{1024}} = \frac{1023 \cdot 2^{1024} - 1024 \cdot 2^{1023}}{1024 \cdot 1023 \cdot \ln(2)} = \frac{1022 \cdot 2^{1023}}{1037322 \cdot \ln(2)} \simeq 1{,}421 \cdot 10^{304{,}95}$$

Como consecuencia de esta observación podemos considerar que vamos a disponer de abundantes primos todo lo grandes que queramos para aplicar los métodos criptográficos.

Ahora es cuando llega el verdadero problema: encontrarlos. Pongámonos a ello inmediatamente, lo ideal sería disponer de una función $f(n)$ que tomara el valor del número primo n-ésimo, pero las cosas no suelen ser tan fáciles... ¿O sí? En [Gauss] encontramos una introducción al interesante resultado que nos ofrecía William H. Mills en 1947:

Teorema 2.1.11 (De Mills)**.** *Existe al menos un número real A tal que la función*

$$f(n) = \lfloor A^{3^n} \rfloor$$

es función generadora de primos.

Donde $\lfloor b \rfloor$ es la parte entera de b.

Desgraciadamente este resultado no va a facilitarnos la tarea ya que para el cálculo de los decimales de A (que vale aproximadamente 1,3063778838...) hace falta calcular ciertos números primos muy grandes. En definitiva, la pescadilla que se muerde la cola. Se han hallado los 7000 decimales *conocidos* de A gracías a la simplificación de cálculos que permite tomar por cierta la *Hipótesis de Riemann*

Conjetura 2.1.12 (Hipótesis de Riemann)**.** *La parte real de todo cero no trivial de la función zeta de Riemann es* $\frac{1}{2}$*. Siendo la función zeta (o dseta) de Riemann la siguiente*

$$\begin{aligned} \zeta : \mathbb{C} &\longrightarrow \mathbb{C} \\ s &\longrightarrow \zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} \end{aligned}$$

Desafortunadamente la *Hipótesis de Riemann* sigue siendo una conjetura desde hace más de 150 años a pesar de los múltiples intentos que ha habido de demostrarla así que tendremos que tomar otra senda: dado un número p, ¿podemos afirmar que dicho número es primo?

Si consideramos discernir la primalidad del número 107 podemos utilizar la *criba de Eratóstenes* o bien comprobar si tiene algún divisor entero entre el 2 y $\lfloor \sqrt{107} \rfloor$. Estos métodos son dos ejemplos básicos de los llamados tests de primalidad.

Pero, ¿qué ocurre cuando el número con el que tenemos que trabajar es el 1.647.382.901.873? El tiempo que nos llevaría analizar la primalidad de esta nueva cifra utilizando alguno de los anteriores métodos citados sería desorbitado, hay que buscar otros métodos más rápidos.

Pero, ¿qué quiere decir exactamente *rápidos*?

2.2. Coste computacional

Sacar papel y lápiz para empezar a hacer cálculos no es la mejor herramienta de la que disponemos hoy en día, todos los tests de los que hablamos están pensados para ser implementados en un ordenador. La forma de calcular la rapidez de un algoritmo consiste básicamente en calcular de forma asintótica el número de operaciones necesarias para terminar el proceso, en nuestro caso, decidir si un número es primo o no lo es.

A los algoritmos informáticos se acostumbra a suministrarles datos a partir de los cuales el ordenador puede realizar las operaciones pertinentes, el número de operaciones dependerá entonces del número n introducido. En general lo que nos interesa es acotar superiormente dicha cifra en función del tamaño del parámetro que introduzcamos, más concretamente, vamos a estudiar qué ocurre cuando n se hace muy grande.

Definición 2.2.1. *Sea $f : \mathbb{N} \to [0,\infty)$ Se define el conjunto de funciones de orden O de f como:*

$$O(f) = \{g : \mathbb{N} \to [0,\infty) \mid \exists c \in \mathbb{R},\ c > 0, n_0 \in \mathbb{N} \text{ tal que } g(n) \leq cf(n)\ \forall n > n_0\}$$

Nótese que si $g \in O(f)$ entonces $2 \cdot g \in O(f)$ y que si $g \in O(f)$, entonces $O(g) \subset O(f)$ pero no necesariamente se tiene que $f \in O(g)$

Definición 2.2.2. *Decimos que f es un algoritmo resoluble en tiempo polinomial si existe un polinomio p tal que $f \in O(p)$.*

En caso contrario diremos que f es un algoritmo que requiere un tiempo exponencial.

Ejemplo 2.2.3. *Dado $n \in \mathbb{N}$ cualquiera, queremos verificar si n es primo mediante el test de primalidad de las divisiones sucesivas visto anteriormente. Para implementar dicho método habría que programar un algoritmo, que denominaremos $T(n)$ similar al siguiente:*

```
mientras i < sqrt(n)
   si resto de dividir i entre n es 0 entonces
      devolver 'n es compuesto'
   fin si
fin mientras
devolver 'n es primo'
```

Como lo que nos interesa es dar una cota máxima para el número de operaciones necesarias del algoritmo vamos a suponer que nos encontramos en la situación más costosa, que en este caso será cuando n sea primo. En dicho supuesto, habrá que hacer una división por cada valor de i desde 2 hasta $\lfloor\sqrt{n}\rfloor$, por tanto habrá que realizar como mucho

$$\lfloor\sqrt{n}\rfloor - 2 \text{ divisiones}$$

Diremos que la complejidad algorítmica de $T(n) \in O(\sqrt{n})$

Si afinamos un poco más, podemos modificar este algoritmo y dividir por solo los números impares lo que hará que tan solo necesitemos $\dfrac{\lfloor\sqrt{n}\rfloor - 2}{2}$ divisiones.

O mejor aún, suponiendo que conocemos todos los primos menores que n podemos dividir simplemente por todos los que sean menores de $\sqrt{n}$ con lo que el algoritmo acabará como mucho tras $\dfrac{\lfloor\sqrt{n}\rfloor}{\ln(n)}$ operaciones.

Sin embargo, los tres algoritmos pertenecen al orden $O(\sqrt{n})$ y cuando el número al que queremos aplicar el test es muy grande, todos los algoritmos tendrán que realizar un número similar de cálculos.

Sea k el número de cifras de n (en base 10, por comodidad, aunque el sentido de esta notación viene por la longitud en bits que necesita un ordenador para escribir en base 2 los números candidatos), podemos escribir

$$n = a_k 10^k + a_{k-1} 10^{k-1} + \ldots + a_1 10 + a_0$$

y entonces el orden de complejidad de

$$T(n) = T(a_k 10^k + a_{k-1} 10^{k-1} + \ldots + a_1 10 + a_0) = \overline{T}(k)$$

y cuando queramos trabajar directamente con el número de cifras tendremos

$$O(\overline{T}(k)) = \overline{O}\left(\sqrt{a_k 10^k + a_{k-1}10^{k-1} + ... + a_1 10 + a_0}\right) = \overline{O}(10^{\frac{k}{2}})$$

En este caso estamos ante un algoritmo resoluble en ***tiempo polinomial*** *pero de* ***tiempo exponencial*** *siempre que nos refiramos al número de cifras. Esto no es deseable para nuestros propósitos ya que queremos que el algoritmo trabaje rápidamente, o sea, en tiempo polinomial, cuando aumentemos el número de cifras.*

Si $f \in O(\log(n))$ entonces $f \in \overline{O}(\log(10^k)) \rightarrow f \in \overline{O}(k)$. En consecuencia, lo ideal sería encontrar un algoritmo que tuviera un orden de complejidad similar al del logaritmo o equivalentemente, polinomial cuando consideramos como parámetro el **número de cifras**.

En general, el problema de la factorización es un problema resoluble en tiempo exponencial respecto al número de cifras, de ese hecho nace la seguridad de los sistemas criptográficos.

Capítulo 3

Tests de primalidad

Definición 3.0.4. *Se llama **test de primalidad** a un algoritmo determinista que decide si un número candidato a ser primo lo es, basándose en el cumplimiento de ciertas propiedades por parte del número candidato.*

3.1. Test de pseudoprimalidad

Dado que los test de primalidad son muy costosos cuando crece la magnitud de los candidatos a primos, podemos relajar las exigencias de determinación ya que es más fácil verificar que un número es compuesto. Para ello disponemos de los llamados test de pseudoprimalidad.

Definición 3.1.1. *Se llama **test de pseudoprimalidad** a un algoritmo que decide si un número candidato es compuesto, basándose en el cumplimiento de ciertas propiedades por parte del número candidato.*

Esto quiere decir que si el algoritmo decide que el candidato n es compuesto, entonces n será compuesto con toda seguridad, sin embargo, los test de pseudoprimalidad no aseguran la primalidad del candidato.

En [DHM05] vamos a encontrar un estudio muy práctico sobre los test de primalidad y pseudoprimalidad aplicados en esencia al sistema criptográfico RSA y de donde se ha obtenido gran parte de la información de este capítulo.

Uno de los test de pseudoprimalidad más conocidos está basado en el siguiente teorema:

3.1.1. Pequeño teorema de Fermat

Sobre el año 1636 Pierre de Fermat enunciaba en una carta el siguiente célebre resultado:

Teorema 3.1.2 (Pequeño teorema de Fermat)**.** *Sea $p \in \mathbb{N}$ un número primo, entonces se tiene que $\forall a \in \mathbb{Z}_p, a \neq 0$*

$$a^{p-1} \equiv 1 \mod p$$

De forma equivalente podemos escribir $a^p \equiv a \mod p$

Demostración. Existen varias formas de demostrar este teorema, aquí utilizaremos una versión de la demostración que dio Euler basada en el principio de inducción a pesar de no ser la más corta.

Pero antes, necesitaremos demostrar el siguiente lema:

Lema 3.1.3. *Sea $p \in \mathbb{N}$, p es primo si y solo si $\binom{p}{k} \equiv 0 \mod (p)\ \forall k \in Z_p, k \neq 0$*

Demostración. Si acudimos a la definición de un número combinatorio tenemos que $\forall p,k \in \mathbb{N}, p \geq k$ se define

$$\binom{p}{k} = \frac{p!}{(p-k)!\cdot k!} = \frac{p(p-1)(p-2)\cdots 2\cdot 1}{(p-k)(p-k-1)\cdots 2\cdot 1\cdot k(k-1)(k-2)\cdots 2\cdot 1}$$

Supongamos que p es primo. Como los números combinatorios son enteros, es inmediato comprobar que si p es primo entonces se cumple que $\nexists k \in \mathbb{N}, o < k < p$ tal que $k|p$ o bien $(p-k)|p$ y en consecuencia se deduce que

$$p \textit{ divide } \binom{p}{k}$$

o lo que es lo mismo, $\binom{p}{k} \equiv 0 \mod p$.

Supongamos ahora que p es compuesto, entonces $\exists q,i \in \mathbb{N}$ con q primo tal que $q^i|p$ pero $q^{i+1} \nmid p$.

$$\binom{p}{q} = \frac{p!}{(p-q)!q!} = \frac{p\cdots(p-q+1)\cdots 2\cdot 1}{(p-q)(p-q-1)\cdots 2\cdot 1\cdot q(q-1)(q-2)\cdots 2\cdot 1} = \frac{p\cdots(p-q+1)}{q\cdots 2\cdot 1}$$

Sabemos que $q^i|p$ fijémonos que $q \nmid 1$, $q \nmid 2$,..., $q \nmid q-1$ luego el denominador es divisible por q mientras que como $q \nmid p-1$, $q \nmid p-2$,..., $q \nmid p-q+1$ el numerador es divisible por q^i pero no por q^{i+1} en consecuencia

$$q^i \nmid \binom{p}{q}$$

como $q^i|p$ tenemos que $p \nmid \binom{p}{q}$ y por tanto

$$\exists q \in \mathbb{Z}_p, q \neq 0 \textit{ tal que } \binom{p}{q} \neq 0$$

□

Con esta pequeña herramienta estamos listos para demostrar el teorema de Fermat por inducción sobre la base a.

Suponemos que p es un número primo.

Si $a = 1$, tenemos que $1^p \equiv 1 \mod (p)$.

Si $a = 2$, por la fórmula del binomio, $2^p \equiv (1+1)^p \equiv \sum_{k=0}^{p} \binom{p}{k} 1^k 1^{p-k} \equiv \sum_{k=0}^{p} \binom{p}{k}$ y por el lema que acabamos de demostrar, tenemos que $\sum_{k=0}^{p} \binom{p}{k} \equiv \binom{p}{0} + \binom{p}{p} \equiv 1+1 \equiv 2 \mod (p)$

Supongamos ahora que se cumple $a^p \equiv a \mod p$ para un cierto a. Veamos que ocurre con el siguiente término $(a+1)^p$.

$$(a+1)^p \equiv \sum_{k=0}^{p} \binom{p}{k} a^k 1^{p-k}$$

de nuevo por el lema anterior, todos los coeficientes serán 0 menos los correspondientes a $k = 0$ y $k = p$ así pues

$$(a+1)^p \equiv \sum_{k=0}^{p} \binom{p}{k} a^k 1^{p-k} \equiv a^p 1^0 + a^0 1^p \equiv a+1 \mod (p)$$

quedando así completada al demostración por el método de inducción. □

El trabajo de Euler en este problema no se detuvo ahí, Euler logro demostrar una versión más general de este resultado:

Teorema 3.1.4 (Euler)**.** *Sea $p \in \mathbb{N}$, entonces se tiene que $\forall a \in \mathbb{Z}_p$ coprimo con p*

$$a^{\varphi(p)} \equiv 1 \mod p$$

Donde $\varphi(p)$ es la ***función de Euler****, que toma como valor el número de elementos más pequeños y coprimos con p o lo que es lo mismo,*

$$\varphi(n) = |\{a < n | mcd(a,n) = 1\}|$$

En particular, si p es primo, todos los $a \in \mathbb{Z}_p$ son coprimos con p y en consecuencia $\varphi(n) = p-1$ obteniendo así el pequeño teorema de Fermat.

Si echamos la vista atrás ya habíamos hablado de la función de Euler $\varphi(n)$ en el primer capítulo ya que se utiliza en el método criptográfico RSA como parte de la clave privada.

Nos centraremos ahora en el caso particular del teorema de Fermat en el que p es primo y entonces $\varphi(p) = p-1$. Hay que tener cuidado pues en general no se cumple el recíproco de ninguno de los dos teoremas tal y como podemos ver en el siguiente

Ejemplo 3.1.5. *Tomando $a = 2$ y $p = 341$ se tiene que*

$$2^{340} \equiv 1 \mod 341$$

pero sin embargo $p = 341 = 11 \cdot 31$ luego p no es primo

La situación anterior se podría intentar solucionar cambiando el valor de a para encontrar una congruencia distinta de 1, sin embargo en general esto no va a ser posible.

Existe un conjunto de números llamados *números de Carmichael* que cumplen la congruencia del teorema de Fermat pero **no** son primos. Concretamente todos los números de Carmichael son producto de tres o más primos distintos todos ellos dos a dos.

Hay infinitos números de Carmichael aunque aparecen de forma muy esporádica. Solo hay 7 números de este tipo menores de 10000, menos de 600000 que estén por debajo de 10^{17} esto hace que la probabilidad de que eligiendo un número n al azar por debajo de 10^{17} este sea de Carmichael es aproximadamente 10^{-11} mientras que la probabilidad de encontrar un primo se acerca a $\frac{1}{\ln(10^{17})} = \frac{1}{17\ln(10)} \simeq \frac{1}{39}$ que es notoriamente mayor.

Si n es un número impar compuesto y **no** es un número de Carmichael hay como mucho $\frac{\varphi(n)}{2}$ valores de a que verifican $a^{n-1} \equiv 1 \mod n$

En consecuencia, a pesar de que en la mayoría de los casos obtendremos el buen resultado, no podemos utilizar el algoritmo como un test de primalidad pero del contra recíproco del pequeño teorema de Fermat podemos sacar un interesante test de pseudoprimalidad.

Corolario 3.1.6. *Si existe $a \neq 0 \in \mathbb{Z}_p$ tal que*

$$a^{p-1} \not\equiv 1 \mod p$$

entonces p es un número compuesto.

Escogiendo varios valores de a al azar y calculando a^{p-1} aumentan las probabilidades de dar con el buen resultado para decidir la primalidad de p siempre y cuando no hayamos topado con un número de Carmichael.

Si se utiliza la *exponenciación modular*, para cada base a el algoritmo tiene un orden de complejidad de

$$O(\log^2(n) \cdot \log(\log(n)) \cdot \log(\log(\log(n))))$$

3.1.2. Miller-Rabin

El test de *Miller-Rabin* es el sistema que más se utiliza en la actualidad dada su rapidez, aunque se sacrifica la certitud de un test de primalidad, con pocas iteraciones que se realicen se alcanza una muy buena precisión. El algoritmo se basa en el siguiente lema:

Lema 3.1.7. *Sea p un número primo, y sea $x \in \mathbb{Z}_p$ tal que*

$$x^2 \equiv 1 \mod p$$

entonces, $x \equiv 1$ o bien $x \equiv p-1$ en módulo p.

Demostración. Como p es un número primo, $\mathbb{Z}_p$ es un dominio de integridad, por tanto tenemos

$$\begin{gathered} x^2 \equiv 1 \mod p \\ x^2 - 1 \equiv 0 \mod p \\ (x+1)(x-1) \equiv 0 \mod p \end{gathered}$$

Como $\mathbb{Z}_p$ es un dominio de integridad, se tiene o bien $x \equiv 1$ o bien $x \equiv -1 \equiv p-1$ □

Teorema 3.1.8 (Miller - Rabin)**.** *Sea p un número primo, escribimos $p-1 = 2^s m$ donde m es un número impar, entonces $\forall a \in \mathbb{Z}_p$ se tiene*

$$a^m \equiv 1$$

o por el contrario, existe al menos un r entre 0 y $s-1$ tal que

$$a^{2^r m} \equiv -1$$

Demostración. Por el pequeño teorema de Fermat tenemos que

$$a^{p-1} \equiv a^{2^s m} \equiv 1$$

luego por el lema anterior tenemos

$$a^{2^{s-1} m} \equiv 1 \text{ o bien } a^{2^{s-1} m} \equiv -1$$

En el caso de que $a^{2^{s-1} m} \equiv -1$ tenemos el resultado. En el caso contrario si $s \neq 1$ podemos volver a calcular la raíz cuadrada y tenemos de nuevo por el lema anterior

$$a^{2^{s-2} m} \equiv 1 \text{ o bien } a^{2^{s-2} m} \equiv -1$$

iteramos sucesivamente hasta que encontramos -1 o bien llegamos a

$$a^{2^0 m} \equiv a^m \equiv 1 \text{ o } a^m \equiv -1$$

En ambos casos, obtenemos el resultado. □

En su versión original, este test basado en el anterior teorema fue propuesto por G. L. Rabin como un test de primalidad determinista que se apoyaba, al igual que el teorema de Mills, en la Hipótesis generalizada de Riemann[1], sin embargo, tras unos pequeños retoques M. O. Rabin lo transformó en un algoritmo de pseudoprimalidad aleatorizado que o bien asegura que un número candidato es compuesto o bien afirma que es primo con una probabilidad determinada.

Se elige k un parámetro que indicará la precisión del test. Sea n un número impar, sea $a_1, ..., a_k \in \mathbb{Z}_n$ elegidos de forma aleatoria e independiente en el intervalo $2, n-2$, escribimos $n-1 = 2^s m$ donde m es un número impar.

[1] En el teorema de Mills se habla sobre la Hipótesis de Riemann, la cual no se debe confundir con la Hipótesis generalizada de Riemann.

```
para j=1 hasta k
   si (a(j)^m != 1 (mod n) y a(j)^m !=n-1 (mod n))
      devolver n compuesto
   en otro caso
      para i=1 hasta s-1
         si a(j)^((2^i)*m) = 1
            devolver n compuesto
         fin si
      fin para
   fin si
fin para
devolver n probable primo
```

La velocidad de este algoritmo es $O(\log^3(n))$

La ventaja de este método respecto a los anteriores radica en que **no existe** ningún número n compuesto que supere el test para todos los posibles valores de a (tal como ocurría con los números de Carmichael en el test de Fermat), más aún, la proporción de valores de a que superan el test respecto a los posibles valores que toma dicho parámetro siendo n compuesto es inferior a $\frac{1}{4}$ con lo cual con k iteraciones superadas hay como mucho una probabilidad de $\left(\frac{1}{4}\right)^k$ de obtener un falso positivo. Conforme aumenta el número de iteraciones la probabilidad tiende a 0 sin embargo, nunca podremos estar totalmente seguros de la primalidad de un número, siempre existirá cierto riesgo de habernos equivocado.

Ejemplo 3.1.9. *Hemos hablado anteriormente de la clave privada guardada en nuestro DNI compuesta por dos números primos de gran tamaño. Cada DNI contiene dos claves distintas, esto quiere decir cuatro primos en total. Estos números primos se buscan mediante varias pasadas del algoritmo Miller - Rabin con lo que existe cierta probabilidad de dar por primos números que no lo son. Si cada ciudadano español posee cuatro números primos y en España hay 47 millones de personas con un DNI a su nombre quiere decir que se han generado 188 millones de números primos, si cada primo ha sido verificado utilizando por ejemplo 6 veces el algoritmo de Miller - Rabin quiere decir que cada uno de esos primos son números primos* ***probables*** *con una probabilidad de, como mucho,* $\left(\frac{1}{4}\right)^6 \simeq 2{,}4414 \cdot 10^{-4}$ *de ser en realidad números compuestos lo que quiere decir que hay unos* 45898 *falsos primos en España que pueden causar problemas a la hora de identificarnos de forma digital.*

3.1.3. Propiedades deseables

En general es deseable que los test de primalidad que se buscan cumplan una serie de propiedades que garantizan el buen comportamiento y la validez del algoritmo frente a cualquier situación.

- Un test de primalidad que ofrece una cierta probabilidad de que un número sea primo se denomina test **aleatorizado**, mientras que uno que afirma con determinación dicha propiedad se conoce como test **determinista**. Preferiremos un test determinista sobre uno aleatorizado, como en este caso es el algoritmo de Miller-Rabin.

- La rapidez del algoritmo es vital, sobre todo nos interesa su comportamiento asintótico. Considerándose rápido un test si su orden de complejidad es polinómico respecto al número de cifras de entrada.

- Los algoritmos **incondicionales** son aquellos que funcionan sin tener que trabajar bajo ciertas suposiciones como en el test de Fermat, donde teníamos que suponer que no habíamos encontrado un número de Carmichael.

- Existen ciertos tests de primalidad que no pueden ser aplicados a cualquier candidato si no que solo funcionan utilizando cierto subconjunto de números. Buscaremos algoritmos **generales** aunque en ciertas ocasiones pueden resultar útiles dichos tests más específicos.

Miller - Rabin cumple tres de las cuatro propiedades. En la práctica, el carácter aleatorizado del test no supone un problema crítico ya que si ejecutamos el test 50 veces la probabilidad de que devuelva probable primo siendo compuesto es inferior a la probabilidad real de que ocurra un fallo de hardware en el ordenador que computa las instrucciones.

3.2. Test de Lucas-Lehmer

El test de Lucas - Lehmer es un algoritmo rápido, incondicional y determinista pero aplicable solo a cierto tipo de números.

3.2.1. Números de Mersenne

Definición 3.2.1. *Se dice que un número M es de* Mersenne *si es una unidad menor que una potencia de 2.*

$$M_n = 2^n - 1$$

El conjunto de los números de Mersenne cumple que si n es compuesto, entonces M_n también lo es. Desafortunadamente no se cumple en general el recíproco, de hecho, tan solo se conocen actualmente 48 números de Mersenne que sean primos. Una de las ventajas que tienen estos números es que disponemos de más herramientas para discernir su primalidad que si se tratase de un número cualquiera. Esto ha hecho que historicamente desde la aparición de las computadoras, el número primo más grande conocido prácticamente siempre ha sido (y es) un número de Mersenne. El número primo que ostenta dicho record en la actualidad es el $2^{57,885,161} - 1$ y se trata de un número de más de **diecisiete millones** de cifras.

3.2.2. Lucas - Lehmer

El test de Lucas - Lehmer es un test de primalidad que decide si un número de Mersenne es primo o no lo es.

Teorema 3.2.2. *Sea p un primo impar M_p el número de Mersenne tal que $M_p = 2^p - 1$. Definimos la sucesión $\{S_i\}$ como*

$$S_i = \begin{cases} 4 & si\ i = 0 \\ S_{i-1}^2 - 2 & En\ otro\ caso \end{cases}$$

Se cumple que M_p es primo si y solo si $S_{p-2} \equiv 0 \mod (p)$

Nota: p es un número primo exponencialmente más pequeño que M_p y podremos asegurar su primalidad fácilmente.

El número de operaciones requeridas para llevar a cabo este algoritmo es del orden de $O(\log(n \cdot \log(n)))$ o equivalentemente $\overline{O}(k^2 \log(k))$ donde k representa la longitud de n.

Capítulo 4

El test de primalidad AKS

4.1. AKS

En 2002 Manindra Agrawal, Neeraj Kayal y Nitin Saxena (AKS), tres científicos computacionales del Instituto Tecnológico Hindú de Kanpur diseñaron el primer test de primalidad que cumple las cuatro propiedades deseables de estos algoritmos: determinismo, de tiempo polinomial (respecto la longitud del número candidato), generalidad e incondicionalidad.

El algoritmo está basado en una generalización del pequeño teorema de Fermat en los polinomios.

Teorema 4.1.1. *Sean $n, a \in \mathbb{N}$ con n y a coprimos, entonces se cumple*

$$(x+a)^n \equiv x^n + a \mod (n)$$

si y solo si n es primo.

Demostración. $\Rightarrow$) Supongamos n primo.

$$(x+a)^n \equiv \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

como n es primo sabemos que $\binom{n}{k} \equiv 0 \mod (n)\ \forall k, 0 < k < n$ y entonces se cumple

$$(x+a)^n \equiv \binom{n}{0} x^0 a^n + \binom{n}{n} x^n a^0 \equiv x^n + a^n \mod (n)$$

por el pequeño teorema de Fermat tenemos que $a^n \equiv a \mod (n)$ por tanto

$$(x+a)^n \equiv x^n + a^n \equiv x^n + a \mod (n)$$

$\Leftarrow$) Supongamos ahora que n es compuesto. Como n es compuesto existen $q, k \in \mathbb{N}$, con p primo tal que $q^k | n$, $q^{k+1} \nmid n$. Entonces tenemos que $q^k \nmid \binom{n}{q}$ y es coprimo con a^{n-q} (por ser a coprimo con n) luego el coeficiente de x^q, $\binom{n}{q} a^{n-q}$ no es dividido por q^k, y por tanto tampoco por n, en consecuencia

$$(x+a)^n - x^n + a \not\equiv 0 \mod (n)$$

□

Este teorema constituye de por sí una caracterización de número primo a partir de la que puede construirse un test de primalidad aunque demasiado lento y no serviría para mejorar nada. Sin embargo, la idea de los tres autores consiste en realizar los cálculos módulo $x^r - 1$ donde r sea lo suficientemente pequeño. Esto provoca una pérdida de generalidad en el teorema anterior pero la solucionan demostrando que existe un r y un número máximo de bases a acotado por una función de

tiempo polinomial de tal forma que si se sigue cumpliendo la igualdad para dichos a, se recupera la caracterización del teorema 4.1.1

Los pasos del algoritmo AKS que hay que implementar en un ordenador para averiguar si n es primo serían, escritos de forma simplificada, los siguientes:

1. Si existe $a \in \mathbb{N}$ tal que $n = a^b$ para algún $b > 1$ devolver COMPUESTO.
2. Encontrar el menor r de forma que $o_r(n) > \log^2(n)$.
3. Si $1 < mcd(a,n) < n$ para algún $a \leq r$ devolver COMPUESTO.
4. Si $n \leq r$ devolver PRIMO.
5. Desde $a = 1$ hasta $\lfloor \sqrt{\varphi(r)}\log(n) \rfloor$ comprobar
   - si $(x+a)^n \not\equiv x^n + a \mod (x^r - 1, n)$ devolver COMPUESTO.
6. Devolver PRIMO.

Donde log es el logaritmo en base 2 y $o_r(n)$ es el orden de n módulo r:

$$o_r(n) = \text{mín}\left\{k \in \mathbb{N} | n^k \equiv 1 \mod r\right\}$$

Notar que, en los pasos 3 y 5, si r no es pequeño, el número de operaciones a realizar corre el riego de dispararse. Tal y como veremos en la demostración, r está acotado por $\log^5(n)$ manteniendo así el orden de complejidad polinomial respecto al número de cifras de n. Por otro lado, en el paso 3 se realizan r cálculos del $mcd(a,n)$, esto implica que, si cuando elegimos r en el paso 2, se cumple que $\sqrt{n} \leq r \leq \log^5(n)$ es más rápido aplicar el test de las divisiones sucesivas que realizar el paso 3. Esto puede ocurrir para todos los $n < 3{,}43 \cdot 10^{15}$ y claramente, si $\sqrt{n} \leq r$, no tiene sentido aplicar los pasos 3 y 4 (ni seguir con la ejecución del AKS) cuando podemos aplicar el algoritmo de las divisiones. En el caso contrario, si $n > 3{,}43 \cdot 10^{15}$ tampoco tiene sentido realizar la comparación entre n y r ya que $n > \sqrt{n} \geq \log^5(n) \geq r$. De este hecho puede estudiarse una mejora del algoritmo suprimiendo el paso 4 para evitar cálculos en caso de que el candidato n sea relativamente pequeño.

No obstante aquí utilizaremos el algoritmo original que cobra sentido para n grande ya que lo importante en el AKS es justamente, tal como veremos, su buen comportamiento cuando n crece. Vamos a demostrar que el algoritmo AKS devuelve PRIMO en el caso de que el número n lo sea y COMPUESTO en el caso contrario mediante una sucesión de pequeños lemas:

Lema 4.1.2. *Si n es primo, el algoritmo AKS devuelve PRIMO*

Demostración. Si n es primo, es inmediato comprobar que los pasos 1 y 3 nunca devolverán COMPUESTO. Como consecuencia del teorema 4.1.1 en el paso 5 tampoco devolverá nunca COMPUESTO. Por tanto, el algoritmo devolverá PRIMO en el paso 4 o en último lugar en el 6. □

Fijémonos que si el algoritmo devolviese PRIMO en el paso 4, entonces n debe ser primo obligatoriamente, de lo contrario, habríamos sido capaces de encontrar un divisor en el paso anterior. En consecuencia, sólo nos queda comprobar que si el algoritmo devuelve PRIMO en el último punto habiendo realizado todos los pasos previos, entonces n es primo; o lo que es equivalente, comprobar que no hay números compuestos que superen los pasos 1, 3 y 5. Para ello tiene especial relevancia el paso 2 en el que hay que elegir de forma adecuada r.

Lema 4.1.3. *Existe* $r \in \mathbb{N}$, $r \leq \text{máx}\left\{3, \lceil \log^5(n) \rceil\right\}$ *tal que* $o_r(n) > \log^2(n)$

Demostración. Si $n = 2$ tenemos que $r = 3$ satisface la propiedad $o_3(2) = 2 > 1 = \log^2(2)$ por tanto supondremos que $n \geq 3$ lo que implica que $\lceil \log^5(n) \rceil > 10$.

Consideremos el producto:

$$n^{\lfloor \log(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2(n) \rfloor} (n^i - 1)$$

Elegimos $r \in \mathbb{N}$ de forma que sea el menor entero positivo que no divide dicho producto, que sabemos que existe.

Fijémonos que el máximo número $k \in \mathbb{N}$ que satisface $m^k \leq B = \lceil \log^5(n) \rceil$ es $k = \lfloor \log(B) \rfloor$. Sea $r_1^{c_1} \cdots r_z^{c_z}$ la descomposición de r en potencias de primos, tenemos que si todos los primos $r_1, ..., r_z$ dividen a n y siendo $c_i = \text{máx}(c_1, ..., c_z)$ se tiene que $r | (r_1 \cdots r_z)^{c_i}$ y por lo anterior, tendríamos que $r | n^{\lfloor \log(B) \rfloor}$. Deducimos de aquí que no todos los primos de r dividen a $mcd(r,n)$ y que $\frac{r}{mcd(r,n)}$ tampoco divide al producto de los $(n^i - 1)$ por la coprimalidad entre n y $n^i - 1$.

Como r es el mínimo número con esas condiciones, forzosamente se cumple que $mcd(r,n) = 1$, y como r no divide a ningún $n^i - 1$ para $1 \leq i \leq \lfloor \log^2(n) \rfloor$ tenemos $o_r(n) > \log^2(n)$.

Finalmente, basta comprobar que $r \leq \lceil \log^5(n) \rceil$, teniendo en cuenta que hemos supuesto $n > 2$

$$n^{\lfloor \log(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2(n) \rfloor} (n^i - 1) < [n^{\lfloor \log(B) \rfloor} \cdot \prod_{i=1}^{\lfloor \log^2(n) \rfloor} (n^i) < n^{\lfloor \log(B) \rfloor} \cdot n^{\sum_{i=1}^{\lfloor \log^2(n) \rfloor} i} \leq$$

$$\leq n^{\lfloor \log(B) \rfloor} \cdot n^{1+2+3+...+(log^2(n)-1)} = n^{\lfloor \log(B) \rfloor + \frac{1}{2}(\log^2(n)-1)\log^2(n)} \leq n^{\log^4(n)} \leq 2^{\log^5(n)} \leq 2^B$$

Como $\lceil \log^5(n) \rceil > 10$ podemos aplicar el siguiente lema cuya demostración no realizaremos por apartarse demasiado de nuestros objetivos pero que se puede encontrar en [Nai82]:

Lema 4.1.4. *Llamamos $MCM(m)$ al $mcm(1,2,3,4,...,m)$ donde mcm es el mínimo común múltiplo. Si $m \geq 7$ entonces:*

$$MCM(m) \geq 2^m$$

Así pues, dado que r no divide al producto y el producto es menor que $MCM(B)$, el lema 4.1.4 implica que $r \leq B$ ya que de lo contrario por la minimalidad de r, todos los números $i \leq B$ dividirían al producto que está acotado llegando así a una contradicción.

□

Hemos conseguido demostrar la existencia y acotar superiormente r por $\lceil \log^5(n) \rceil$. Además dado que $n \neq 1$ se cumple $o_r(n) > 1$ y entonces debe existir un factor primo p de n de forma que $o_r(p) > 1$ y además $p > r$ o de lo contrario habríamos obtenido alguna caracterización de n en los pasos 3 y 4. Como $mcd(n,r) = 1 = mcd(p,r)$, $p, n \in \mathbb{Z}_r^*$, llamaremos $l = \lfloor \sqrt{\varphi(r)} \log(n) \rfloor$ al número de ecuaciones que se verifican en el paso 5. Como hemos supuesto que el algoritmo no ha devuelto COMPUESTO en el paso 5 quiere decir que se han verificado cada una de las ecuaciones y tenemos:

$$(x+a)^n \equiv x^n + a \quad \text{mód } (x^r - 1, n)$$

$\forall a = 0, ..., l$ esto implica:

$$(x+a)^n \equiv x^n + a \quad \text{mód } (x^r - 1, p) \tag{4.1}$$

$\forall a = 0, ..., l$ y por el teorema 4.1.1

$$(x+a)^p \equiv x^p + a \quad \text{mód } (x^r - 1, p) \tag{4.2}$$

al juntar las ecuaciones 4.1 y 4.2 deducimos

$$(x+a)^{\frac{n}{p}} \equiv x^{\frac{n}{p}} + a \quad \text{mód } (x^r - 1, p)$$

$\forall a = 0, ..., l$

Vemos que el comportamiento de $\frac{n}{p}$ y n en módulo $(x^r - 1, p)$ es el mismo, vamos a definir esta propiedad:

Definición 4.1.5. *Para un polinomio $f(x)$ y $m \in \mathbb{N}$ decimos que m es* introspectivo *para $f(x)$ si*

$$[f(x)]^m \equiv f(x^m) \mod (x^r - 1, p)$$

Por tanto $\frac{n}{p}$ y n son introspectivos para $f(x) = x + a$

Lema 4.1.6. *Los números introspectivos son cerrados respecto a la multiplicación.*

Demostración. Sean m y m' dos números introspectivos para $f(x)$, tenemos que comprobar que $m \cdot m'$ también es introspectivo para $f(x)$.

Dado que m es introspectivo, se cumple

$$[f(x)]^{m \cdot m'} \equiv [f(x^m)]^{m'} \mod (x^r - 1, p)$$

igualmente, ya que m' es introspectivo, si sustituimos x por x^m tenemos

$$\begin{aligned} [f(x^m)]^{m'} &\equiv f(x^{m \cdot m'}) \mod (x^{m \cdot r} - 1, p) \\ &\equiv f(x^{m \cdot m'}) \mod (x^r - 1, p) \end{aligned}$$

ya que $x^r - 1$ divide a $x^{m \cdot r} - 1$

Uniendo ambas ecuaciones concluimos

$$[f(x)]^{m \cdot m'} \equiv f(x^{m \cdot m'}) \mod (x^r - 1, p)$$

□

Además ocurre algo similar con los polinomios:

Lema 4.1.7. *Si m es introspectivo para $g(x)$ y $f(x)$ también lo es para $g(x) \cdot f(x)$.*

Demostración. Sea $g(x)$, $f(x)$ dos polinomios tal que m es introspectivo para ellos.

$$\begin{aligned} [f(x) \cdot g(x)]^m &\equiv [f(x)]^m \cdot [g(x)]^m \\ &\equiv f(x^m) \cdot g(x^m) \mod (x^r - 1, p) \end{aligned}$$

□

Los dos lemas anteriores justifican que cada número en el conjunto $I = \left\{ (\frac{n}{p})^i \cdot p^j | i, j \geq 0 \right\}$ es introspectivo para todos los polinomios del conjunto $P = \left\{ \prod_{a=0}^{l} (x+a)^{e_a} | e_a \geq 0 \right\}$. Definiremos ahora dos grupos basados en estos resultados que serán cruciales para el desarrollo de la demostración.

El primero es el conjunto de todos los residuos del conjunto I módulo r, lo denominaremos G. Este conjunto es un subgrupo de $\mathbb{Z}_r^*$ puesto que como hemos visto, $mcd(n,r) = 1 = mcd(p,r)$ cuyo cardinal denotamos $|G| = t$. Tanto n como p (módulo r) son generadores de G y en consecuencia como $o_r(n) > \log^2(n)$, se verifica $t > \log^2(n)$.

Para definir el segundo grupo utilizaremos polinomios ciclotómicos sobre cuerpos finitos. Denotaremos F_p al cuerpo finito de p elementos con p primo. Utilizaremos el hecho de que si $h(x)$ es un polinomio irreducible de grado d sobre F_p entonces $F_p[x]/(h(x))$ es un cuerpo finito de orden p^d.

Sea $Q_r(x)$ el r-ésimo polinomio ciclotómico sobre F_p, $Q_r(x)$ divide a $x^r - 1$ y se descompone como factores irreducibles de orden $o_r(p)$ (podemos encontrar todas estas propiedades más desarrolladas en [LN86]). Sea $h(x)$ uno de dichos factores irreducibles, como $o_r(p) > 1$ el grado de $h(x)$ también es mayor que 1. El segundo grupo es el conjunto de todos los residuos de los polinomios de P módulo $h(x)$ y p al que llamaremos $\wp$ y que está generado por los elementos x, $x+1$,...,$x+l$ del cuerpo $F = F_p[x]/(h(x))$, además, $\wp$ es un subgrupo multiplicativo de F.

Lema 4.1.8 (Hendrik Lenstra Jr.)**.**

$$|\wp| \geq \binom{t+l}{t-1}$$

Demostración. En primer lugar vamos a comprobar que para dos polinomios distintos cualesquiera de grado menor que t pertenecientes a P se obtienen dos elementos distintos de $\wp$. Sean $f(x)$ y $g(x)$ dos polinomios de P. Supongamos $f(x) = g(x)$ en F, para cada $m \in I$ se verifica $[f(x)]^m = [g(x)]^m$. Como m es introspectivo para f y g y además $h(x)$ divide a $x^r - 1$ se tiene

$$f(x^m) = g(x^m)$$

Luego es obvio que para cada $m \in G$, x^m es una raíz del polinomio $Q(y) = f(y) - g(y)$ y como G es un subgrupo de $\mathbb{Z}_r^*$, cada uno de los x^m es una raiz primitiva r-ésima de la unidad. Por tanto habrá $|G| = t$ raíces distintas de $Q(y)$ en F. Sin embargo, el grado de $Q(y)$ es estrictamente menor que t ya que f y g pertenecen a $\wp$. Por tanto llegamos a contradicción.

Fijémonos que $i \neq j$ en F_p para todo $1 \leq i, j \leq l$ puesto que $l = \lfloor \sqrt{\varphi(r)} \log(n) \rfloor < \sqrt{r} \log n < r < p$ y por tanto los elementos $x, x+1,...,x+l$ son todos distintos en F.

Como el grado de $h(x)$ es mayor que 1, $x+a$ no es idénticamente nulo para ningún $a = 0, ..., l$. En consecuencia existen al menos $l+1$ polinomios distintos de grado 1 en $\wp$ y por lo tanto existen como mínimo $\binom{t+l}{t-1}$ polinomios distintos de grado menor o igual que t en $\wp$.

□

En el caso de que n no sea una potencia de p, es posible acotar superiormente el cardinal de $\wp$.

Lema 4.1.9. *Si n no es una potencia de p entonces* $|\wp| \leq n^{\sqrt{t}}$.

Demostración. Consideramos el siguiente subconjunto de I

$$\widehat{I} = \left\{ (\frac{n}{p})^i \cdot p^j | 0 \leq i, j \leq \lfloor \sqrt{t} \rfloor \right\}$$

Si n no es una potencia de p entonces existen $(\lfloor \sqrt{t} \rfloor + 1)^2 > t$ elementos distintos en $\widehat{I}$. Como $|G| = t$ deben existir al menos dos elementos de $\widehat{I}$ que sean iguales módulo r, llamémosles m_1 y m_2 con $m_1 > m_2$. En este caso tenemos que

$$x^{m_1} \equiv x^{m_2} \mod x^r - 1$$

Para cada $f(x) \in P$ se cumple que

$$\begin{aligned} [f(x)]^{m_1} &\equiv f(x^{m_1}) \mod (x^r - 1, p) \\ &\equiv f(x^{m_2}) \mod (x^r - 1, p) \\ &\equiv [f(x)]^{m_2} \end{aligned}$$

Esta ecuación se cumple por tanto en F, lo que implica $f(x) \in \wp$ es una raíz del polinomio $Q'(y) = y^{m_1} - y^{m_2}$ en F. Dado que esto se cumple para todos los polinomios de $\wp$ tenemos que el polinomio $Q'(y)$ tiene al menos $|\wp|$ raíces distintas en F. Y se tiene

$$|\wp| \leq grado(Q'(y)) = m1 \leq (\frac{n}{p} \cdot p)^{\lfloor \sqrt{t} \rfloor} \leq n^{\lfloor \sqrt{t} \rfloor}$$

Obteniéndose la cota del lema. □

Con todos estos lemas a nuestra disposición estamos listos para terminar la demostración:

Teorema 4.1.10. *Si el algoritmo devuelve PRIMO, entonces n es primo.*

Demostración. Tal como hemos visto al principio de la demostración, si el algoritmo devuelve primo en los primeros pasos es inmediato comprobar que efectivamente n lo es. Supongamos que el algoritmo ha devuelto PRIMO en el último paso.

El lema 4.1.8 implica que siendo $t = |G|$ y $l = \lfloor \sqrt{\varphi(r)} \log(n) \rfloor$ se cumple

$$\begin{aligned}
|\wp| &\geq \binom{t+l}{t-1} \\
&\geq \binom{l+1+\lfloor \sqrt{t}\log(n) \rfloor}{\lfloor \sqrt{t})\log(n) \rfloor} (\text{ Puesto que } t > \sqrt{t}\log(n)) \\
&\geq \binom{2\lfloor \sqrt{t}\log(n) \rfloor + 1}{\lfloor \sqrt{t}\log(n) \rfloor} (\text{ Por ser } l = \lfloor \sqrt{\varphi(r)}\log(n) \rfloor \geq \lfloor \sqrt{t}\log(n) \rfloor) \\
&> 2^{\lfloor \sqrt{t}\log(n) \rfloor + 1} (\text{ Ya que } \lfloor \sqrt{t}\log(n) \rfloor > \lfloor \log^2(n) \rfloor \geq 1) \\
&\geq n^{\sqrt{t}}
\end{aligned}$$

Por el lema 4.1.9 tenemos que si n no es una potencia de p, entonces $n^{\sqrt{t}} \geq |\wp| > n^{\sqrt{t}}$, lo que es absurdo, así que necesariamente n es una potencia de p y por tanto, siendo $n = p^k$, si $k > 1$ entonces el algoritmo habría devuelto COMPUESTO en el primer paso. Finalmente $k = 1$ y $n = p$ primo. □

Para estimar el número de operaciones necesarias para llevar a cabo el test de primalidad vamos a estudiar cada paso por separado:

1. En el primer paso se puede factorizar el polinomio $x^b - n$ para $b \leq \log(n+1)$. Si obtenemos un factor de grado uno se tiene que $x^b - n = (x-a)g(x) \Longrightarrow a^b = n$ y el algoritmo devuelve COMPUESTO. Este proceso se puede realizar en $O(\log^2(n))$ para cada uno de los $\log(n+1)$ distintos b posibles por lo que concluimos que este paso tiene un orden de $O(\log^3(n))$.

2. En el segundo paso calculamos el orden de n módulo r para cada r hasta comprobar si el orden es superior a $\log^2(n)$. Por el lema 4.1.3 sabemos que $r \leq \log^5(n)$ luego realizaremos este paso en $O(\log^7(n))$ operaciones.

3. En este paso calcularemos r veces el *mcd*. Calcular el *mcd* tiene un orden de complejidad de $O(\log(n))$ luego el tercer paso tiene un coste de $O(\log^6(n))$.

4. Sólo realizamos una comparación: $O(1)$.

5. En este paso debemos realizar $\lfloor \sqrt{\varphi(r)}\log(n) \rfloor$ comprobaciones de nulidad. Para cada uno de ellos hay que realizar $O(\log(n))$ multiplicaciones de polinomios de grado r con coeficientes no mayores que $O(\log(n))$. El orden de complejidad de este paso es por tanto $O(r\sqrt{\varphi(r)}\log^3(n)) = O(\log^{\frac{21}{2}}(n))$

El tiempo que requiere el cuarto paso es mayor (asintóticamente hablando) que el de todos los demás pasos. Es del orden de $O(\log^{\frac{21}{2}}(n))$, lo que marca el coste computacional del algoritmo. Ya se han encontrado formas de mejorar el tiempo de ejecución principalmente en la forma de acotar r llegando a reducir el orden a

$$O(\log^6(n)) \simeq \overline{O}(k^6)$$

En el propio artículo en el que se publicó el método AKS [AKS](pág. 7) los autores proponían una forma de reducir el tiempo a $O(\log^3(n))$ si se demostraba como cierta la llamada conjetura de Agrawal.

Conjetura 4.1.11 (Agrawal)**.** *Sean $r,n \in \mathbb{N}$ coprimos, entonces si*

$$(x-1)^n \equiv x^n - 1 \mod (n, x^r - 1)$$

se cumple que n es primo o bien $n^2 \equiv 1 \mod (r)$

Desafortunadamente, se sospecha que dicha conjetura no se cumple en general. Se está intentando demostrar alguna versión de la conjetura con hipótesis menos restrictivas o comprobar que se cumple para números mayores de cierta cota.

4.2. Un pequeño ejemplo

Vamos a aplicar el algoritmo AKS para averiguar si el número $n = 31$ es primo o no:

Primer paso

$n = a^b \iff a = \sqrt[b]{n}$ Basta comprobar las raíces k-ésimas de n desde $k = 2$ hasta $k \leq \log_2(n) \simeq 4{,}95$: $\sqrt{31} \simeq 5{,}56$; $\sqrt[3]{31} \simeq 3{,}14$; $\sqrt[4]{31} \simeq 2{,}35$

Por tanto concluimos que 31 no es una potencia de $a \in \mathbb{N}$ con $a < n$.

Segundo paso

$\log_2^2(31) \simeq 24{,}54$ luego buscamos un r que verifique que el orden de n módulo r sea 25 o mayor. No tiene sentido buscar $r < 25$ puesto que el orden del grupo no puede ser menor que el orden de sus elementos.

$o_r(31) = \text{mín}\left\{k \in \mathbb{N} | 31^k \equiv 1 \mod r\right\}$

$$\begin{aligned} o_{25}(31) &= 5 < 25 \\ o_{26}(31) &= 4 < 25 \\ o_{27}(31) &= 9 < 25 \\ o_{28}(31) &= 6 < 25 \\ o_{29}(31) &= 30 \geq 25 \end{aligned}$$

Por tanto $r = 29$.

Tercer paso

Hay que calcular el $mcd(a,n)$ para cada $a \leq r$.

$$\begin{aligned} mcd(2,31) &= 1 \\ mcd(3,31) &= 1 \\ mcd(4,31) &= 1 \\ &\ldots \\ mcd(29,31) &= 1 \end{aligned}$$

Superado el tercer paso pasamos al siguiente.

Cuarto paso

Como $29 < 31$ no nos pronunciamos sobre la primalidad de 31 y pasamos al siguiente paso.

Quinto paso

Hay que calcular en primer lugar $\varphi(29) = 28$

$$l = \lfloor \sqrt{\varphi(29)} \log_2(31) \rfloor = \lfloor 26{,}21514... \rfloor = 26$$

Calculamos para $a = 1$ hasta 26:

$$(x+a)^{31} \quad \text{mód } (x^{29}-1, 31)$$

y verificamos que en los 26 casos se verifica la igualdad $(x+a)^{31} \equiv x^{31}+a \quad \text{mód } (x^{29}-1, 31)$

Sexto paso

Como hemos superado todos los pasos anteriores podemos concluir que 31 es un número primo.

4.3. Importancia del resultado

Está claro que no tiene sentido utilizar el algoritmo AKS para verificar la primalidad de números de pocas cifras: hasta que n no sea un número de unas 50 cifras sigue siendo más fácil realizar $\sqrt{n}$ operaciones que $\log^{10,5}(n)$. Tal y como podemos comprobar en el ejemplo anterior, en el tercer paso se verifican 29 máximos comunes divisores cuando utilizando el método de las divisiones sucesivas solo haría falta comprobar $\lfloor \sqrt{31} \rfloor = 5$ divisiones.

En el trabajo de final de carrera [BorTFC] encontramos un interesante estudio y aplicación de este algoritmo y su complejidad computacional así como de sus futuras posibles implicaciones.

Hasta el descubrimiento de este resultado sólo se habían logrado implementar algoritmos que cumpliesen tres de las cuatro propiedades deseables para los test de primalidad. El orden de complejidad del método AKS es cuadrático respecto al algoritmo de Miller - Rbin, además las constantes multiplicativas que acompañan al orden del método AKS son muy grandes, por lo tanto el tiempo es proporcionalmente mucho mayor aunque no se refleje en los órdenes de complejidad lo que ha hecho que se siga utilizando en la práctica el test aleatorizado. Tal vez si se demostrase la conjetura de Agrawal el AKS podría empezar a competir por ser utilizado como test de primalidad eficaz.

Para decidir si un número es primo de forma rotunda se dispone también de distintos test de primalidad basados en curvas elípticas que son de tiempo exponencial pero muy eficientes incluso para números grandes, ya que las constantes multiplicativas que acompañan los tiempos de ejecución son pequeñas al igual que los exponentes. Ésto no quiere decir que a partir de un cierto n_0 el algoritmo AKS no sea más eficaz que los utilizados actualmente (que según la teoría, lo es), pero como dicho n_0 tiene un número de cifras extremadamente elevado y no se utilizan en la práctica números tan grandes, realizan menos operaciones los test basados en curvas elípticas. Por tanto no hay lugar a implementar de forma eficaz el algoritmo AKS. No se espera que el algoritmo AKS suponga una revolución en la práctica a pesar de haberlo supuesto en la teoría al tratarse del primer test que demuestra que la primalidad es un problema resoluble en tiempo polinomial lo que ha impulsado enormemente la investigación en esta rama de las matemáticas y la computación.

El AKS no supone un problema de seguridad en las claves de encriptación ya que estas están basadas en la dificultad de factorizar un número en sus factores primos, no en determinar la primalidad de dicho número. En realidad el AKS permitiría encontrar primos con más dígitos que no pertenezcan a ninguna familia conocida con mayor facilidad, lo cual dificultaría la búsqueda por fuerza bruta o por los métodos más comunes de descifrado de claves.

Bibliografía

[Nai82] M. Nair, *On Chebyshev-type inequalities for primes.*, Amer. Math. Monthly, 89:126 - 129, 1982.

[LN86] R. Lidl y H. Niederreiter, *Introduction to finite fields and their applications.*, Cambridge University Press, 1986.

[Gauss] Miguel Ángel Morales Medina, *El teorema de Mills: mucho ruido y pocas nueces*, http://gaussianos.com/el-teorema-de-mills-mucho-ruido-y-pocas-nueces/, 5 octubre 2015.

[BorTFC] C. E. Borges Hernández, *Test de primalidad AKS*, Universidad de Cantabria, 2005.

[AKS] M. Agrawal N. Kayal y N. Saxena, *PIMES is in P*, Department of Computer Science and Engineering Indian Institute of Technology Kanpur, 2002.

[DHM05] R. Duran L. Hernández y J. Muñoz, *El criptosistema RSA*, Instituto de física aplicada C.S.I.C. Madrid, 2005.

[GV98] R. Guerequeta y A. Vallecillo, *Técnicas de Diseño de Algoritmos*, Escuela Técnica Superior de Ingeniería Informática de la universidad de Málaga, 1998.

[DMmr] David Marker, *The Rabin-Miller Primality Test*, http://homepages.math.uic.edu/ marker/math435/rm.pdf, Dep. of Mathematics, Statistics, and Comp. Science, University of Illinois. Edición septiembre 2015.

[PDNI] Cuerpo Nacional de Policia, *DNI electrónico*, http://www.dnie.es/PortalDNIe/