Criptografía

2^o ciclo de Ingeniería Informática — Optativa

Créditos: 6 = 3T + 3P

Bibliografía: Comentaremos algunos de los textos que nos parecen más interesantes para el desarrollo de la asignatura.

La historia de la criptología está contada de manera muy amena y con el suficiente rigor en varios libros. En mi opinión el libro de Singh *Los Códigos Secr3tos* ([Sin00]) es el más recomendable para un primer contacto, posteriormente se puede leer el clásico de Kahn *The codebreakers* ([Kah96]) que tiene un carácter más enciclopédico.

Los siguientes textos son asequibles y tratan amplias partes del programa de forma rigurosa y clara:

[FdlGH$^+$00] Fúster, de la Guía, Hernández, Montoya y Muñoz, J. *Técnicas criptográficas de protección de datos. 2^a Ed.* .

[MT97] Munuera, C. and Tena, J. *Codificación de la Información.*

[PS97] Pastor, J. and Sarasa, M.A *Criptografía digital. Fundamentos y aplicaciones.*

[Sta99] Stallings, W. *Cryptography and network security. Principles and practice.*

Además de los anteriores, para la asignatura similar en la Licenciatura de Matemáticas, son también accesibles (y recomendables):

[Chi95] Childs, L. *A concrete introduction to higher algebra.*

[Gib93] Giblin, P. *Primes and programming.*

[Kob87] Koblitz, N. *A Course in Number Theory and Cryptograhy.*

[LT90] López, F. and Tena, J. *Introducción a la teoría de los números primos* .

En algunas partes del curso se han utilizado algunos libros de propósito más amplio y de mayor profundidad. La mayoría de ellos superan el nivel recomendable para los alumnos de ambas asignaturas, aunque en algunos aspectos puntuales pudieran ser útiles a los alumnos de matemáticas. Citaremos entre ellos de manera especial:

[Coh93] Cohen, H. *A course in computational algebraic number theory.*

[CP01] Crandall, R. and Pomerance, C. *Prime numbers. A computational perspective.*

[vzGG99] von zur Gathen, J. and Gerhard, J. *Modern computer algebra* .

[Kob98] Koblitz, N. *Algebraic aspects of criptology.*

[Yan00] Yan, S.Y. *Number Theory for Computing.*

Además de los textos mencionados una importante fuente de información son algunas páginas web en las que se puede encontrar información actual y rigurosa sobre criptografía y las técnicas de teoría de números útiles. Entre ellas citaremos:

http://www.iacr.org Página de la International Association for Cryptologic Research.

http://www.criptored.upm.es Página de la red temática iberoamericana de criptografía y seguridad de la información.

http://www.utm.edu/research/primes/ Una de las mejores páginas sobre números primos, mantenida por Chris Caldwell.

http://www.kriptopolis.com Contiene interesante información sobre criptología.

http://www.iec.csic.es/criptonomicon/ Dedicado a los problemas de la seguridad en Internet.

http://www.alpertron.com.ar/tnumeros.htm Página personal de Dario Alpern, contiene algunos algoritmos interesantes, entre otros la factorizacion con curvas elipticas.

Índice Temático:

1.. Introducción

La criptología en la historia. Sistemas criptográficos. Sistemas de clave privada y de clave pública. Criptoanálisis. Seguridad de un sistema criptográfico.

Históricamente las técnicas criptograficas aparecen ligadas, sobre todo, a la necesidad de mantener secretas ciertas informaciones entre gobiernos y/o militares.

Su importancia en la historia se refleja en numerosos episodios, muchos de los cuales están suficientemente documentados. A partir del renacimiento es frecuente encontrar numerosos nombres de matemáticos ilustres ligados a momentos históricos concretos en los que su contribución fué esencial. De todos ellos, el destacado papel del equipo de Bertley liderado por Alan Turing en el desciframiento del código de las máquinas de cifrado Enigma, usadas por las fuerzas del eje en la segunda guerra mundial, es posiblemente el más conocido, seguramente porque la proximidad histórica le presta todavía una emotividad de la que carecen otros acontecimientos más remotos.

A partir de la segunda guerra mundial y, sobre todo, con la llegada de los medios de comunicación electrónicos el objetivo primario de la criptología se extiende a una necesidad universal. Particulares, organizaciones comerciales, políticas, ... precisan garantizar que sus comunicaciones son seguras; sobre todo porque los medios informáticos y electrónicos ya no garantizan la (relativa) garantía de confidencialidad de un mensajero personal o incluso de un sobre cerrado. En este proceso, a partir de los años 70, la propia criptología comienza a definir con claridad algunos de sus objetivos, extiende y sistematiza sus métodos y, como consecuencia, comienza a estructurarse como una disciplina científica.

Cuando hablamos de seguridad de una transmisión entre dos personas (tradicionalmente denotadas por A (Alicia) y B (Benito)), lo primero es preguntarse ante qué o ante quién. Surge entonces de manera natural la idea de una tercera persona, C (el escucha, criptoanalista o a veces simplemente el enemigo), que desea interferir en el proceso de comunicación entre A y B, ya sea para conocer el contenido de los mensajes, para suplantar la personalidad de alguno de los interlocutores o con cualquier otro fín "malévolo" para los intereses de A y B.

Podemos pensar que el objetivo último de la *criptografía* es proporcionar métodos que conviertan un canal de comunicación entre A y B que es principio inseguro (es decir, susceptible de ser interferido por terceros) en un sistema en el que A y B puedan confiar. En el lado opuesto el papel del *criptoanálisis* es el de buscar los métodos que permitan romper la seguridad del sistema implantado por el criptógrafo. El término *criptología* se reserva para la ciencia que incluye ambas facetas.

Sistemas criptográficos. Un sistema criptográfico (o criptosistema) está formado por los siguientes ingredientes:

1. Un conjunto $\mathcal{M}$, llamado el conjunto de mensajes originales o mensajes en claro. Normalmente $\mathcal{M} = \mathcal{A}^*$, es decir, los mensajes $M \in \mathcal{M}$ son sucesiones finitas de símbolos de un cierto alfabeto $\mathcal{A}$.
2. Un conjunto finito $\mathcal{K}$, llamado el conjunto de claves (o llaves).
3. Un conjunto $\mathcal{C}$: el conjunto de mensajes cifrados. Habitualmente se suele tomar también $\mathcal{C} = \mathcal{A}^*$.
4. Una aplicación de cifrado (o encriptación) $c : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$.

5. Una aplicación de descifrado $d : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$.
6. Un conjunto de nodos o usuarios N junto con un subconjunto $L \subset N \times N$ (líneas de transmisión). Los elementos de L están formados pués por parejas $(e, r) \in N \times N$, el nodo e se llama el emisor y el nodo r el receptor.
7. Para cada línea $\ell = (e, r) \in L$ una clave k_ℓ de cifrado y una clave k'_ℓ de descifrado tales que $d(c(M, k_\ell), k'_\ell) = M$.

Por lo tanto, si el nodo e desea enviar un mensaje M al nodo r con $\ell = (e, r) \in L$ procede primero al cifrado del mismo: $c(M, k_\ell) \in \mathcal{C}$. El nodo r, una vez que ha recibido el mensaje cifrado recupera el mensaje en claro mediante $d(c(M, k_\ell), k'_\ell) = M$.

Textos y números: Clásicamente los mensajes se suponen escritos en un lenguaje y por lo tanto mediante signos de un alfabeto (p.ej. el alfabeto latino). Sin embargo para su tratamiento es conveniente escribirlos numéricamente. Para ello, si $N \geq \#\mathcal{A}$ basta definir una inyección de $\mathcal{A}$ en $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$. También se utiliza a veces una inyección de $\mathcal{A}$ en un cuerpo finito con q elementos, $\mathbb{F}_q$.

Algunas formas concretas de realizar esta operación son, en la práctica, más adecuadas para su implementación en un ordenador. En los ejemplos y programas utilizaremos el siguiente método:

A) Ascii: Identificamos de los caracteres con su *código ascii* (que es la codificación numérica usada por los ordenadores). Normalmente se usa una codificación de 8 bits, lo que permite codificar 256 símbolos bien mediante la identificación de los mismos con $\mathbb{Z}/256\mathbb{Z}$ bien con $\mathbb{F}_2^8$ (secuencias de 8 bits) bien con el cuerpo finito $\mathbb{F}_{256}$. En nuestro caso usamos con mayor frecuencia $n : \mathcal{A} \to \mathbb{Z}/256\mathbb{Z}$.

B) Bloques: Los bloques formados por un número k de letras, $\alpha = a_1, \ldots, a_k$, se codifican numéricamente mediante

$$n(\alpha) = n(a_1) + n(a_2)256 + \cdots + n(a_k)256^{k-1} ,$$

siendo $n(a_i) \in \mathbb{Z}/256\mathbb{Z}$ el codigo ascii de a_i. Obsérvese que $n(\alpha) < 256^k$, por lo tanto podemos supone que $n\alpha \in \mathbb{Z}/N\mathbb{Z}$ para cualquier $N \geq 256^k$.

Clave privada y clave pública. Se pueden distinguir dos grandes clases de criptosistemas: los de clave pública y los de clave privada. Los sistemas de clave privada incluyen todos los sistemas criptográficos clásicos, los de clave pública sólo aparecen con el desarrollo y la implantación a gran escala de las telecomunicaciones y la informática.

En los sistemas de *clave privada* los usuarios del sistema (normalmente pocos e idealmente dos) comparten y guardan en secreto una pareja de claves k, k' de cifrado y descifrado respectivamente. Normalmente el conocimiento de una de ellas permite fácilmente el cálculo de la otra. Se llaman también sistemas *simétricos*.

En un criptosistema de *clave pública* se supone normalmente que $L = N \times N$ y cada usuario u del sistema posee un par de claves: (e_u, d_u). La primera de ellas es

pública y es la que usará cualquier otro usuario del sistema para enviar un mensaje cifrado al usuario u. La segunda de ellas es conocida sólo por el usuario u y es la que utiliza para descifrar los mensajes que recibe. Así pues las claves usadas en la línea $\ell = (a, b)$ son las del usuario b: $k_\ell = e_b$ y $k'_\ell = d_b$. Los sistemas de clave pública se llaman también *asimétricos*.

Un sistema de clave pública reposa en la condición de que, aún conocida e_u ***sea imposible en la práctica*** el cálculo de la clave privada d_u. Esta condición debe interpretarse en términos de la ***complejidad computacional*** del cálculo de d_u a partir de e_u (y como consecuencia de la cantidad de pares mensaje claro/mensaje cifrado) que conozca el criptoanalista.

Las condiciones que debe cumplir un criptosistema de clave pública fueron establecidas por *Diffie y Hellman* en 1976 (ver [DH76]) (antes de que se conociera ningún ejemplo del mismo).

Criptoanálisis. Seguridad. El objetivo del *criptoanálisis* es descubrir el contenido de un mensaje cifrado y, a ser posible, el método y la clave utilizada. El criptoanalista buscará, en general, también interferir e intervenir en el sistema de comunicación (por ejemplo falsear mensajes e identidades). En el caso clásico (sistemas de cifrado en pequeños grupos o persona a persona) es importante el hecho de que el propio sistema de cifrado puede mantenerse en secreto, sin embargo en el contexto de la criptología moderna ello no es posible, ya que habitualmente las comunicaciones se deben facilitar mediante métodos estandarizados y por ello conocidos.

Los tipos de actuación (*ataques*) que puede llevar a cabo el criptoanalista se pueden clasificar en dos grandes grupos:

A) Ataques pasivos. El critoanalista actúa a partir de un texto cifrado T intentando descubrir el mensaje en claro M y a ser posible la clave de cifrado. Ello se puede hacer de varias maneras:

1. Texto cifrado conocido: se conoce solamente el texto cifrado T.

2. Texto claro conocido: se conoce una porción del mensaje en claro y su correspondiente cifrado.

3. Texto claro elegido: el criptoanalista puede elegir un texto en claro y conocer su correspondiente cifrado.

Nótese que en los sistemas de clave pública el criptoanalista puede cifrar un texto cualquiera, de manera que los ataques del último tipo no son tan extraños como pudiese parecer a priori.

B) Ataques activos: El criptoanalista puede intervenir en el proceso de comunicación, de manera que puede intentar:

1. Sustituir la personalidad de uno de los usuarios.

2. Falsear mensajes.

La solidez de un sistema criptográfico se mide en función de la dificultad de ser atacado con exito por un criptoanalista. Hay varios conceptos de seguridad:

1. Seguridad perfecta: Si es irrompible cuando al criptoanalista se le suponen tiempo y recursos ilimitados (Ej. cifrado de Vernam).

2. Condicional: Seguro hasta que se desarrollen nuevos o mejores métodos de criptoanálisis.

3. Probable: Sistemas que no han sido rotos, pero de los que no se puede demostrar su seguridad matemáticamente (durante bastante tiempo el DES, AES).

4. Computacional: Basado en la complejidad computacional (matemáticamente probada) del sistema (Ej. el RSA).

La teoría de la seguridad perfecta fue iniciada por Shannon (1949), basada en la *entropía* del lenguaje y desarrollada a partir de la teoría de la información.

2.. Criptología de clave privada

Sustituciones y polisustituciones. Códigos homófonos. Sustituciones polialfabéticas: el cifrado de Vigenère. Trasposiciones. Cifrado en flujo: cifrado de Vernam. Secuencias cifrantes. Sucesiones recurrentes y registros lineales con desplazamiento. Métodos de cifrado en bloque. Cifrado matricial. El DES. El AES.

Métodos clásicos de cifrado

Sustituciones. Consisten en substituir el alfabeto $\mathcal{A}$ por otro alfabeto $\mathcal{B}$ (clásicamente el propio $\mathcal{A}$ permutado). En términos numéricos se trata de definir una función biyectiva $f : \mathcal{A} \to \mathcal{A}$. Un sistema sencillo de generar permutaciones en el alfabeto consiste en tomar como clave la pareja $(a, b) \in \mathbb{Z}_N^2$, siendo $a \in \mathbb{Z}_N$ un elemento inversible (i.e. tal que $\mathsf{mcd}\,(a, N) = 1$), y la operación de cifrado (*cifrado lineal*):

$$\begin{array}{rccc} f: & \mathbb{Z}_N & \to & \mathbb{Z}_N \\ & n & \mapsto & an + b\,. \end{array}$$

El ejemplo de cifrado por sustitución más antiguo (documentado) es posiblemente el conocido como cifrado de *César* que consiste en fijar la clave de cifrado $a = 1$, $b = 3$.

CRIPTOANÁLISIS:. Descrito por Al Kindi (Bagdad, S. IX), se basa en la estructura del lenguaje, ya que una permutación del alfabeto no altera la frecuencia de las letras. Un conocimiento detallado de la frecuencia de aparición de las mismas proporciona con facilidad el descifrado y la clave.

Como forma de "romper"la frecuencia de aparición de las letras se idearon varios métodos. Describiremos algunos de ellos.

Polisubstituciones. Consisten en sustituir bloques de letras en lugar de letras individualmente. Así podemos hablar de bisubstituciones, trisubstituciones,... Siguen siendo vulnerables al análisis de frecuencias, basado ahora en frecuencias de grupos de letras en lugar de frecuencias de letras.

Códigos homófonos. La idea es romper la frecuencia por el método de sustituir los simbolos más frecuentes por *varios* simbolos del alfabeto $\mathcal{B}$. Así pues, consisten en sustituir el alfabeto $\mathcal{A}$ por un alfabeto $\mathcal{B}$, con más elementos que el inicial, tomar una partición en $\mathcal{B}$, $\mathcal{P}$, y una aplicación $f : \mathcal{A} \to \mathcal{P}$. Para un símbolo $x \in \mathcal{A}$, el subconjunto $f(x) \subset \mathcal{B}$ ha de ser mayor cuanto mayor sea la frecuencia del símbolo x y el cifrado de x puede ser cualquiera de los símbolos de $f(x)$.

Substituciones polialfabéticas. Consisten en utilizar *varias permutaciones del alfabeto* (es decir varios alfabetos alternativos, que tradicionalmente son permutaciones del original) con arreglo a alguna pauta. El más famoso por su utilidad durante varios siglos es el conocido como *cifrado de Vigenère* (llamado en su día *le chiffre indéchiffrable*) que describimos a continuación.

Se fija una palabra clave: $\mathbf{k} = k_1 k_2 \dots k_s$. El mensaje original en claro $\underline{m} = m_0, \dots, m_n, \dots$ se reparte en bloques de longitud s y el caracter i de cada bloque se cifra con la cifra de César F_{k_i}, es decir,

$$m_{st+i} \mapsto m_{st+i} + k_i \ .$$

En el caso clásico el cifrado/descifrado se hace con la ayuda de un cuadro de Vigenère o de un disco de cifras con bastante rapidez.

Si la longitud de la clave $\mathbf{k}$ es la misma que la del mensaje a cifrar, es aleatoria y se utiliza una sola vez el método es incondicionalmente seguro (coincide con un cifrado de Vernam), pero tenemos el problema de transmitir la clave que es de la misma complejidad que transmitir el mensaje en claro.

CRIPTOANÁLISIS:. se basa (para textos suficientemente largos en relación a la clave) en que separando el texto cifrado en bloques de tamaño s solamente necesitamos

hacer un análisis de frecuencias para cada uno de los s mensajes parciales. El cálculo de s se estima mediante la correlación entre las tablas de frecuencias de los subtextos cifrados con la tabla de frecuencias del alfabeto. Es decir, las frecuencias de cada uno de los subtextos $ks+i$, $k = 1, \ldots$ debe ser "similar" al del alfabeto llano. Otra forma habitual consiste en un análisis de bloques de $2, 3, \ldots$ letras repetidos en el texto. Dichos bloques indican (si el texto es suficientemente largo) posibles candidatos para s (los divisores de la distancia entre repeticiones). Por otro lado, el uso de palabras clave *fáciles de recordar* (nombres propios, lugares geográficos,...) facilita en gran medida el criptoanálisis.

Transposiciones. Es un tipo de cifrado en bloques. Consiste en permutar los caracteres de una secuencia de k símbolos consecutivos mediante una permutación fija $\sigma \in S_k$. Es un método de seguridad perfecta si k coincide con la longitud del mensaje, la permutación es totalmente aleatoria y se utiliza una sola vez.

Una debilidad del sistema es que no rompe la frecuencia de las letras, por lo que una parte sustancial de la estructura del idioma permanece. Para textos largos en relación con la clave el criptoanálisis se basa en pruebas sucesivas con permutaciones, normalmente guiadas por grupos frecuentes en el idioma (por ejemplo en castellano la letra Q aparece siempre ligada a la U, en ingles el grupo TH es muy frecuente,..)

El escitalo griego (Esparta, s. V a.c.) es posiblemente el ejemplo más antiguo de cifrado por trasposición conocido.

Métodos de cifrado en flujo: cifrado de Vernam

Supondremos que los mensajes son sucesiones de digitos binarios, es decir que el alfabeto es $\mathbb{F}_2 = \{0, 1\}$.

Si el mensaje a cifrar es $M = m_1, \ldots, m_n$ se toma como clave de cifrado $K = k_1, \ldots, k_n \in \mathbb{F}_2^n$ una sucesión *aleatoria* de la *misma longitud* que el mensaje. El mensaje se cifra como:

$$c(M) = M \oplus K = m_1 + k_1, m_2 + k_2, \ldots, m_n + k_n \ .$$

Evidentemente la operación de descifrado es la misma, ya que

$$c(M) \oplus K = M \ .$$

El cifrado de Vernam es incondicionalmente seguro (es decir, es un cifrado perfecto) a condición de que la clave sea *realmente aleatoria* y se *utilice una sola vez*.

Los ***inconvenientes*** que presenta, y que lo hacen difícil de usar sin una cierta preparación previa, son la dificultad práctica de generar una secuencia realmente aleatoria y el hecho de que la transmisión de la clave (que debe ser segura) requiere el mismo esfuerzo que la del mensaje.

Habitualmente se utilizan simplificaciones del método anterior que se conocen con el nombre genérico de *técnicas de cifrado en flujo* y que consisten en

- El emisor A, mediante una *clave corta y secreta* y un *método determinista (público)* genera una secuencia binaria K llamada (*secuencia cifrante*) mediante la que cifra el mensaje como $M \oplus K$.
- El receptor B, dispone de la *misma clave* y el mismo algoritmo determinista, por tanto puede generar la misma secuencia cifrante K y recupera el mensaje como $c(M) \oplus K$.

Secuencias cifrantes. Las secuencias cifrantes, también llamadas sucesiones pseudo-aleatorias, deben cumplir una serie de requisitos para que el sistema anterior sea "razonablemente seguro", básicamente debe *emular* una sucesión realmente aleatoria (algo que es dificil de certificar). Algunas de las condiciones razonables que se testean son :

- El periodo debe ser al menos tan largo como el mensaje.
- Las cantidades de ceros y unos deben ser similares.
- Las ráfagas de ceros y unos deben aparecer distribuidas como si realmente fuesen aleatorias.

Aparte hay exigencias estadísticas en términos de la correlación entre coincidencias y discrepancias en permutaciones cíclicas.

Para la generación de secuencias cifrantes se utilizan como base los *registros de desplazamiento realimentados linealmente (LFSR: Linear Feed-Back Shift Register)* y *no linealmente (NLFSR: Non-Linear ...)*. Para una descripción detallada de los primeros, incluyendo los ataques posibles nos remitimos al apartado *Sucesiones en recurrencia lineal* del Tema *Aplicaciones de los cuerpos finitos* (asignatura de Álgebra) donde se hace un estudio detallado de los conceptos y de las matemáticas que involucran.

Métodos de cifrado en bloques

Todos ellos tienen en común que el mensaje original M se divide en *bloques* de tamaño fijo k, es decir, $M = m_1, \ldots, m_n = M_1, \ldots, M_r$, siendo $M_1 = m_1, \ldots, m_k$, $M_2 = m_{k+1}, m_{k+2}, \ldots, m_{2k}$, ... :

$$M_i = m_{(i-1)k+1}, m_{(i-1)k+2}, \ldots, m_{ik} \quad (1 \leq i \leq \lfloor r/k \rfloor)$$

Cifrado matricial (Hill). Se fija como clave secreta una matriz cuadrada C de tamaño k, con coeficientes en $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$ y que sea invertible (es decir con la condición de que su determinante sea una unidad del anillo $\mathbb{Z}_N$). El bloque $\underline{z} = (z_1, \ldots, z_k) \in (\mathbb{Z}_N)^k$ se cifra entonces mediante $c(z) = \underline{z}C$ y por lo tanto el mensaje M mediante $c(M) = c(M_1) \cdots c(M_r)$. El destinatario del mensaje conoce C, puede por tanto calcular su inversa, C^{-1}, y descifra el bloque $\underline{w} \in (\mathbb{Z}_N)^k$ mediante $d(\underline{w}) = \underline{w}C^{-1}$.

El sistema anterior presenta algunas variantes, citaremos algunas de ellas:

Cifrado afín: En este caso la clave consiste en un par $(C, \underline{x})$, donde C es una matriz inversible $k \times k$ en $\mathbb{Z}_N$ y $\underline{x} \in (\mathbb{Z}_N)^k$. La operación de cifrado (respectivamente descifrado) para un bloque $\underline{z} \in (\mathbb{Z}_N)^k$ es en este caso:

$$\underline{z} \mapsto c(\underline{z}) = \underline{z}C + \underline{x} \qquad (resp. \underline{w} \mapsto d(\underline{w}) = (\underline{w} - \underline{x})C^{-1})$$

Cifrado afín variable: Para la clave se toma una terna $(C, D, \underline{x})$ formada por dos matrices inversibles y $\underline{x} \in (\mathbb{Z}_N)^k$. El primer bloque se cifra mediante $c(M_1) = M_1C + \underline{x}D$ y los restantes:

$$c(M_i) = M_iC + M_{i-1}D$$

Las técnicas descritas en el cifrado en flujo se pueden apllicar de manera natural para generar variantes de cifrado en bloque que extienden la anterior, para ello solo debemos poner matrices fijas $C_1, \ldots, C_p$ en lugar de coeficientes en un LFSR y $c(M_p) = M_pC_p + \cdots + M_1C_1$

Seguridad de los métodos matriciales: El cifrado matricial es muy vulnerable para los ataques a texto claro conocido. Es claro que si conocemos el cifrado $\underline{w}_1, \ldots, \underline{w}_k$ de k bloques en claro $\underline{z}_1, \ldots, \underline{z}_k$ que sean linealmente independientes podemos determinar la matriz de cifrado resolviendo los k sistemas de ecuaciones con k incognitas resultantes de plantear:

$$\underline{z}_i \begin{pmatrix} X_{11} & \ldots & X_{1k} \\ \vdots & & \vdots \\ X_{k1} & \ldots & X_{kk} \end{pmatrix} = \underline{w}_i$$

Para el cifrado afín son necesarios $k+1$ bloques en claro y sus correspondientes cifrados.

El DES (Data Encryption Standard). El sistema de cifrado DES fué adoptado en 1977 por el National Bureau of Standards como estandar federal en los Estados Unidos para aplicaciones no clasificadas. Es una extensión de un sistema desarrollado por IBM llamado *Lucifer* y utiliza una clave de 56 bits. Hasta 1998 se ha estando utilizando sistemáticamente, en esa fecha se ha sustituido por el *AES* (*Advanced Encryption Standard*) que utiliza claves de 128 bits que se pueden extender de ser necesario en el futuro hasta los 256. El DES se implementa bien mediante hardware (hay varias compañias que fabricaron microchips para ello), bien mediente software (por ejemplo en los navegadores). Se pueden encontrar varios programas de cifrado con DES en la red.

El sistema parte de una clave de 64 bits, de los cuales se eliminan los 8 bits de paridad. Por lo tanto a todos los efectos posemos suponer que la clave original K está formada por 56 bits. La clave original K genera sucesivamente 16 claves $K_1, K_2, \ldots, K_{16}$ todas ellas de longitud 48.

El sistema cifra bloques de información M de 64 bits, en primer lugar aplica una permutación P fija y el bloque permutado se divide en dos subbloques (L_0, R_0) cada uno de 32 bits. Posteriormente, para $i = 1, \ldots, 16$ se aplica el proceso siguiente:

$$L_i := R_{i-1}$$
$$R_i := L_{i-1} \oplus f(R_{i-1}, K_i)$$

dónde $f(R_{i-1}, K_i)$ es un bloque de 32 bits calculado en función de K_i y R_{i-1}. Al resultado final (L_{16}, R_{16}) se le aplica la permutación P^{-1} y el resultado final es el cifrado del bloque M.

Generación de las subclaves:. La clave original de 64 bits se reduce a dos bloques de 28 bits, (C_0, D_0) convenientemente permutados por la permutación $PC1$. Cada uno formados por los bits de posiciones (nótese que no se conserva su orden natural, si no que se permutan):

$$C_0 = [57, 49, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35, 27,$$
$$19, 11, 3, 60, 52, 44, 36]$$
$$D_0 = [63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37,$$
$$29, 21, 13, 5, 28, 20, 12, 4]$$

La clave K_i se forma entonces mediante:

$$C_i = LS(C_{i-1}) \ , \quad D_i = LS(D_{i-1}) \ , \quad K_i = PC2(C_i, D_i)$$

siendo:

- LS una permutación circular a la izquierda de una posición para $i = 1, 2, 16$ y de dos posiciones para los restantes índices.
- $PC2$ es una permutación junto con una selección de 48 de los 56 bits.

La función $f(R_{i-1}, K_i)$. La entrada de la misma son los 32 bits de R_{i-1} y los 48 bits de K_i. La función f viene dada por las siguientes operaciones:

- R_{i-1} se expande en $E(R_{i-1})$ que está formado por 48 bits.
- $B := E(R_{i-1}) \oplus K_i$
- $B = (B^1, \ldots, B^8)$ se reparte en 8 bloques de 6 bits cada uno.
- Se aplica a B^i la función S_i, el resultado es $S_i(B^i)$ que está formado por 4 bits.
- La salida $S(B) = (S_1(B^1), \ldots, S_8(B^8))$ se permuta por una permutación Q proporcionando $f(..)$.

Las *cajas* S_i, $i = 1, \ldots, 8$ son una de las razones de la robustez del método. Cada una de ellas, S, es una tabla de doble entrada formada por 16 columnas (numeradas

en binario $0000, 0001, \ldots$) y 4 filas (numeradas $00, 01, 10, 11$). Cada fila de la tabla contiene una permutación de los enteros $0, 1, \ldots, 15$. Si el bloque de entrada es $b = (b_6 b_5 \ldots b_1)$, la salida $S(b)$ viene dada por la representación binaria del entero que corresponde a la fila $b_6 b_1$ y la columna $b_5 b_4 b_3 b_2$.

Criptoanálisis. El criptoanálisis del DES se ha realizado de varias maneras, algunas más eficaces que otras. Desde el sistema de fuerza bruta (probando todas las claves posibles) hasta métodos bastante elaborados basados casi todos ellos en ataques con grandes cantidades de texto a texto claro elegido (criptoanálisis diferencial –Biham y Shamir– y lineal –Matsui–).

En Enero de 1997, durante la conferencia anual de la compañía RSA, se ofreció un premio de 100.000 dólares por romper el DES. El reto se llevó a cabo el 17 de Junio de 1997 mediante 75.000 ordenadores coordinados por Internet que consiguieron romper el sistema después de explorar aproximadamente un 25 % de las claves posibles (aproximadamente $17 \cdot 10^{15}$).

El AES (Advanced Encryption Standard). En 1997 el NIST (National Institute for Standars and Technology) convocó públicamente un concurso para la adopción de un nuevo estándar de cifrado en bloque simétrico que sustituyese al DES. El ganador del mismo fué el sistema llamado RIJNDAEL, desarrollado por V. Rijmen y J. Daemen (Univ. de Lovaina). Describiremos brevemente como funciona este nuevo estándar.

En primer lugar, el sistema admite claves de longitudes 128, 192 o 256 bits, y cifra bloques de las mismas longitudes (ambas se pueden fijar independientemente). La primera característica llamativa del sistema es que maneja toda la información como una secuencia de bytes, no de bits. Cada byte (8 bits) se trata como un elemento del cuerpo $\mathbb{F}_{256}$ construido a partir del polinomio primitivo $1 + X + X^3 + X^4 + X^8$. Supongamos que se desea cifrar un mensaje de longitud 192 (es decir, 24 bytes) mediante una clave de 128 bits (16 bytes). Tanto la clave como el mensaje se usan en forma matricial, con entradas en $\mathbb{F}_{256}$ y siempre con 4 filas (el número de columnas depende del tamaño de la clave y del mensaje). En concreto la clave K de 16 bytes se representa como una matriz K, de tamaño 4×4 y el mensaje de 24 bytes como una matriz M de tamaño 4×6.

El cifrado consiste esencialmente en un número variable de vueltas R (que depende de las longitudes de la clave y el mensaje, en nuestro ejemplo sería $R = 12$) de un algoritmo básico que utiliza como entradas (para la vuelta i-ésima) una *clave de vuelta* K_i y la matriz saliente de la vuelta anterior M_{i-1}.

Las sucesivas claves de vuelta se construyen mediante un algoritmo recursivo, bastante complejo y que incluye en particular un proceso de expansión hasta el tamaño de clave requerido (para adaptarlo al tamaño del mensaje). No entraremos en el algoritmo de generación de dichas claves, de manera que suponemos que, a partir de K y del tamaño del mensaje M, se han formado claves $K_0, \ldots, K_r \ldots$,

siendo K_i (la clave de la vuelta i-ésima) una matriz del mismo tamaño que el mensaje original.

El algoritmo de cifrado es el siguiente:

Algoritmo (Cifrado Rijndael).
Entrada: El mensaje M y las claves $K_0, \ldots, K_r$.
Iniciamos con $M' := M + K_0$
Para $i = 1, \ldots, 15$ hacemos:

1. $M' :=$ ByteSub(M').
2. $M' :=$ ShiftRow(M').
3. $M' :=$ MixColumn(M').
4. $M' :=$ AddRoundKey(M', K_i).

$M' :=$ ByteSub(M')
$M' :=$ ShiftRow(M')
$M' :=$ AddRoundKey(M', K_i)

Obsérvese que hay una operación previa, después 15 vueltas idénticas (con claves de vuelta diferentes) y, finalmente, una vuelta más pero sin una de las etapas. En cada vuelta la clave solo interviene en la última operación. Cada una de las operaciones en los pasos $1-4$ es como sigue:

ByteSub: Se una operación sobre cada una de las entradas (bytes) de la matriz M' que consiste, para $a = (a_1, \ldots, a_8) \in \mathbb{F}_{256}$ en:
- Sustituir a por su inverso, $a := 1/a$.
- Hacer un cifrado afín de Hill, $a \mapsto a \cdot A + b$, siendo A una matriz binaria fija de tamaño 8×8 y b un vector fijo de 8 bits.

ShiftRow: Las filas de la matriz M' se permutan por medio de una permutación que depende del número de columnas de M' (que es el mismo que el de M).

MixColumn: Opera sobre cada columna $c_j = (c^j{}_0, c_1^j, c_2^j, c_3^j)$ de la matriz M'. Para ello se escribe la columna en forma polinómica: $c_j(T) = c_0^j + c_1^j T + c_2^j T^2 + c_3^j T^3$ y se hace la transformación

$$c_j(T) := c_j(T)c(T) \bmod (T^4 + 1)$$

siendo $c(T)$ un polinomio fijo de grado 3.

AddRoundKey: La matriz M' se sustituye por la matriz $M' + K_i$.

Se puede encontrar toda la información detallada, en particular el algoritmo para la generación de las distintas claves de vuelta, en las páginas oficiales de Rijndael: http://www.esat.kuleuven.ac.be/~rijmen/rijndael/index.html y de NIST: http://www.nist.gov/aes.

3.. Criptología de clave pública

Cifrado asimétrico: una necesidad. Condiciones de Diffie-Hellman. La seguridad computacional. Factorización: RSA. El problema del logaritmo discreto: El Gamal. El problema de las mochilas: Merkle-Hellman.

En el primer tema ya comentamos esencialmente en qué consiste un sistema criptográfico de clave pública (o asimétrico). El modelo fué creado por W. Diffie y M.E.Hellman en 1975 y publicado en 1976 en el artículo titulado *New Directions in Cryptography* (v. [DH76]). La criptografía de clave pública no nace con la idea de sustituir a los métodos de cifrado con clave privada, ni hoy día es ese su interés primordial, si no que pretende resolver con ella algunos de los grandes problemas que presenta el uso masivo de cifrado de clave privada en redes vulnerables (como lo son las electrónicas). Los dos problemas principales son:

Intercambio de claves: El cifrado de clave privada requiere que dos usuarios concierten una (unas) determinada clave y que ésta solo sea conocida por ellos. Ahora bien, no lo pueden hacer a través del canal, ya que este es vulnerable y en un sistema electrónico el intercambio presencial no tiene sentido.

Autentificación e integridad: En relación con el intercambio de claves uno de los problemas que surgen es el de la autentificación, tanto del interlocutor como del mensaje. Es decir debemos garantizarnos que nuestro interlocutor es quién dice ser, de la misma manera que debemos tener la garantía de que un determinado mensaje (o clave) es legítimo y no ha sido sustituido o alterado por un tercero.

Un sistema criptográfico de clave pública no presenta problemas de intercambio de claves, ya que cada usuario puede, si dispone de suficiente capacidad de cálculo, generar sus propias claves y su clave privada no sale nunca del ámbito privado del usuario que la genera. Veremos después como se pueden resolver los problemas anteriores mediante el uso de la criptografía de clave pública. La razón por la cual los sistemas de clave pública no se suelen usar en los intercambios masivos de información es la rapidez, por ejemplo, el AES (que es de clave privada) es centenares de veces más rápido que el sistema RSA (de clave pública). Los sistemas de clave pública se suelen usar en sistemas mixtos, por ejemplo, para intercambiar claves y autentificar usuarios previamente a transmitir la información con un sistema de clave privada.

En el artículo ya citado Diffie y Hellman establecen una serie de condiciones que deben ser satisfechas por un criptosistema de clave pública, dichas condiciones reciben el nombre de ***condiciones de Diffie y Hellman*** y son las siguientes:

1. El cálculo de las llaves, pública y privada, debe ser computacionalmente sencillo.
2. El proceso de cifrado debe ser computacionalmente sencillo.

3. El proceso de descifrado, conociendo la llave secreta, debe ser también computacionalmente sencillo.
4. La obtención de la llave secreta, a partir de la pública, debe ser un problema computacionalmente imposible.
5. La obtención del mensaje en claro, conociendo el mensaje cifrado y la llave pública, debe asimismo ser computacionalmente imposible.

Para el cumplimiento de las condiciones de Diffie y Hellman, es necesario lo que se denomina una *Función Trampa*. En primer lugar una aplicación $f : A \to B$ se denomina ***Función de una vía*** si:

a) para $x \in A$ es computacionalmente sencillo calcular $f(x)$
b) dado $y \in \mathrm{Im}(f)$, es computacionalmente imposible, en general, determinar un elemento $x \in A$ tal que $f(x) = y$.

Una ***función trampa*** es una función de una vía, f, tal que existe una información complementaria secreta (***la trampa***), que permite calcular eficientemente un elemento $x \in A$ tal que $f(x) = y$. La existencia de funciones de una vía no está demostrada matemáticamente, sin embargo, a partir de la teoría de la complejidad, se tiene el convencimiento heurístico de la existencia de un buen número de ellas. Entre ellas, la función que envía dos primos en su producto es la más conocida (para calcular la inversa de un número n sería necesario factorizar n, problema que si los primos son grandes se considera computacionalmente intratable).

El sistema Criptográfico RSA

El sistema criptográfico RSA fué propuesto por Rivest, Shamir y Adleman (de ahí su nombre) en 1978 y es el primer criptosistema de clave pública construido. A fecha de hoy sigue siendo el más utilizado. La (probable) función de una vía en que se apoya es el producto de números enteros, ya que la factorización se considera un problema intratable.

Pasemos ahora a describir el sistema RSA:

Generación de las claves:

1.1. Cada usuario i del sistema elige una pareja de números primos p_i, q_i
1.2. Calcula $n_i = p_i q_i$ y la función de Euler de n_i, $\phi(n_i) = (p_i - 1)(q_i - 1)$.
1.3. Elige arbitrariamente e_i, $0 < e_i < \phi(n_i)$, tal que $\mathrm{mcd}(e_i, \phi(n_i)) = 1$ y calcula su inverso modular $d_i \equiv e_i^{-1} \bmod \phi(n_i)$.
1.4. Toma como clave pública (n_i, e_i) y como su clave privada d_i.

Cifrado: Un mensaje M, $0 < M < n_i$ se cifra mediante:

$$\begin{array}{rcl} \mathbb{Z}/n_i\mathbb{Z} & \longrightarrow & \mathbb{Z}/n_i\mathbb{Z} \\ M & \mapsto & M^{e_i} \equiv C \bmod n_i \end{array}$$

Descifrado: Un mensaje cifado C, $0 < C < n_i$ se descifra mediante:

$$\begin{array}{rcl} \mathbb{Z}/n_i\mathbb{Z} & \longrightarrow & \mathbb{Z}/n_i\mathbb{Z} \\ C & \mapsto & C^{d_i} \equiv M \bmod n_i \end{array}$$

Protocolo de intercambio de mensajes. La descripción que hemos hecho contiene el método de transmitir un mensaje numérico. Veamos ahora como se establece un método que permita enviar y descifrar un mensaje en forma de texto. Por supuesto no es el único, hay muchas formas de realizar esta tarea.

Supongamos que las claves de un usuario A son (n, e, d) y que otro usuario B desea enviarle un mensaje que consiste en un texto T escrito en caracteres ascii. El usuario B procede como sigue:

1. Calculamos el mayor entero k tal que $256^k < n$.
2. Troceamos el mensaje T en bloques de k caracteres cada uno, $T = [T_1, T_2, \ldots, T_r]$.
3. Cada uno de los bloques T_i proporciona un entero positivo M_i, $0 \leq M_i < 256^k < n$. Por tanto la lista T se transforma en una lista $M = [M_1, \ldots, M_r]$.
4. Cada uno de los enteros M_i se cifra mediante la clave pública: $C_i := M_i^e \bmod n$. Se obtiene de salida una lista $C = [C_1, \ldots, C_r]$ con $0 \leq C_i < n$ $(1 \leq i \leq r)$.
5. Cada uno de los C_i proporciona una sucesión de $k+1$ caracteres ascii, S_i, ya que $256^k < n \leq 256^{k+1}$. Así pues obtenemos $S = [S_1, \ldots, S_r]$.
6. Se concatenan los bloques $S_1, \ldots, S_r$ en un texto R que es el texto cifrado.

Obsérvese que el texto cifrado es más largo que el texto original, pues cada bloque aumenta en un carácter.

Cuando el usuario A recibe el mensaje R procede casi siguiendo los mismos pasos anteriores, las únicas diferencias aparecen en las siguientes etapas:

2. El mensaje R se trocea en bloques de tamaño $k+1$.
4. Se usa la clave privada d en lugar de la pública e.
5. Los números resultantes de la operación anterior son los enviados $M_1, \ldots, M_r$, por tanto $M_i < 256^k$ y corresponden a los bloques T_i de k caracteres del mensaje enviado.

Complejidad de las operaciones del RSA. Para asegurarnos que el sistema descrito realmente es eficaz es imprescindible antes verificar que, en determinadas condiciones, satisface las condiciones de Diffie y Hellman. Analizaremos el problema en el capítulo siguiente, pero antes avancemos algunas de las pruebas que necesitaremos.

En primer lugar, observemos que el método reposa en la certidumbre de que, en general, es difícil factorizar un número que es producto de dos primos. Evidentemente esto no es verdad en general, los primos en cuestion deben satisfacer algunas condiciones suplementarias. En general los métodos de factorización eficaces existentes imponen condiciones sobre el tipo de primos que se admiten. Digamos que las parejas de primos que evitan los métodos de factorización conocidos se llamarán ***primos seguros*** y notemos que, claramente, este concepto es variable por cuanto métodos nuevos de factorización impondrán nuevas condiciones. Algunas de las propiedades de un primo seguro son evidentes, por ejemplo, han de ser "grandes"; a día de hoy se estima que los primos seguros han de tener un mínimo de 100 dígitos.

Además de esta primera condición es necesario analizar las distintas etapas con el fín de asegurarnos que las operaciones que han de ser rápidas efectivamente lo son y que las que han de ser intratables también. Por ejemplo, es claro que, como (n_i, e_i) son públicos, si a partir de ellos se pudiese calcular $\phi(n_i)$ fácilmente entonces el método no serviría. Afortunadamente es sencillo probar que, si n es el producto de dos primos, entonces el problema de calcular $\phi(n)$ es computacionalmente equivalente a factorizar n.

Separaremos las comprobaciones necesarias en dos grupos. El primero incluye todas las operaciones aritméticas que han de realizarse de manera eficiente, suponiendo que ya hemos elegido dos primos seguros. Dejaremos para el siguiente tema el problema de la factorización y, como consecuencia, de la elección de los primos seguros; así como un pequeño análisis de algunas precauciones más que se deben tomar.

En cuanto a las operaciones que incluimos en el primer grupo, debemos analizar:

a. La multiplicación de números grandes.

b. El cálculo del máximo común divisor y de un inverso modular.

c. El cálculo de las exponenciales modulares.

MULTIPLICACIÓN. Una de las operaciones que debemos repetir varias veces es la multiplicación de enteros grandes, por lo que no está de más comprobar que dicha operación se puede hacer de manera eficaz. Es sencillo comprobar que la multiplicación, por el método estandar, de dos enteros binarios de longitudes k y ℓ precisa un máximo de $k\ell$ sumas de un bit (operaciones bit), por lo tanto el producto de dos números enteros con a lo sumo r cifras decimales requiere a lo más $\log^2 r$ operaciones, es decir, es del orden $\mathcal{O}(\log^2 r)$ y por lo tanto es una operación eficiente. Actualmente hay algoritmos de multiplicación más rápidos, el más conocido es el método de Karatsuba (cuya descripción es completamente asequible) que permite multiplicar dos enteros de longitud binaria menor o igual que k en $\mathcal{O}(k^{1,59})$ operaciones bit. El método más rápido se debe a Schönhage y Strassen y su complejidad temporal es $\mathcal{O}(k \log k \log\log k)$.

Dado que el tiempo de la multiplicación está ya estimado y precisaremos repetir muchas veces dicha operación denotaremos por $M(r)$ la complejidad de multiplicar dos enteros menores o iguales que r. Es sencillo convencerse de que la complejidad

de la división es la misma que la del producto, es decir, la división requiere también $\mathcal{O}(M(r))$ operaciones. Por otro lado la conversión de binario a decimal (o viceversa) de un entero de longitud r es también de complejidad $\mathcal{O}(M(r))$.

ALGORITMO DE EUCLIDES EXTENDIDO. Como ya sabemos el algoritmo de Euclides extendido sirve tanto para calcular el máximo común divisor de dos enteros a, b (que supondremos $a \geq b$) como para calcular el inverso multiplicativo de b módulo a. En lugar de describirlo, escribiremos un programa en Maple que lo calcula:

Programa: Euclides extendido.

```
>euclides:=proc(a::integer,b::integer)
> local x,y,r;  x:=[1,0]; y:=[0,1]; r:=[a,b];
> while irem(r[1],r[2],'q')<>0 do
>      r:=[r[2],r[1]-q*r[2]];
>      x:=[x[2],x[1]-q*x[2]];
>      y:=[y[2],y[1]-q*y[2]];
> od;
> [r[2],x[2],y[2]];
> end:
```

Es sencillo comprobar que si los restos sucesivos del algoritmo son $r_0 = b > r_1 > \cdots > r_s$ se tiene que $r_{j+2} < r_j/2$. Por tanto, el número de etapas del algoritmo es del orden $\mathcal{O}(\log a)$ y cada una de ellas precisa $\mathcal{O}(\log^2 a)$ operaciones bit, por tanto la complejidad de ambas operaciones (cálculo del máximo común divisor o inversos módulo a) es $\mathcal{O}(\log^3 a)$, o si preferimos $\mathcal{O}(\log a M(a))$.

Se puede hacer un análisis más detallado del algoritmo y con ciertas mejoras se puede mejorar su complejidad hasta $\mathcal{O}(M(a) \log\log a)$.

Nótese que la búsqueda aleatoria del entero e_i queda también resuelta ya que, elegido un entero r al azar se debe comprobar que $\mathsf{mcd}\,(r, \phi(n)) = 1$. Esta operación es rápida, como acabamos de comprobar, y la probabilidad de que el número elegido no sea primo con $\phi(n)$ es muy pequeña.

EXPONENCIACIÓN MODULAR. Las operaciones de cifrado y descifrado son iguales y consisten en calcular $a^b \bmod n$ para enteros n, a y b. Para realizar dicha operación existe un conocido algoritmo llamado *algoritmo de cuadrados iterados* o de *exponenciación modular*. Se basa en que si hacemos la división entera de b entre 2: $b = b_0 + 2b'$, tendremos que $a^b = a^{b_0}(a^2)^{b'}$. Ahora podemos iterar, sustituyendo a por a^2 y acumulando en $r = a^{b_0}$ el resultado parcial. El siguiente programa de Maple lleva a cabo la operación descrita usando para ello $\log b$ iteraciones y en cada una de ellas una división y dos multiplicaciones:

Programa: Exponenciación modular.

```
>expmod:=proc(a,b,n)
>  local i, r, x;  i:=b; r:=1; x:=a mod n;
>  while i>0 do
>         if irem(i,2)=1 then r:=r*x mod n; fi;
>         x:=x*x mod n; i:=iquo(i,2);
>  od;
>  r; end:
```

Por tanto la complejidad de la exponenciación modular es a lo sumo de $\mathcal{O}(\log b \log^2 n) \leq \mathcal{O}(\log^3 n)$ o, mejor aún, $\mathcal{O}(\log n M(n))$.

El problema del logaritmo discreto

En un grupo abeliano finito G (habitualmente pensamos en el grupo multiplicativo del anillo $\mathbb{Z}/m\mathbb{Z}$ o en el grupo multiplicativo de un cuerpo finito) es sencillo calcular a^x para un elemento $a \in G$ y un entero x. Sin embargo, conocidos $a, y \in G$, y una potencia de a, en general es muy dificial calcular $x \in \mathbb{N}$ tal que $a^x = y$. Dicho problema se conoce con el nombre del ***Problema del Logaritmo Discreto***.

Intercambio de claves de Diffie-Hellamn. El problema del logaritmo discreto fué utilizado por Diffie y Hellman como un medio seguro para intercambiar claves entre dos usuarios a través de un canal inseguro. El sistema para ello es relativamente simple: supongamos que los usuarios A y B desean ponerse de acuerdo en una clave común para un sistema de intercambio de clave privada. Supongamos que dicha clave debe ser un elemento de un cuerpo finito $\mathbb{F}_q$, siendo q potencia de un primo p (ambos pueden ser públicos). El método es el siguiente:

1. Conciertan un generador β del grupo $\mathbb{F}_q^*$.
2. A elige un entero a (que mantiene en secreto), calcula β^a y lo transmite a B. B hace lo propio, elige b y transmite β^b.
3. A y B toman como clave secreta β^{ab}.

La hipótesis de Diffie y Hellman era que es computacionalmente intratable calcular β^{ab} conocidos β^a y β^b (que pueden ser conocidos por cualquier escucha en el canal). Se considera (aunque no está probado) que el problema es equivalente a calcular a a partir de β^a (o b a partir de β^b), es decir, equivalente a resolver el problema del logaritmo discreto.

Basados en el problema del logaritmo discreto hay varios sistemas de clave pública (ElGamal, Massey-Omura) y de firma digital, notablemente el sistema DSS (Digital Standar Signature) propuesto por el NIST (National Institute of Standards and Technology, USA) como sistema de firma digital estándar.

El Sistema Criptográfico de ElGamal. Todos los usuarios del sistema usarán el mismo cuerpo finito $\mathbb{F}_q$ ($q = p^m$ debe ser "muy grande") y un elemento $g \in \mathbb{F}_q^*$ (preferiblemente un generador del grupo cíclico $\mathbb{F}_q^*$, aunque no es imprescindible).

Generación de claves: Cada usuario elige un entero n con $0 < n < q-1$ (su clave secreta) y calcula g^n (su clave pública).

Cifrado: Los mensajes se suponen elementos del cuerpo $\mathbb{F}_q$. El remitente elige un entero k arbitrario y cifra el mensaje M como el par $(C', C) = (g^k, Mg^{nk})$.

Descifrado: El receptor calcula $(C')^n$ y recupera el mensaje original como

$$M = C/(C')^n \ .$$

Los sistemas criptográficos basados en el logaritmo discreto son seguros hasta el momento, aunque conviene ser precavidos, pues hay métodos que, en condiciones particulares, son eficientes para su cálculo. Por ejemplo, el método index-calculus permite el cálculo en cuerpos del tipo $\mathbb{F}_{2^m}$ cuando m no es demasiado grande. De hecho un sistema patentado por Hewlett-Packard sobre $\mathbb{F}_{2^{127}}$ dejó de ser seguro a partir del mencionado algoritmo. Actualmente se recomiendan valores de $m > 1000$.

El problema de las mochilas

Otro problema $\mathcal{NP}$, candidato a función de una vía, es el denominado ***problema de las mochilas***: Sea $\{a_1, a_2, \ldots, a_n\}$ un conjunto de números naturales positivos o *pesos*. El problema de las mochilas consiste en, dado un número natural S determinar, si existe, un subconjunto de $\{a_i\}$, la suma de cuyos elementos sea S, es decir, determinar $\{x_1, x_2, \ldots, x_n\}$, $x_i \in \{0, 1\}$, tales que $S = \sum_{i=1}^{n} x_i a_i$.

En alguno casos el problema tiene una solución trivial, por ejemplo si la sucesión de pesos $\{a_1, a_2, \cdots, a_n\}$ es *supercreciente* (es decir, si verifica $a_i > \sum_{j=1}^{i-1} a_j$ para $i = 2, 3, \cdots, n$) se procede de la siguiente forma:

Algoritmo.

Completamos con $a_{n+1} = \sum_{i=1}^{n} a_i$ y $a_0 = 0$.
Iniciamos con $R = S$, $x = [0, \ldots, 0]$ de longitud n y $i = n + 1$.
Mientras $R \neq 0$ hacer

> Si $R > a_i$ el problema no tiene solución. En otro caso:
> Para $j = i - 1, \ldots, 1$ hacer:
> > Si $R > a_j$ entonces $R := R - a_j$; $x[j] := 1$; $i := j$.

Devolvemos x.

Sistema de Merkle-Hellman. Basado en el problema anterior, Merkle y Hellman propusieron en 1978 el criptosistema de clave pública que lleva su nombre:

Generación de claves: Cada usuario selecciona:

- Una sucesión supercreciente $\{b_1, b_2, \ldots, b_n\}$
- Dos números naturales M y W, primos entre sí y tales que $M > \sum_{i=1}^{n} b_i$ y $W < M$.
- Una permutación π de $\{1, 2, \ldots, n\}$.
- Para $i = 1, \ldots, n$ hacemos $a'_i \equiv b_i W \mod M$ y $a_i = a'_{\pi(i)}$.
- La clave pública es la sucesión $\{a_1, \ldots, a_n\}$.
- La clave privada es $(\{b_1, \ldots, b_n\}, M, W, \pi)$.

Cifrado: Un mensaje M con $0 \leq M < 2^n$ se cifra mediante:

1. Se escribe el mensaje en binario $M = [x_1, \ldots, x_n]$.
2. El mensaje cifrado es $C = \sum_{i=1}^{n} x_i a_i < M$.

Descifrado:

1. Calculamos $S \equiv CW^{-1} \mod M$. Nótese que

$$S \equiv \sum x_i a_i W^{-1} = \sum x_i a'_{\pi(i)} W^{-1} \equiv \sum x_i b_{\pi(i)} \equiv \sum x_{\pi^{-1}(i)} b_i \ (\mod M).$$

2. Calculamos $y = [y_1, \ldots, y_n]$ con $S = \sum_{i=1}^{n} y_i b_i$.
3. Permutamos la lista y, $x = \pi[y]$ y recuperamos el mensaje como $M = \sum_{i=1}^{n} x_i 2^{i-1}$.

El método de Merkle-Hellman (más simple de implementar que el RSA y unas 100 veces más rápido) suscitó grandes esperanzas. Sin embargo fué roto por Shamir en 1982. Trás la rotura Merkle y Hellman propusieron un sistema con varias transformaciones o *iteraciones* modulares para hacerlo más *robusto* pero este sistema iterado de Merkle-Hellman ha sido también roto (1984). Otras variantes del problema de las mochilas (como el de Chor-Rivest) no han sido rotas hasta la fecha.

Sistema de Chor-Rivest*. Daremos sólo una aproximación a dicho sistema, menos eficaz que el método completo.

Generación de claves: Cada usuario selecciona:

- Tomamos un primo p y un entero r de manera que $q = p^r - 1$ tenga sus factores primos relativamente pequeños.
- Elegimos al azar un polinomio $F \in \mathbb{F}_p[X]$ irreducible de grado r, un generador multiplicativo α de $\mathbb{F}_q$ y un entero m. Denotamos por x la clase de X en $\mathbb{F}_p[X]/(F) \simeq \mathbb{F}_q$.
- Para un entero $n < p, r$ calculamos $b_1, \ldots, b_n$ tales que $\alpha^{b_j} = x + (j-1)$.
- Tomamos una permutación π de n elementos y tomamos $a_i \equiv b_{\pi(j)} + m \bmod (q-1)$.
- La clave pública es $\{a_1, \ldots, a_n\}$.

Para la elección de p y r son suficientes dos o tres cifras, los propuestos por Chor y Rivest eran $p = 197$, $r = 24$. Las condiciones sobre la factorización de $q - 1$ tienen por objetivo que el cálculo de los logaritmos discretos en $x + (j-1)$ sea un problema accesible.

Cifrado: Un mensaje M formado por una sucesión de n bits $M = x_1, \ldots, x_n$ se cifra en el par $(C, C') = (\sum_{i=1}^{n} x_i a_i, \sum x_i)$.

Descifrado:

1. Calculamos $\alpha^{C - mC'}$ y su representación polinómica $G(X) \in \mathbb{F}_p[X]_m$. Nótese que
$$\alpha^{C-mC'} = \prod \alpha^{x_j b_{\pi(j)}} = \prod (x + \pi(j))^{x_j} \equiv \prod (X + \pi(j))^{x_j}$$
2. Factorizamos $G(X)$ (hay algoritmos eficientes para ello).
3. Como $G(X) = \prod (X + \pi(j))^{x_j}$, hemos recuperado el mensaje $M = x_1, \ldots, x_n$.

Residuosidad cuadrática*

Si p es un número primo sabemos que es sencillo saber si un entero x es un residuo cuadrático módulo p, basta para ello calcular el símbolo su Legendre, $\left(\frac{x}{p}\right)$, ya que se tiene

$$\left(\frac{x}{p}\right) = 1 \iff x \in Q_p$$

(Q_p denota el conjunto de cuadrados módulo p). Sea ahora n un entero cualquiera, formalmente el símbolo de Legendre se extiende mediante el símbolo de Jacobi, si $n = p_1 \cdot p_2 \cdots p_r$ es la factorización de n en producto de primos, entonce se define

$$\left(\frac{x}{n}\right) = \left(\frac{x}{p_1}\right) \cdot \left(\frac{x}{p_2}\right) \cdots \left(\frac{x}{p_r}\right) .$$

Sigue siendo verdad que si y es un cuadrado módulo n entonces $\left(\frac{y}{n}\right) = 1$, pero ahora existen enteros y con $\left(\frac{y}{n}\right) = 1$ que no son cuadrados módulo n. El problema de la residuosidad cuadrática consiste precisamente en, dado y con $\left(\frac{y}{n}\right) = 1$ decidir si $y \in Q_n$ o no. Se trata de un problema $\mathcal{NP}$ y se cree que es equivalente al problema de factorizar n.

Basándose en dicho problema el sistema criptográfico más conocido es el sistema de cifrado probabilístico de Goldwasser y Micali:

Generación de claves: Cada usuario sigue las siguientes etapas:

- Selecciona dos primos grandes p y q de tamaño similar. Calcula $n = pq$.
- Selecciona y, $1 < y < n$ de manera que $y \notin Q_n$ pero $\left(\frac{y}{n}\right) = 1$.
- Clave pública $[n, y]$. Clave secreta $[p, q]$.

Cifrado: Un mensaje es una sucesión binaria $m = (m_1, \ldots, m_k)$. Supongamos que $[n, y]$ es la clave pública del destinatario. Entonces:

- Para $i = 1, \ldots, k$ hacemos:
  Elegimos aleatoriamente x_i, $1 < x_i < n$ y primo con n.
  $z_i = x_i^2 y^{m_i} \bmod n$
- Enviamos $(z_1, \ldots, z_n) \in (\mathbb{Z}_n)^k$

Descifrado: Recibido el mensaje $(z_1, \ldots, z_n)$. Para $i = 1, \ldots, k$ hacemos:
Calculamos $e = \left(\frac{z_i}{p}\right)$, $e' = \left(\frac{z_i}{q}\right)$.
Si $e = e' = 1$ entonces $m_i := 0$. En otro caso $m_i = 1$.
Hemos recuperado el mensaje llano: $(m_1, \ldots, m_k)$.

Nótese que el sistema anterior cifra un mismo mensaje de formas diferentes cada vez. El sistema se considera semánticamente seguro (quiere decir que en un tiempo polinómico un posible atacante pasivo no obtiene ninguna información a partir del mensaje cifrado que no pueda obtener sin él). El mayor inconveniente que presenta es que el mensaje cifrado es mucho más largo que el mensaje llano, ya que cada bit se sustituye por un entero $< n$. Sobre la complejidad del sistema (básicamente el cálculo de símbolos de Jacobi) nos remitimos al tema siguiente, donde se usan en el algoritmo de Solovay-Strassen.

Raíces cuadradas modulares*

Si a es un residuo cuadrático módulo n el problema de encontrar las soluciones de la congruencia $x^2 \equiv a \bmod n$ es un problema $\mathcal{NP}$-completo y proporciona otro candidato a función de una vía. Se cree que es un problema equivalente a factorizar el entero n, aunque no ha sido probado.

En el caso en que $n = p$ es un número primo la congruencia anterior se puede resolver de manera eficiente. Un método para ello es el siguiente algoritmo (Adleman, Manders, Miller):

Algoritmo.
Datos de entrada: $(a,p) \in \mathbb{Z}^2$, p primo.
Escribir $p-1 = 2^e n$ con n impar.
Calculamos b tal que $\left(\frac{b}{p}\right) = -1$ (un no residuo cuadrático).
Iniciamos con $y := a$, $r \equiv a^{\frac{n+1}{2}} \bmod p$, $SOL = 0$.
Mientras que $SOL = 0$, hacer

Calculamos k mínimo con $y^{2^k n} \equiv 1 \bmod p$.

Si $k = 0$ entonces $SOL = 1$.

En caso contrario: $y := yb^{2^{e-k}} \bmod p$, $r := r \cdot \left(b^{2^{e-k-1}}\right)^{-1} \bmod p$.

Devolvemos r.

Señalemos que en el algoritmo anterior es necesario buscar un entero b que no sea residuo cuadrático módulo p. Estrictamente no se conocen algoritmos polinómicos que calculen un tal número (se puede ver que existen si la hipótesis de Riemann es cierta). Sin embargo, como la mitad de los enteros positivos no nulos menores que p no son residuos cuadráticos, se puede diseñar un algoritmo probabilístico tipo las Vegas que encuentre uno en un tiempo esperado polinómico en $\log p$.

El método anterior se puede extender (con alguna dificultad añadida) al caso en que n es potencia de un primo. Si conocemos la factorización de n se combinan las soluciones módulo potencias de primos mediante el teorema chino para conseguir todas las soluciones módulo n.

En el caso particular en que $p \equiv 3 \bmod 4$ el cálculo de las raíces cuadradas de a es particularmente sencillo, ya que $p+1$ es múltiplo de 4 y por lo tanto $\pm a^{\frac{p+1}{4}}$ son las dos raíces cuadradas de a módulo p. Notemos que en este caso $\left(\frac{-1}{p}\right) = -1$ por lo que las dos raíces tienen distinto símbolo de Legendre, es decir, una es un cuadrado módulo p y la otra no.

Sistema de Rabin-Williams. Basado en este problema Rabin propuso un sistema criptográfico de clave pública en 1979 que utiliza un número n producto de dos primos grandes p y q tales que $p \equiv q \equiv 3 \bmod 4$. El entero n es la clave pública y los primos p y q constituyen la clave privada. Dado un mensaje m, $1 < m < n$, el cifrado de m es $c \equiv m^2 \bmod n$.

Para descifrar el mensaje c se calculan primero las soluciones de $Z^2 \equiv c \bmod p$, $\pm x$, y de $Z^2 \equiv c \bmod q$, $\pm y$. Como consecuencia

$$\pm q^{p-1} x \pm p^{q-1} y \bmod n$$

son las cuatro raíces cuadradas módulo n de c. Por tanto una de ellas es el mensaje m.

Una de las razones por las que este sistema no es demasiado útil es porque, en general, no hay forma de distinguir cual de las cuatro soluciones es el mensaje m. Si corresponde a un texto, posiblemente sea la única raíz que nos proporcione un texto inteligible, pero esto no es una vía demasiado satisfactoria.

Para eliminar este inconveniente Williams propuso una mejora que consiste en añadir dos bits al mensaje cifrado que permitan identificar sin ambigüedad cual de las 4 raíces es la correcta. Para ello tomamos también un entero s tal que $\left(\frac{s}{n}\right) = -1$ (en particular, s no es un residuo cuadrático módulo n) y usar como clave pública (n, s). La clave privada es en este caso $d = (\phi(n) + 4)/8$. El cifrado del mensaje m es la terna (C, b, e) definida de la siguiente forma:

$\left(\frac{m}{n}\right) = (-1)^b$.

$C \equiv (ms^b)^2 \bmod n$.

$e \equiv ms^b \bmod 2$.

Para recuperar el mensaje en claro se procede de la siguiente manera a partir de (C, b, e):

$m' := C^d \bmod n$.

$e' := e + m' \bmod 2$

$m := ((-1)^{e'} m')/(s^b) \bmod n$.

Otros sistemas de clave pública. Aunque el RSA, el método de las mochilas y el logaritmo discreto (sobre un cuerpo finito), son los tres sistemas más conocidos de clave pública, otros muchos han sido propuestos, basados en técnicas de curvas elípticas, códigos correctores, cuerpos de números, grupos, conjuntos complementarios, cifrado probabilístico, física cuántica, etc.

4.. Análisis del sistema RSA

Factorizacion. Ataques al RSA. Primalidad. Claves seguras y complejidad. Logaritmo discreto.

En este tema estudiaremos básicamente la solidez del sistema criptográfico RSA, es decir terminaremos el análisis, comenzado en el tema anterior, de las condiciones de Diffie y Hellman para dicho sistema en particular. Puesto que el sistema reposa en el problema de factorización comenzaremos por revisar algunos de los algoritmos de factorización más conocidos como medio de establecer una certidumbre "razonable" de la dificultad del problema. Posteriormente mencionaremos algunos ataques (además de la factorización) para el sistema RSA y a partir de ellos estableceremos unas condiciones mínimas de seguridad (sobre los números primos p y q) que se deben cumplir para que el método sea seguro.

El análisis anterior nos llevará a comprobar que las condiciones de Diffie y Hellman relativas a la generación de las claves se satisfacen. Por lo tanto se estudiará la complejidad de la generación de primos seguros, problema intrínsecamente unido al problema de primalidad (determinar si un número es o no primo).

Finalmente describiremos alguno de los métodos de cálculo del logaritmo discreto, aunque no analizaremos exhaustivamente los sistemas criptográficos basados en él.

Factorización

El ataque conceptualmente más sencillo al sistema RSA consiste en intentar factorizar en número n. Por tanto primero analizaremos algunos métodos sencillos de factorización con objeto de evitarlos en la elección de los números primos p y q.

El primero de ellos es sencillo de describir, consiste en hacer divisiones sucesivas bien por números primos pequeños bien por números primos "cercanos" a $\sqrt{n}$. Nótese que no es necesario conocer la primalidad de cada uno de los números cercanos a $\sqrt{n}$, basta calcular el máximo común divisor del candidato y de n. Como consecuencia una primera precaución es que los primos p y q deben ser ***grandes*** y no deben ser ***cercanos***.

Método de Fermat. Un método más sofisticado se debe a Fermat. Supongamos que tenemos dos enteros x e y de tal forma que $x^2 - y^2 = n$. En este caso $n = (x+y)(x-y)$ y, salvo que $x - y = 1$, tenemos una factorización de n. Por tanto podemos proceder como sigue:

Algoritmo. Fermat
Iniciamos con $x := [\sqrt{n}] + 1$.

> Si $x^2 - n$ es un cuadrado perfecto y^2 entonces devolvemos $x + y$, $x - y$.
>
> En otro caso $x := x + 1$.

El algoritmo anterior es eficiente si p y q son dos primos cercanos, y como consecuencia cercanos a $\sqrt{n}$. Por tanto es conveniente que p y q no sean de la misma longitud. Si tomamos esta precaución el método es impracticable debido al alto número de iteraciones que precisaría. En la práctica se debe usar limitando el número de iteraciones para evitar parar los procesos por medios más drásticos.

Cribas. La antigua idea de utilizat cribas, al modo de la criba de Eratóstenes, para cuestiones de factorización se puede utilizar en el contexto actual, produciendo mejores rendimientos en los algoritmos. En [CP01] se describen varias formas eficaces en el uso de las mismas. En nuestro caso, veamos como se puede utilizar en el método de Fermat que acabamos de describir.

Si queremos comprobar si $x^2 - n$ es o no un cuadrado, una forma de eliminar una cantidad considerable de casos es primero reducir módulo un entero (que debe ser pequeño) m y comprobar si $x^2 - n$ es un cuadrado módulo m o no lo es. Se supone que se ha computado previamente una tabla con los residuos cuadráticos

módulo m, con lo que esta última comprobación es rápida. El nombre de criba viene de éste hecho: cribamos los x posibles previamente a comprobar si son o no un cuadrado. Por otro lado la reducción es considerable, si tomamos $p_1, \ldots, p_r$ primos y cribamos sucesivamente mediante ellos, cada una de las etapas disminuye a la mitad aproximadamente los números x que se deben comprobar. Por tanto para comprobar una lista que a priori consta de M enteros x sólo debemos hacer la comprobación final con $M/2^r$ de ellos.

Podemos programar el algoritmo anterior de la siguiente forma:

Algoritmo. Fermat con cribas
Entrada: n (entero para factorizar), B, A.
Calculamos los primos $p_1, \ldots, p_r < B$.
Para $i = 1, \ldots, r$ hacemos

$C[i]$ el conjunto de cuadrados módulo p_i

$C[i] := (C[i] + n \bmod p_i)$.

Iniciamos con $x = [\sqrt{n}] + 1$
Para x, mientras $x < A + \sqrt{n}$ hacer

Para $i = 1, \ldots, r$ hacer

Si $x^2 \bmod p_i \notin C[i]$ entonces $x := x + 1$.

$y = [\sqrt{x^2 - n}]$

Si $y^2 = x^2 - n$ entonces $x - y$ es un factor de n

Método ρ de Pollard. Se trata de un método probabilístico, tipo Montecarlo, la base del mismo es el conocido como problema del cumpleaños.

Partimos de una función $f : \mathbb{Z}_n \to \mathbb{Z}_n$ y de un valor inicial $x_0 \in \mathbb{Z}^n$. Construimos la sucesión $x_1, \ldots, x_r, \ldots$ mediante $x_{i+1} := f(x_i)$, para $i \geq 0$. Supongamos que la sucesión construida semeja una sucesión aleatoria de elementos de $\mathbb{Z}_n$. Sean $p < q$ los primos elegidos (que son desconocidos para nosotros). Forzosamente existen t y ℓ de manera que $x_t \equiv x_{t+\ell} \bmod p$ (decimos que es una *colisión* módulo p), tomamos $t + \ell$ el menor posible. Como q es un primo diferente de p probablemente tendremos que $x_t \not\equiv x_{t+\ell} \bmod q$ y en este caso podemos concluir que $\mathsf{mcd}\,(x_{t+\ell} - x_t, n)$ es un factor no trivial de n. Es un ejercicio de cálculo demostrar que el número de iteraciones esperado para encontrar la primera colisión módulo p es del orden de $\mathcal{O}(\sqrt{p})$.

El método anterior requiere, además de calcular $x_i = f(x_{i-1})$, calcular el $\mathsf{mcd}\,(x_i - x_j, n)$ para todo $j < i$. Afortunadamente este último paso se puede mejorar mediante un sencillo método (llamado truco de Floyd) que consiste en construir otra sucesión a partir de la misma función pero más "deprisa", habitualmente se toma $y_1, \ldots, y_r, \ldots$ construida como $y_0 = x_0$, $y_i := f(f(y_{i-1}))$, y se buscan solamente las colisiones de parejas (x_i, y_i). Es sencillo demostrar que el número de iteraciones necesarias para conseguir una colisión (x_i, y_i) no es mucho mayor que el número de iteraciones usando el primer proceso.

Finalmente como función 'supuestamente aleatoria' para recorrer $\mathbb{Z}_n$ se toma habitualmente $f(x) = x^2 + 1 \bmod n$ (nunca una función lineal) y se comienza en un elemento x_0 elegido al azar en $\mathbb{Z}_n$. El tiempo esperado para encontrar el factor primo más pequeño es del orden de $\mathcal{O}(\sqrt{p}\log^2(n)\log\log(n))$ y para factorizar completamente n del orden de $\mathcal{O}(\sqrt[4]{n})$.

El siguiente es un programa en Maple que realiza el método anterior y que admite como segundo parámetro el número de iteraciones:

Programa: Algoritmo ρ de Pollard.

```
> pollard:=proc(n)
> local x,y,d,i,k;
> readlib(randomize)();
> x:=rand(n)(); y:=x;
> if nargs=2 then k:=args[2]; else k:=10000; fi;
> for i from 1 to k do
>           x:=x&^2+1 mod n; y:=(y&^2+1)&^2+1 mod n;
>           d:=igcd(x-y,n);
>           if d>1 and d<n then print(i); RETURN(d);
>                elif d=n then RETURN(‘fallo‘); fi;
> od;
> RETURN(‘No se encontro‘);
> end:
```

El método de Pollard es muy rápido y seguro cuando hay algún primo no demasiado grande. En general, si p es el factor primo más pequeño de un número n, el tiempo promedio estimado para encontrar el factor p es del orden de $1{,}3\sqrt{p}$. Como ejemplo indiquemos que el algoritmo anterior empleó (en un Pentium IV a 1.6GHz con 256 MB de memoria RAM) 26.13 segundos de CPU en efectuar 8476 iteraciones que proporcionaron el factor primo 100,000,891 de un número de 97 cifras. Aunque parece rápido, unas sencillas cuentas muestran que si el factor primo mas pequeño de n tiene 50 cifras decimales, el tiempo esperado es de más de $2 * 10^{16}$ años!!! (en la misma máquina). El método $\rho - 1$ de Pollard fue empleado en 1979 por Penk para encontrar el factor primo más pequeño del número de Mersenne $2^{257} - 1$ y en 1981 por Brent y Pollard para encontrar el factor primo más pequeño de los números de Fermat $F_n = 2^{2^n} + 1$ para $5 \leq n \leq 11$ (para $n = 8$ es $1, 238, 926, 361, 552, 897$).

Métod $p - 1$ de Pollard*. El método $p - 1$ de Pollard es uno de los muchos que utilizan para su ejecución la construcción de una base de factores (en la siguiente sección se analizarán algunos métodos basados en bases de factores). Fijemos $C \in \mathbb{R}$, $C > 0$ y tomemos el conjunto B formado por todos los primos menores o iguales que C. Se dice que un primo p es ***suave*** respecto de B si todos los factores primos de $p - 1$ están en B.

Supongamos que n es compuesto y que uno de sus factores primos p satisface la condición de que si ℓ^e divide a $p-1$ con ℓ primo entonces $\ell^e \leq C$. Sea b el mínimo común múltiplo de los enteros menores que C. Nótese que se tiene que ℓ^e divide a b y también $p-1$ divide a b. Si tomamos $a \in \mathbb{Z}_n$, por el pequeño teorema de Fermat, se tiene que $a^b \equiv 1 \bmod p$ y por lo tanto $d := \mathsf{mcd}\,(a^k - 1, n)$ es un factor propio de n.

El siguiente programa es una forma de materializar el algoritmo $p-1$ de Pollard, funciona razonablemente bien para enteros $C \leq 1000$.

Programa: Algoritmo $p-1$ de Pollard.

```
> pollard1:=proc(n)
> local a,g,q,l,i,b,C;
> a:=rand(n)(); g:=igcd(a,n);
> if g>1 then RETURN(g); fi;
> if nargs=2 then C:=args[2]; else C:=1000; fi;
> for i from 1 to B do
>    q:=ithprime(i);
>    l:=floor(evalf(ln(n)/ln(q)));
>    a:=a&^(q^l) mod n;
>    g:=igcd(a-1,n);
>    if g=n then RETURN(‘fallo‘);
>    elif g<>1 then  RETURN(g); fi;
> od;
> RETURN(‘No se encontro‘);
> end:
```

El algoritmo de Pollard factoriza en un tiempo razonable si alguno de los primos p ó q es suave respecto a C, por lo tanto si $p-1$ ó $q-1$ tienen todos sus factores primos "pequeños". Como consecuencia, si queremos que los primos p y q sean "seguros" debe ocurrir que $p-1$ y $q-1$ no tengan todos sus factores primos pequeños. Una forma posible es que sean del tipo $p = ap' + 1$ y $q = bq' + 1$ para primos grandes p' y q'.

Los métodos que hemos descrito son todos ellos de complejidad exponencial. Hay métodos más eficientes aunque, por supuesto, no se conoce ninguno de complejidad polinómica. A continuación describiremos alguno de los métodos de complejidad subexponencial.

Métodos de bases de factores*. Se trata de una generalización del método de Fermat, por tanto su base es que si encontramos x e y tales que $x^2 \equiv y^2 \bmod n$ pero $x \not\equiv \pm y \bmod n$ entonces $\mathsf{mcd}\,(x-y, n)$ es un factor de n.

Llamaremos base de factores a un conjunto $B = \{p_1, p_2, \ldots, p_r\}$ de manera que $p_1 = -1$ y $p_2, \ldots, p_r$ son primos distintos. Dado un entero z denotaremos por $[z]$ al único representante de la clase de z módulo n con valor absoluto menor que $n/2$. Diremos que un entero b es un B-número para n si $[b^2]$ es un producto de elementos de B. Si este es el

caso, podemos escribir $b = \prod_{i=1}^{r} p_i^{n_i}$ y denotamos por $e(b) = (n_1, \ldots, n_r) \bmod 2 \in (\mathbb{Z}_2)^r$.

Supongamos que tenemos B-números $b_1, \ldots, b_m$ tales que $e(b_1) + \cdots + e(b_m) = 0$ en $(\mathbb{Z}_2)^r$. Entonces, tomando $a_i = [b_i^2] = \prod_{i=j}^{r} p_j^{n_{ij}}$ $(1 \leq i \leq m)$, tendremos que

$$\prod_{i=1}^{m} a_i = \prod_{j=1}^{r} p_j^{\sum n_{ij}} = \prod_{j=1}^{r} p_j^{2\gamma_j} = \left(\prod_{j=1}^{r} p_j^{\gamma_j} \right)^2 .$$

Por tanto, si tomamos $b = [\prod_i b_i]$, $c = [\prod_j p_j^{\gamma_j}]$, obtenemos que $b^2 \equiv c^2 \bmod n$. Si $b \equiv \pm c \bmod n$ debemos repetir con una nueva base de factores B. Obsérvese que si n tiene $s \geq 2$ divisores primos distintos, el teorema Chino implica que cualquier cuadrado de $\mathbb{Z}_n$ tiene 2^s raices cuadradas en $\mathbb{Z}_n$, por lo tanto la probabilidad de que b y c no sirvan es $2/2^s \leq 1/2$.

No hemos indicado ningún método todavía para la elección de la base B ni de los enteros $b_1, \ldots, b_m$. Notemos en primer lugar que si B está fija, es suficiente con tomar $m = r+1$. Una forma de hacerlo es fijar como B el conjunto de los $r-1$ primeros números primos junto con -1. Después la elección de los $b_1, \ldots, b_m$ se puede hacer al azar comprobando que son B-números. Si hacemos la elección para valores cercanos a $\sqrt{kn}$, con diferentes $k = 1, 2, \ldots$, los representantes $[b_i^2]$ son "pequeños" y probablemente el algoritmo terminará pronto. Otra opción es elegir primero los enteros $b_1, \ldots, b_m$ con el criterio anterior y elegir la base de factores como los primos que aparecen en su descomposición.

El método anterior, con elecciones aleatorias de $b_1, \ldots, b_m$ y una 'razonable' estimación de B, fué desarrollado por Dixon en 1981 (*random-squares method*) y es todavía el método probabilístico con una complejidad probada rigurosamente más baja. Se puede ver una descripción completa del mismo en [vzGG99].

FRACCIONES CONTINUAS*. Veremos un método más eficaz para la elección de la base B que es la base del método de factorización llamado de las fracciones continuas. Dados $z_0, \ldots, z_n \in \mathbb{Z}$, denotamos por $[z_0, \ldots, z_n]$ el número racional definido por

$$[z_0, \ldots, z_n] = z_0 + \cfrac{1}{z_1 + \cfrac{1}{z_2 + \cfrac{1}{\cdots \cfrac{1}{z_n}}}}$$

Dado un número real x, definimos $a_0 = [x]$ y supuesto que hemos definido $a_0, \ldots, a_i$, entonces $a_{i+1} = [1/x - [a_0, \ldots, a_i]]$. La sucesión $a_0, a_1, a_2, \ldots$ se llama la fracción continua de x, y x se suele representar mediante

$$x = [a_0, a_1, a_2, \ldots] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}}$$

Nótese que x es racional si y sólo si la fracción continua es finita. El número racional

$$\frac{m_i}{n_i} = [a_0, \ldots, a_i]$$

se llama el i-ésimo convergente de x. Se tiene que x es el límite de la sucesión de racionales m_i/n_i y de hecho proporcionan una aproximación óptima a x en el sentido de que los denominadores son mínimos. El numerador y denominador se pueden obtener recursivamente por las fórmulas:

$$m_i = a_i m_{i-1} + m_{i-2} \qquad n_i = a_i n_{i-1} + n_{i-2}\ .$$

El resultado clave que permitirá su uso en la construcción de bases de factores es el siguiente. Tomamos m_i/n_i el i-ésimo convergente de $\sqrt{n}$. Entonces $|[m_i^2]| < 2\sqrt{n}$. Este hecho garantiza que los numeradores de los convergentes de $\sqrt{n}$ crecen, módulo n, moderadamente y podemos usarlos para calcular la sucesión $b_1, \ldots, b_m$:

Algoritmo.
Entrada: un entero impar n, una cota C.
Iniciamos con: $b_{-1} = -1$, $b_0 = a_0 = [\sqrt{n}]$ y $x_0 = \sqrt{n} - a_0$.
Para $i = 1, \ldots,$ hacemos

$a_i = [1/x_{i-1}]$, $x_i = (1/x_{i-1}) - a_i$.

$b_i = a_i b_{i-1} + b_{i-2} \bmod n$.

Calculamos $[b_i^2]$ y descartamos b_i si $[b_i^2]$ tiene factores primos mayores que C

Elegimos como B los factores primos de los $b_1, \ldots, b_m$ que aparecen más de una vez.
Nos quedamos con los $b_1, \ldots, b_m$ que son B-números.

Basado en el método anterior, Morrison y Brillhart desarrollaron en 1970 el que se conoce como método de las fracciones continuas ($CFRAC$) que les permitió factorizar el séptimo número de Fermat $F_7 = 2^{2^7} + 1$. Es el método usado por Maple por defecto.

Para indicar la complejidad de estos algoritmos es cómodo introducir la siguiente notación. Denotamos

$$L_n(\gamma, c) = \mathcal{O}\left(e^{c(\ln n)^{\gamma}(\ln\ln n)^{1-\gamma}}\right).$$

Diremos que un algoritmo es es un $L(\gamma)$-algoritmo si aplicado a un entero n su complejidad temporal es del orden de $L_n(\gamma, c)$ para alguna constante c. Nótese que un algoritmo polinómico es un $L(0)$-algoritmo y uno exponencial es un $L(1)$-algoritmo. Diremos que un algoritmo es subexponencial si es un $L(\gamma)$-algoritmo para algún γ con $\gamma < 1$.

En estos términos la complejidad del método de las fracciones continuas se estima (no ha sido probado rigurosamente) en $L(1/2, \sqrt{2} + \epsilon)$.

Cribas cuadráticas*. Como ya comentamos antes, la idea de usar cribas permite mejorar la eficacia de algunos algoritmos. El método de las cribas cuadráticas (conocido abreviadamente como QS: quadratic sieve) fué introducido por Pomerance en 1982 sobre la base de usar bases de factores y un método de criba. Dicho método fué el más eficaz hasta comienzos de los 90, sustituyendo al método de las fracciones continuas. Permitió factorizar en un tiempo razonable enteros de unas 120 cifras, aumentando las exigencias de los primos a elegir en el RSA. Su complejidad (también conjetural) es del orden de $L(1/2, 1)$.

La idea del método de las cribas cuadráticas es la siguiente. Partimos de un entero n, compuesto y que sabemos que no es potencia de un primo. Fijamos un entero b (el candidato natural para b es $[L_n(1/2,1)]$ debido a que es el valor promedio esperado hasta encontrar un factor no trivial) y una base $B = \{p_1, \ldots, p_k\}$ formada por los primos $p \leq b$ tales que $\left(\frac{n}{p}\right) = 1$. Tomamos además $\pm a_i$ tales que $a_i^2 \equiv n \bmod p_i$.

Nuestro objetivo es encontrar $x_1, \ldots, x_r$, con $r \leq k+1$, $[\sqrt{n}] < x_i < [\sqrt{n}] + b$, tales que $x_i^2 - n = \prod_{j=1}^{k} p_j^{e_j^i}$ descompone totalmente en B y $e(x_1) + \cdots + e(x_r) = 0 \in \mathbb{Z}_2^k$ donde $e(x_i) = (e_1^i, \ldots, e_k^i)$ es el vector de exponentes de la factorización de $x_i^2 - n$ en la base B. En este caso, como ocurría en la discusión que hicimos sobre las bases de factores, obtenemos que $(x_1^2-n)(x_2^2-n)\cdots(x_r^2-n)$ es un cuadrado, y^2, módulo n (su factorización se puede encontrar a partir de la factorización de cada uno de los factores, ya calculada). Por tanto tomando $x = x_1 x_2 \cdots x_r$ obtenemos que $d := \mathsf{mcd}\,(x-y, n)$ es un factor de n.

El método de cribado aparece precisamente a la hora de calcular los enteros $x_1, \ldots, x_r$. Para ello se toma la lista de pares (x, x^2-n) con $x \geq [\sqrt{n}]$ y para cada $i = 1, \ldots, k$ se eliminan de la lista todos aquellos x que no aparecen en la forma $a_i + kp_i$ o $-a_i + kp_i$ para algún $k \geq 1$. El método de cribado se puede refinar (ver [CP01]) para detectar las potencias de p_i que aparecen en la factorización de $x^2 - n$.

Otros métodos de factorización. El sucesor al método de cribas cuadráticas (en cuanto a eficacia) fue el método de las cribas de cuerpos de números (NFS: number field sieves) desarrollado, a partir de ideas de Pollard, por varios autores (Lenstra, Lenstra Jr., Manasse, Odlyzko y Pomerance entre ellos). Con dicho método, que sigue siendo uno de los más eficaces, se factorizó en 1990 el noveno número de Fermat, $2^{2^9}+1$. La complejidad (estimada, no rigurosamente probada) del NFS es del orden de $L_n(1/3,1) = \mathcal{O}(exp(c\sqrt[3]{\ln n}\sqrt[3]{(\ln\ln n)^2}))$.

Además de los mencionados merecen ser destacados los métodos de factorización basados en la estructura de grupo de las curvas elípticas sobre cuerpos finitos (Lenstra). Su complejidad es del orden de $L_n(1/2, \sqrt{2}+\mathcal{O}(1))$ (probada rigurosamente bajo ciertas condiciones 'razonables').

Otros ataques al RSA

La elección de la clave e. En un sistema RSA siempre hay algunos mensajes que no se deben cifrar, son aquellos M tales que $M \equiv M^e \bmod n$. Esta condición es equivalente (teorema Chino de los restos) a que $M \equiv M^e \bmod p$ y $M \equiv M^e \bmod q$. Ahora, las soluciones de la ecuación $X \equiv X^e \bmod p$ son exactamente $1 + \mathsf{mcd}\,(e-1, p-1)$ y por lo tanto el número de mensajes M que coinciden con su cifrado son

$$(1 + \mathsf{mcd}\,(e-1, p-1))(1 + \mathsf{mcd}\,(e-1, q-1))\ .$$

Puesto que $e-1$, $p-1$ y $q-1$ son pares, al menos hay 9 mensajes con esta condición. Por supuesto, se pueden calcular todos los mensajes no cifrables para evitarlos, sin

embargo normalmente lo que se hace es elegir e de manera que dicho número sea el menor posible y, en este caso, la probabilidad de que nos encontremos con uno de ellos es practicamente cero.

Así pues lo mejor es tomar e con las condiciones:

1. e es inversible módulo $\phi(n)$.
2. $\mathsf{mcd}\,(e-1, p-1) = \mathsf{mcd}\,(e-1, p-1) = 2$.

Cálculo de la clave de descifrado. Un posible ataque es intentar calcular la clave de descifrado d a partir de (n, e). Además del ataque basado en la factorización (o equivalentemente del cálculo de $\phi(n)$), conviene asegurarse de que no hay demasiados enteros r tales que $M^{er} \equiv M \mod n$ o equivalentemente que $M^{er-1} \equiv 1 \mod n$. Esta condición módulo p y q se traduce en que $er-1$ debe ser múltiplo de $p-1$ y $q-1$, por lo que lo es del mínimo común múltiplo, γ, de ambos. Si elegimos d_0 el inverso multiplicativo de e módulo γ, el conjunto de las posibles claves privadas es $\{d_i = d_0 + i\gamma\}$ para $0 \leq i \leq [(n-d_0)/\gamma]$. Como consecuencia si γ es el mayor posible, el número de claves privadas es el menor. Así pues, como ya ocurría en casos anteriores, lo mejor es elegir $p = 2p'+1$, $q = 2q'+1$ con p' y q' primos grandes. De esta forma $\gamma = 2p'q'$ y, dependiendo del tamaño de d_0, se tendrá que existen un máximo de dos claves privadas posibles.

Otros ataques. Un posible ataque, llamado ataque cíclico, se basa en iterar el cifrado (que es público) hasta conseguir el mensaje original M de nuevo. Es decir, tomar $M_k \equiv M_{k-1}^e \mod n$, $(M_0 = M)$ para $k \geq 1$. Es claro que si M es un mensaje arbitrario y tenemos que $M^{e^k} \equiv M \mod n$ obtenemos una clave de descifrado: e^{k-1}, problema que ya discutimos antes. Si se busca solamente descifrar el mensaje M, las condiciones que ya han aparecido sobre p y q garantizan que no hay demasiados mensajes con números k accesibles.

De la misma naturaleza, aunque un poco más delicado, es el ataque basado en el problema del cumpleaños. Se trata en este caso de averiguar la clave privada a partir de un mensaje (que puede ser elegido por el atacante) M buscando la coincidencia entre dos potencias del mismo. Es decir, se calculan $i > j$ tales que

$$M^i \equiv M^j \mod n\,.$$

Es sencillo comprobar que si se elige un mensaje M suficientemente general el cálculo de una coincidencia como la anterior permite calcular una clave privada. El cálculo de la probabilidades nos dice que la probabilidad de encontrar una coincidencia despues de r pruebas es $\pi \simeq 1 - (e^{3c^2})$, siendo $c \simeq r/\sqrt{\gamma}$. Por tanto encontramos de nuevo que es deseable que γ sea lo mayor posible, es decir, que el máximo común divisor de $p-1$ y $q-1$ sea 2.

Primalidad

Una condición imprescindible para que el cálculo de las claves del sistema RSA sea computacionalmente sencillo es contar con criterios eficaces de primalidad. Por supuesto, y a la vista de la sección anterior, debemos olvidarnos de los métodos basados en encontrar una factorización del candidato.

El hecho crucial que hace del sistema RSA un sistema eficiente de clave pública es precisamente que el problema de saber si un número es o no primo es mucho más sencillo que encontrar una factorización del mismo. De hecho recientemente Agrawal, Kayal y Saxena han probado que la primalidad es un problema de complejidad polinómica (véase [AKS02]), exhibiendo un algoritmo determinista de complejidad $\mathcal{O}(\log^{12} n)$. Bajo ciertas condiciones, que son ciertas para valores muy altos del número n y que se conjeturan ciertas en general, la complejidad desciende hasta $\mathcal{O}(\log^6 n)$. En la práctica criptográfica dicho resultado no es demasiado útil, ya que (como veremos a continuación) existen algoritmos probabilísticos muy rápidos que certifican la primalidad de un número entero con probabilidad tan alta como se desee, pero resuelve una de las conjeturas más imporantes abiertas en este campo, de hecho Manindra Agrawal ha recibido el premio del Instituto Clay en Octubre de 2002 por este resultado (los ganadores anteriores han sido Wiles, Lafforgue, Connes y Smirnov).

Describiremos a continuación algunos de los algoritmos de primalidad más habituales.

Test de Lucas. El pequeño teorema de Fermat es la base para un primer criterio de primalidad determinista, aunque poco efectivo. Sabemos que si p es un número primo entonces $a^{p-1} \equiv 1 \bmod n$ para todo entero a primo con p. Si un entero n satisface la condición $a^{n-1} \equiv 1 \bmod n$ para un cierto entero a se dice que n es pseudo-primo respecto de a. Sin embargo existen números compuestos n que son pseudo-primos respecto de cualquier base a con $\mathsf{mcd}\,(n, a) = 1$. Dichos número se llaman números de Carmichael, todos ellos son libres de cuadrados y dentro de dicha clase se caracterizan porque para cada factor primo p de n se tiene que $p - 1$ divide a $n - 1$. En 1992 Alfords, Granville y Pomerance probaron la existencia de infinitos números de Carmichael.

Un hecho curioso es que los números de Carmichael se pueden utilizar en un sistema RSA sin mayor problema, excepto por un pequeño detalle, son muy sencillos de factorizar como veremos después.

El criterio de primalidad de Lucas dice simplemente que un número n es primo si y sólo si $(\mathbb{Z}/n\mathbb{Z})^*$ es un grupo cíclico de orden $n - 1$. La efectividad de dicho criterio es escasa, por cuanto deberíamos buscar un entero a, primo con n y de manera que $a^{n-1} \equiv 1 \bmod n$ pero $a^m \not\equiv 1 \bmod n$ para todo $m < n - 1$. Es suficiente demostrar que $a^{\frac{n-1}{p}} \not\equiv 1 \bmod n$ para todo primo p divisor de $n - 1$, pero esta reducción requiere conocer la factorización de $n - 1$, por lo que solo es útil cuando n tiene alguna forma

especial. Aún en este caso, por ejemplo para los números de Fermat $F_n = 2^{2^n} + 1$, para probar que F_n es compuesto deberiamos, en principio, probar con todos los posibles números a.

Test de Solovay-Strassen*. Si n es primo y a es un entero primo con n, se tiene que $a^{(n-1)/2} \equiv 1$ si a es un cuadrado módulo n y $a^{(n-1)/2} \equiv -1$ si a no lo es. Por tanto $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right)$, siendo $\left(\frac{a}{n}\right)$ el símbolo de Legendre de a módulo n. A diferencia de lo que ocurría con el criterio de Fermat, en este caso se tiene que n es primo si y sólo si $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right)$ para todo a primo con n. Este criterio recibe el nombre de criterio de Euler y, dado que sería necesaria la comprobación para todas las bases a con $\mathsf{mcd}\,(a, n) = 1$, tampoco proporciona un test eficiente.

Sin embargo en este caso podemos demostrar que si n es un número compuesto impar al menos el 50 % de las bases a no cumpliría la condición $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right)$ (para n compuesto $\left(\frac{a}{n}\right)$ es el símbolo de Jacobi, que generaliza el símbolo de Legendre). Por tanto, si elegimos k bases a de forma aleatoria con la condición $\mathsf{mcd}\,(a, n) = 1$ y testeamos la condición $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right)$ para cada una de ellas tendremos que:

1. Si alguno de los test falla entonces con seguridad n es compuesto.
2. Si n supera los k tests entonces la probabilidad de que n sea compuesto es a lo sumo de $1/2^k$.

Disponemos por tanto de un test probabilistico realmente eficiente ya que, usando la exponenciación modular, su complejidad es del orden de $\mathcal{O}(k \log n M(n))$.

Para implementar el test de Solovay-Strassen es necesario que el símbolo de Jacobi, que extiende el símbolo de Legendre al caso en que la base no es un primo, se pueda calcular de manera eficiente. El cálculo del símbolo de Jacobi se puede hacer usando las siguientes propiedades (en todas ellas, k es un entero impar positivo y n, m enteros primos con k):

1. $\left(\frac{nm}{k}\right) = \left(\frac{n}{k}\right)\left(\frac{m}{k}\right)$.
2. $\left(\frac{n}{k}\right) = \left(\frac{n+ik}{k}\right)$ para todo entero i.
3. $\left(\frac{2}{k}\right) = (-1)^{\frac{(k^2-1)}{8}}$
4. *Reciprocidad:* Si también n es impar positivo se tiene

$$\left(\frac{n}{k}\right) = \left(\frac{k}{n}\right)(-1)^{\frac{(n-1)(k-1)}{4}} .$$

Test de Rabin-Miller. La idea subyacente al criterio de Euler (o si se prefiere al test de Solovay-Strassen) es que la ecuación $X^2 - 1$ tiene sólo las soluciones ± 1 módulo un primo. Se puede refinar el criterio de Euler en el siguiente sentido: supongamos que n es primo y pongamos $n - 1 = 2^e d$ con d impar. Sea a un entero. Entonces la sucesión (módulo n):

$$a^d, a^{2d}, a^{2^2 d}, \ldots, a^{2^e d}$$

tiene la peculiaridad de que:

- O bien está formada toda ella por unos.
- O bien existe j, con $0 \leq j < e$, de manera que $a^{2^j d} = -1$. (En este caso los siguientes son todos iguales a 1).

Para un número n impar arbitrario, decimos que n es pseudoprimo fuerte respecto de un entero a si para ellos se satisface la condición anterior. Se puede comprobar que si n es pseudoprimo fuerte respecto de a entonces es pseudoprimo de Euler respecto de la base a, es decir, $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right)$. Por tanto, también tendremos que n es pseudoprimo fuerte (por definición si lo es respecto de cualquier base a) si y sólo si es primo, con lo que disponemos de un nuevo test de primalidad llamado de Rabin-Miller. Como test determinista tampoco es un test eficiente, ya que sería necesario comprobarlo para todas las bases con lo que su complejidad sería del orden de $\mathcal{O}(n \log^3 n)$.

Si n es un entero impar compuesto se puede demostrar que el número de bases a para las que n es pseudoprimo fuerte es, a lo sumo, de $\phi(n)/4$. Por lo tanto, eligiendo k bases al azar, si n es un entero impar que pasa el test para todas ellas (es decir, si es pseudoprimo fuerte para todas ellas), n es compuesto con probabilidad menor que $1/4^k$.

El siguiente programa de Maple es una posible implementación del test de Rabin-Miller para un entero n y una base a (parámetros de entrada), la salida es 1 si n es pseudoprimo para a y 0 si no lo es:

Programa: Test de Rabin-Miller.

```
> rabmil1:=proc(n,a)
> local test, m,e,b,i;
> test:=0; m:=n-1; e:=0;
> while irem(m,2)=0 do m:=iquo(m,2); e:=e+1; od;
> b:=a&^m mod n;
> if b=1 or b=n-1 then  test:=1;
>    else for i to e-1 do
>        b:=b&^2 mod n;
>        if b=n-1 then test:=1; fi;
>        od;
> fi; test;
> end:
```

Tomando como base el anterior, el siguiente programa, dados enteros n y k, elige k bases al azar y proporciona como salida 0 (en cuyo caso n es compuesto) o 1 (en cuyo caso n es primo con probabilidad mayor que $1 - (1/4)^k$):

Programa: Test de Rabin-Miller.

```
> rabinmiller:=proc(n,k)
> local test,i,a;  test:=1;
> for i to k do a:=rand(n)();
>    if rabmil1(n,a)=0 then test:=0; i:=k;  fi;
> od; test;
> end:
```

Nuevos ataques al RSA. Un resultado importante es que si tenemos un entero n y una base a de manera que n es pseudo-primo respecto de a pero no es pseudo-primo fuerte respecto de a entonces se puede factorizar fácilmente. En efecto, en este caso encontramos un entero $j < e$ de manera que $b = a^{2^j d} \not\equiv \pm 1 \bmod n$ pero $b^2 \equiv 1 \bmod n$, es decir, una raíz cuadrada de 1 diferente de ± 1. Por tanto $(b-1)(b+1)$ es un múltiplo de n y el máximo común divisor de $b-1$ y n proporciona un factor no trivial de n. En particular, los números de Carmichael se pueden factorizar fácilmente, ya que no son pseudo-primos fuertes para la mayoría de las bases pero son pseudo-primos para todas ellas.

Como consecuencia, un riesgo a evitar en la elección de los primos p y q del sistema RSA es que sea sencillo encontrar una base a de manera que n es pseudo-primo respecto de a pero no pseudo-primo fuerte. Como la proporción entre ambos es de dos a uno, lo mejor es que n sea pseudo-primo respecto del menor número posible de bases. Analizando el número de soluciones de la ecuación $x^{n-1} \equiv 1 \bmod n$ se puede comprobar que si $(p-1)/2$ y $(q-1)/2$ son primos entre sí, entonces hay cuatro bases a de manera que n es pseudo-primo respecto de a y solo dos, $a = \pm 1$, de manera que n es pseudo-primo fuerte respecto de ellas. Por lo tanto en este caso el sistema es seguro respecto de este tipo de ataques. A veces un primo impar p con la condición de que $(p-1)/2$ es primo se llama un primo seguro (por ejemplo en Maple).

La elección de n. Una forma posible de usar un sistema criptográfico basado en el RSA es el siguiente. El sistema es el mismo que ya conocemos, pero ahora el entero n es común para todos los usuarios del mismo que, para evitar que los mensajes de uno sean accesibles a los demás, desconocen los factores primos p y q. Este sistema requiere por tanto la existencia de un administrador que distribuye las claves a todos los usuarios. En principio parece un sistema razonable en grupos de usuarios que habitualmente se comunican sólo con el administrador ya que facilita bastante los tiempos de computación del mismo. Sin embargo se puede comprobar que el sistema es muy frágil, cualquiera de los usuarios con una razonable capacidad de cálculo puede calcular la factorización de n y como consecuencia descifrar los mensajes que cualquiera de los usuarios dirija al administrador.

Supongamos que conocemos una pareja (e, d) de manera que $ed \equiv 1 \bmod \phi(n)$. Sabemos que $ed-1$ es un múltiplo de $\phi(n)$. Escribimos $ed - 1 = 2^h r$ con r impar y

consideramos, para un entero a primo con n, la sucesión

$$a^r, a^{2r}, a^{2^2 r}, \ldots, a^{2^h r} \mod n \ .$$

Un análisis de las distintas situaciones que pueden darse (ver por ejemplo [Chi95]) nos garantiza que al menos la mitad de las bases a nos proporcionan un entero $b = a^{2^j r}$, $j < h$, de manera que $b \not\equiv \pm 1 \mod n$ y $b^2 \equiv 1 \mod n$. Por lo tanto se puede factorizar n.

Hipótesis de Riemann Generalizada y primalidad: La función zeta de Riemann se define mediante la fórmula:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

para los números complejos s con $\Re(s) > 1$. Se prueba que es holomorfa en el semiplano $\Re(s) > 1$ y que se puede extender en una función meromorfa en el plano complejo $\mathbb{C}$. La hipótesis de Riemann es que en la banda $0 < \Re(s) < 1$ todos los ceros de ζ están en la recta $\Re(s) = 1/2$.

Dado un homomorfismo χ del grupo multiplicativo $(\mathbb{Z}/m\mathbb{Z})^*$ en $\mathbb{C}^*$, podemos extender χ a un homomorfismo del semigrupo multiplicativo $\chi : \mathbb{N} \to \mathbb{C}^*$ (χ se llama un carácter módulo m). A partir de χ se define su L-serie de Dirichlet mediante

$$L_\chi(s) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^s} \ .$$

La hipótesis de Riemann generalizada (HRG) dice que para todo módulo m y todo carácter χ, los ceros en la banda $0 < \Re(s) < 1$ de la función $L_\chi(s)$ están sobre la recta $\Re(s) = 1/2$.

En el caso de que la hipótesis de Riemann generalizada fuese cierta, se puede probar que existe una constante $c < 2$ de manera que n es primo si y sólo si es pseudoprimo fuerte para todas las bases a con $a < c \log^2 n$. Como consecuencia se tendría el siguiente sugestivo test determinista de primalidad efectivo (ya que su complejidad es $\mathcal{O}(\log^3 n)$):

Algoritmo. Miller

$w := \min\{[2\log^2 n], n-1\}$

Para $a = 2, \ldots, w$ hacer

Si n no es pseudo primo fuerte respecto de a devolver NO.

Devolver SI.

Si el algoritmo devuelve NO, entonces n es compuesto. Si devuelve SI entonces o n es primo o la HRG es falsa.

Otros algoritmos. Respecto de los tests de primalidad es conveniente indicar que existen muchos más que los descritos, aunque para los objetivos de este curso el test de Rabin-Miller es más que suficiente. En particular merece la pena mencionar la existencia de tests de primalidad específicos para cierto tipo de enteros. Por ejemplo para los números de Mersenne (los de la forma $2^p - 1$, p primo) existen algoritmos eficientes como el test de Lucas-Lehmer que es la versión computacional del siguiente enunciado:

Teorema. *Sea $M_p = 2^p - 1$, p primo impar. Sea $r_1 = 4$ y, para $k \geq 2$, sea $r_k = r_{k-1}^2 - 2 \bmod M_p$. Entonces M_p es primo si y sólo si $r_{p-1} \equiv 0 \bmod M_p$.*

Los números de Mersenne suelen proporcionar los primos "record" y se utilizan muy frecuentemente como pruebas de rapidez de cómputo de nuevos sistemas. En la página web: http://www.utm.edu/research/primes/ se puede encontrar información actualizada sobre los primos mayores conocidos y, en particular, sobre los primos de Mersenne conocidos. La página http://www.mersenne.org/ es la página "oficiosa" dedicada a los primoas de Mersenne.

En cuanto a los test de propósito general, los basados en curvas elípticas juegan un papel destacado. Golwasser y Killian en 1986 proporcionan el primero y más conocido, posteriormente fué mejorado por Atkin y Morain mediante el que se conoce hoy día como test ECPP (Elliptic curve primality proving).

La elección de los primos

Tanto los métodos de factorización como la vulnerabilidad a algunos ataques nos han proporcionado ya algunas condiciones sobre los primos p y q. Añadamos que también hay algoritmos de factorización relativamente eficientes si $p+1$ y $q+1$ tienen todos sus factores primos pequeños. Por lo tanto debemos elegir dos primos p y q, de manera que $p = ap' + 1$ y $q = bq' + 1$ para p' y q' primos grandes y de forma que $p+1$ y $q+1$ tengan al menos un primo grande. Hay también métodos de factorización que hacen que los primos sean vulnerables si para cada factor primo r de $p-1$ (o de $q-1$) $r-1$ tiene todos sus factores primos pequeños.

Como consecuencia definiremos ***primo fuerte*** como un número primo p que satisface las siguientes condiciones:

- $p - 1 = ar$, siendo r un primo grande y a un entero.
- $r - 1 = bs$, siendo s un primo grande y b un entero.
- $p + 1 = ct$, siendo t un primo grande y c un entero.

Diremos que la pareja de números primos (p, q) es una pareja de ***primos seguros*** si ambos son primos fuertes y además $(p-1)/2$ y $(q-1)/2$ son primos entre sí. La clave $[n, e]$ decimos que es una clave segura si n es producto de una pareja segura de primos y $\mathsf{mcd}\,(e-1, p-1) = \mathsf{mcd}\,(e-1, q-1) = 2$.

Primos fuertes. Un método para generar primos fuertes ha sido descrito por Gordon en [Gor84]. El siguiente algoritmo, basado en la descripción del método de Gordon hecha en [PS97], produce un número primo de longitud aproximada la misma que se pasa como parámetro.

Programa: Primos fuertes.

```
> primofuerte:=proc(k)
```

```
> local l,r,s,t,p0,p;
> l:=floor((k-2)/2);
> readlib(randomize)();
> rand(10^(l-1)..10^(l))(); s:=nextprime(%);
> rand(10^(l-2)..10^(l-1))(); t:=nextprime(%);
> r:=2*3*t+1; while isprime(r)=false do
>       r:=r+2*t; od;
> p0:=(2*s&^(r-2) mod r)*s-1;
> if irem(p0,2)=0 then p0:=p0+r*s; fi;
> p:=p0+2*2*r*s; while isprime(p)=false
>       do p:=p+2*r*s; od;
> end:
```

Es sencillo comprobar que el primo p de salida es congruente con 1 módulo el primo r y con -1 módulo el primo s. Además el primo r se ha construido de manera que es congruente con 1 módulo el primo t. Los primos s y t son dos primos elegidos al azar con aproximadamente la mitad de los dígitos que indica el parámetro de entrada. En el algoritmo se han utilizado funciones propias de Maple, como `nextprime`, `isprime`,.. cualquiera de ellas es fácil de sustituir por tests de Rabin-Miller con un margen de confianza prefijado (para la función `nextprime` se recorren sucesivamente a, $a+2, a+4, \ldots$, hasta dar con un primo probable). Las operaciones que involucra el procedimiento son de complejidad polinómica, ya conocida, por lo que el algoritmo será de complejidad polinómica siempre que los diferentes bucles se detengan en un número pequeño de pasos. Analizaremos este extremo un poco más adelante.

Claves seguras. Para generar ahora una clave segura solamente necesitamos buscar dos primos fuertes y la clave pública e. El siguiente programa tiene en cuenta los requisitos que se han ido detectando en cuanto a ellos.

Programa: Clave RSA.

```
> genclavesseguras:=proc(k)
> local p,q,n,phin,e,d,i,j,m;
> p:=primofuerte(k);
> if nargs=2 then m:=k+args[2]; else
>          readlib(randomize)(); m:=k+rand(5..10)(); fi;
> i:=2; while i<>1 do
>     q:=primofuerte(m);
>     i:=igcd((p-1)/2,(q-1)/2); od;
> n:=p*q;
```

```
> phin:=(p-1)*(q-1);
> i:=2; j:=3; while i<>1 or j<>2 do
> e:=rand():  i:=igcd(e,phin): j:=igcd(e-1,phin):  od;
> d:=1/e mod phin;
> [[n,e],[n,d,p,q]];
> end:
```

Mencionemos, como prueba puramente heurística, que el programa anterior en pruebas sistemáticas generaba una clave con primos de al menos 100 cifras cada uno en un tiempo que, en promedio, era de unos 5 segundos de CPU.

Complejidad. Una vez que disponemos de algoritmos eficientes para generar parejas de primos fuertes y claves seguras, solo nos resta comprobar que dicho método es computacionalmente eficiente. La eficacia del mismo reposa en dos teoremas, el primero de ellos es el teorema de densidad de números primos y el segundo es el teorema de Dirichlet sobre primos en progresiones aritméticas. En ambos casos la demostración, que utiliza métodos analíticos, queda fuera del alcance de este curso

Teorema de los números primos. Sea $x \in \mathbb{R}$, $x > 0$. Denotamos por $\pi(x)$ el cardinal del conjunto de números primos menores o iguales que x. El teorema de los números primos dice que

$$\pi(x) \simeq \frac{x}{\ln x} \, .$$

Se puede precisar un poco más, ya que se tiene que

$$\frac{x}{\ln x}\left(1+\frac{1}{2\ln x}\right) < \pi(x) < \frac{x}{\ln x}\left(1+\frac{3}{2\ln x}\right)$$

para $x \geq 59$. A partir de este resultado se puede estimar que la probablidad de que un número aleatorio comprendido entre B y $2B$ (para aproximar la longitud binaria que desamos) sea primo es mayor o igual que $1/2\ln B$. Como consecuencia, usando el k-test de primalidad de Rabin-Miller y elecciones aleatorias de números n con $B < n \leq 2B$, podemos afirmar que el algoritmo correspondiente devuelve un número n que es primo con probabilidad al menos de $1-(2\ln B)/4^k$. Además el número de operaciones esperadas de dicho algoritmo sera del orden de $\mathcal{O}(k(\log^2 B)M(\log B))$.

Teorema de Dirichlet. El teorema de Dirichlet mencionado establece que si a y d son enteros primos entre sí, entonces la progresión aritmética $a, a+d, a+2d, \ldots, a+kd, \ldots$ contiene infinitos números primos. Además se puede establecer su densidad, si $\pi(x,d,a)$ denota el cardinal del conjunto de primos de la sucesión que son menores que x se tiene que

$$\pi(x,d,a) \sim \frac{\pi(x)}{\phi(d)} \sim \frac{x}{\phi(d)\ln x} \, .$$

El resultado anterior garantiza que los algoritmos de búsqueda de números primos en progresiones de razón d relativamente pequeña son eficientes.

Algoritmos para el logaritmo discreto*

El problema del logaritmo discreto se considera actualmente más duro que la factorización. Hay tres tipos de algoritmos que se suelen utilizar para resolverlo, según el tipo de ideas:

1. Algoritmos válidos para grupos abelianos geneales. Entre ellos el más conocido es el algoritmo de Shanks (de tipo paso de enano paso de gigante). Además hay también un algoritmo semejante al método ρ de factorización. En general no explotan las condiciones particulares del grupo en el que trabajamos.
2. Algoritmos válidos en grupos abelianos finitos de orden n con la condición de que los factores primos de n son menores que una determinada cota (normalmente pequeña). El algoritmo de Silver-Pohling-Hellman es el prototipo en esta categoría.
3. Algoritmos que utilizan representaciones como producto de un cierto conjunto (no muy numeroso) de elementos. El algoritmo index-calculus es el más conocido de ellos.

Algoritmo de Shanks. Cacula el logaritmo discreto de y en la base a módulo n, es decir, el número entero x tal que $y = a^x \bmod n$.

Algoritmo.
Datos de entrada: y, x, n.
Calculamos $s := [\sqrt{n}]$.
Para $j = 0, \ldots, s-1$ hacemos $S[j] := (ya^j, j)$.
Reordenamos la lista S en función de las primeras entradas de sus términos.
Para $j = 1, \ldots, s$ hacemos $T[j] := (a^{sj}, j)$
Reordenamos la lista T por la primera entrada
Buscamos la primera coincidencia entre los primeros términos ambas listas, es decir, los enteros r y t tales que $ya^r = a^{st}$.
Caculamos $x = ts - r$

El algoritmo requiere almacenar una tabla con $\mathcal{O}(\sqrt{n})$ entradas. Además el coste computacional de ordenar una lista de $\sqrt{n}$ entradas es de $\mathcal{O}(\sqrt{n}\log(\sqrt{m}))$. Por lo tanto la complejidad temporal del algoritmo es $\mathcal{O}(\sqrt{n}\log^3 n)$ y la espacial $\mathcal{O}(\sqrt{n})$. Obviamente es intratable para valores grandes de n.

Index-calculus. Describiremos brevemente las ideas esenciales de este método de calculo del logaritmo discreto para el caso del grupo multiplicativo de $\mathbb{Z}/p\mathbb{Z}$, p un número primo. Fijemos un generador multiplicativo g de $(\mathbb{Z}/p\mathbb{Z})^*$ y supongamos que deseamos calcular el logritmo discreto de y respecto de g, es decir, el entero x con $1 \leq x \leq p-1$ que satisface la igualdad $g^x \equiv y \bmod p$.

Supongamos que fijamos primos $B = \{p_1, \ldots, p_k\}$ "pequeños" y que encontramos r de manera que g^r se puede factorizar en B: $g^r \equiv \prod_{i=1}^{k} p_i^{e_i} \bmod p$. La congruencia anterior induce la congruencia lineal

$$r \equiv e_1 \log_g(p_1) + \cdots + e_k \log_g(p_k) \bmod p - 1 \ .$$

Si disponemos de suficientes ecuaciones lineales del tipo anterior (en las que conocemos r y $e_1, \ldots, e_k$) podemos resolverlas para calcular los logaritmos discretos en la base g de los primos $p_1, \ldots, p_k$, $x_1, \ldots, x_k$. Nótese que esta parte es una computación previa que no depende del entero y.

Ahora, para calcular $\log_g(y) = x$ elegiremos m al azar hasta encontrar uno de manera que $g^m y \bmod p$ se factorice completamente en B, en tal caso tendremos $yg^m \equiv \prod_{i=1}^{k} p_i^{r_i}$ y como consecuencia

$$\log_g(y) \equiv -m + r_1 x_1 + \cdots + r_k x_k \ .$$

El método anterior se implementa tomando una cota b (para el que hay algunas elecciones razonables) y como conjunto de primos B todos los menores o iguales que b. Para el cálculo de los r se hacen también elecciones aleatorias $r \in [1, p-2]$ hasta encontrar b casos en los que g^r se factorice completamente en B. La elección de m también se puede hacer al azar hasta encontrar un m satisfactorio.

Además de los algoritmos ya comentados merece también mención el método ρ de Pollard para el cálculo del logaritmo discreto, su base es similar al algoritmo de factorización del mismo nombre y utiliza también el truco de Floyd para simplificar el test que busca las colisiones.

5.. Protocolos criptográficos

Autentificación, firma digital. Esquemas de compartición de secretos: esquema umbral de Shamir. Transferencia inconsciente. Pruebas de conocimiento cero.

Actualmente las aplicaciones de la criptografía rebasan ampliamente la tradicional de la transmisión segura de la información. Es habitual agrupar dichas aplicaciones bajo el epígrafe común de *Protocolos criptográficos*. Entendemos por un protocolo criptográfico un conjunto bien definido de reglas (es decir, un protocolo) encaminadas a una tarea específica y que entre sus herramientas precisa algún sistema criptográfico.

Hay numerosos protocolos criptográficos que intentan dar respuesta a múltiples problemas. Tanto las necesidades como el número de protocolos aumenta día a día, ya que el problema de la seguridad plantea continuamente nuevas exigencias y nuevas situaciones. Sin embargo hay algunas técnicas básicas que son de uso generalizado y casi general en la mayor parte de los protocolos concretos. En este tema nos dedicaremos sobre todo a ellas, dejando aparte las aplicaciones a problemas concretos. Alguno de estos últimos se puede desarrollar o plantear como trabajo sin mayores dificultades, ya que las técnicas que se usan son conocidas al menos en su parte esencial.

Entre los protocolos de uso universal más conocidos mencionemos los siguientes:

- *Protocolos de Autentificación de Usuario*: Cuando B recibe un mensaje, que pretende ser de A, necesita una forma de garantizar que realmente es A quien lo envia.

  La identificación del usuario A puede ser directa, si el receptor B puede comprobar algo que caracteriza al emisor A sin ninguna duda. Este es el caso de la *Firma Digital*. O también puede ser indirecta, por ejemplo planteando al supuesto emisor A una pregunta que sólo el auténtico A puede responder satisfactoriamente (*Desafio-Respuesta*).

- *Protocolos de Autentificación del Mensaje.* Tratan de garantizar que el mensaje que A envía a B no ha sido alterado por una tercera persona.

- *Protocolos para Compartir Secretos.* Son métodos para distribuir una cierta información (el *Secreto*) entre un conjunto $\mathcal{P}$ de participantes, de forma que solamente ciertos subconjuntos prefijados de $\mathcal{P}$ puedan, uniendo sus informaciones parciales (participaciones), recuperar la información original. El ejemplo estándar es la distribución de una clave para abrir una caja fuerte entre n usuarios de manera que para poder abrir la caja se necesiten un mínimo de t de ellos.

- *Transferencia inconsciente.* La idea básica es que un usuario A puede transmitir un mensaje a B de tal forma que al finalizar A sepa que B ha recibido el mensaje sólo con una cierta probabilidad, la misma que tiene B de haber recibido el mensaje enviado por A. La utilidad de este aparente galimatías es múltiple como veremos.

- *Pruebas de Conocimiento Cero.* Técnicas que permiten a un individuo convencer a otro de que posee una cierta información, sin revelarle nada sobre su contenido.

Protocolos de Autentificación

En la seguridad de las comunicaciones, junto al clásico requisito criptográfico de la *privacidad o confidencialidad*, es importante el problema de autenticidad del emisor y de no falsificación o alteración del mensaje. Ambos problemas están estrechamente relacionados y la Criptografía de clave pública, permite dar una solución simple y satisfactoria a los mismos. Nos centraremos en una descripción del mismo mediante el RSA.

Supongamos que el emisor j, que dispone de sus claves (n_j, e_j, d_j), envía un mensaje M al usuario i, cuyas claves son (n_i, e_i, d_i), y que desea, además de que el mensaje sea indescifrable por terceras personas, autentificar el mensaje. Procede entonces de la siguiente forma (suponemos que $M < n_j < n_i$):

- Cifra el mensaje M con su clave privada, de manera que obtiene

$$C_1 := M^{d_j} \bmod n_j \ .$$

- Procede después a cifrar el mensaje C_1 mediante la clave pública de i:

$$C := C_1^{e_i} \bmod n_i \ .$$

Cuando i recibe el mensaje procede de la siguiente forma:

- Descifra C mediante su clave privada, $D_1 := C^{d_i} \bmod n_i = C_1$.
- Finalmente utiliza la clave pública de j para obtener $M := D_1^{e_j} \bmod n_j$.

Observamos que, el secreto del mensaje queda garantizado, pues solo i conoce su propia clave privada, lo que garantiza que solo i puede recuperar el mensaje C_1. También queda garantizada la autenticidad del remitente y la integridad del mensaje, ya que solo j conoce su propia clave secreta y por lo tanto solo j ha podido cifrar el mensaje M en C_1 de manera que $M \equiv C_1^{e_j} \bmod n_j$.

El método anterior plantea algunos interrogantes, por ejemplo la condición $M < n_i < n_j$ parece limitar la versatilidad del método, asimismo en un sistema abierto surge la duda de cómo sabe i que el mensaje C o C_1 viene del usuario j (para usar la clave pública de j). Técnicamente se pueden resolver sin demasiadas dificultades, una forma es el método que aparece en el apéndice; dicho método fué probado por una pequeña red de usuarios sin tropezar con ninguna disfunción.

Firma Digital. El método anterior, con RSA, se puede ver también como un ejemplo de *Firma Digital*. La Firma Digital pretende ser el análogo de la firma ordinaria en los diferentes tipos de operaciones electrónicas que lo requieran, por ejemplo, firmas de contratos, autentificación, valor legal, etc. Para ello debe satisfacer una serie mínima de requisitos, entre ellos (además de ser computacionalmente sencilla de generar) citemos:

- *Personal*: Solo el propietario puede generar su propia firma.
- *Segura*: Es decir, imposible de falsificar por un tercero.
- *Fácil de Autentificar*: El receptor y, eventualmente, un juez, deben ser capaces de asegurar que la firma es de quién dice ser.
- *No repudiable*: El autor de la firma no debe tener la posibilidad de rechazarla como falsa.
- *Fácil de Generar*.

Para evitar que un criptoanalista pueda tomar la firma de un usuario (interceptando un mensaje firmado) y adherirla a otro mensaje a su voluntad se requiere también que la firma digital dependa del mensaje firmado. Obsérvese que en la firma ordinaria no se exige este requisito, la suplantación o falsificación de firma es en este

caso más complicada ya que intervienen factores ajenos como la caligrafía personal del firmante.

El método de autentificación con el RSA que hemos descrito antes tiene el inconveniente de requerir un doble cifrado de todo el mensaje y, como consecuencia, si éste es largo se puede convertir en un proceso largo. Para resolver este problema se suelen utilizar las funciones Hash o funciones resumen.

Una función Hash es una función que, para cada mensaje M de longitud arbitraria, produce un nuevo mensaje $H(M)$ de longitud fija y, normalmente, pequeña. Diremos que la función Hash H es unidireccional si es computacionalmente dificil encontrar otro mensaje M' tal que $H(M) = H(M')$. Además se requiere que H sea una función fácilmente computable.

Mediante una función Hash H se puede establecer un protocolo de autentificación del mensaje M (que previamente puede estar cifrado para garantizar su privacidad) de la siguiente forma:

- El emisor j calcula $H(M)$ y procede a su cifrado mediante su clave privada, $N := H(M)^{d_j} \bmod n_j$.
- Envía al usuario i el par (M, N).
- Cuando i recibe la pareja (M, N) procede a calcular $H(M)$ a partir de M y a descifrar N mediante la clave pública de j, obteniendo $N' := N^{e_j} \bmod n_j$.
- Compara N' con $H(M)$. Si coinciden aceptará el mensaje como procedente de j y, eventualmente, procede a descifrarlo. En caso contrario lo rechaza.

Las funciones hash más conocidas y utilizadas son la MD5 (usada en las primeras versiones de PGP) que por cada bloque de 512 bits produce una salida de 128 y el algoritmo SHA, desarrollado para su uso en el estandar de firma digital DSS, que produce un resumen de 160 bits a partir de bloques de 512.

Firma DSS (Digital Signature Standard). Es un sistema de firma digital basado en el esquema de firma digital de ElGamal y propuesto como protocolo estándar por el NIST (National Institute of Standars and Technology). El algoritmo correspondiente, llamado DSA, tiene tres partes principales:

Generación de las claves

Generación de la firma

Verificación de la firma

Generación de las claves: El usuario debe disponer de:

- Un primo p de longitud binaria 512
- Un primo q, divisor de $p - 1$, de longitud 160.
- Un elemento $g \in \mathbb{Z}/p\mathbb{Z}$ de orden multiplicativo q.
- La función SHA (vale cualquier función resumen con una salida de 160 bits).

- Una clave secreta x, $1 < x < q$.
- La clave pública $y = g^x \bmod p$.

Generación de la firma: Para generar la firma digital de un mensaje m, el remitente sigue los siguientes pasos:

1. Elige al azar $k \in \mathbb{Z}/q\mathbb{Z}$.
2. Calcula $r := (g^k \bmod p) \bmod q$.
3. Calcula $s := k^{-1}(H(m) + xr) \bmod q$.
4. La firma digital es la pareja (r, s).

Para verificar la firma (r, s) de un mensaje m, el receptor sigue los siguientes pasos:

Verificación de la firma:

1. Calcula $t := s^{-1} \bmod q$.
2. Calcula $H(m)$.
3. Calcula $r' := (g^{H(m)t} y^{rt} \bmod p) \bmod q$.
4. Acepta la firma (y el mensaje) si $r = r'$. Lo rechaza si $r' \neq r$.

Algunas de las virtudes del sistema son la rapidez, tanto en la generación de las claves como en la generación de la firma (nótese que, por ejemplo, r puede estar precomputado, ya que no depende del mensaje), y el hecho de que la firma es relativamente corta. Entre las críticas que ha recibido citemos el que la clave se considera demasiado corta (521 bits) para que se considere segura durante largo tiempo (se estima que sería conveniente aumentarla a 1024), que no puede ser utilizada para distribución de claves y que no es compatible con otros sistemas estándar.

Esquemas para Compartir Secretos

Los esquemas (o protocolos) para compartir secretos tratan de resolver el siguiente problema: Sea N un conjunto finito de usuarios, M una determinada información (el secreto) y G un subconjunto del conjunto de partes de N, $G \subset \mathcal{P}(N)$ (estructura de acceso). Se desea distribuir informaciones parciales de M entre los usuarios, llamemos m_i a la información del usuario $i \in N$ (participación), de manera que solamente la reunión de las informaciones parciales de una agrupación autorizada (por definición los elementos de G) puede descubrir el secreto M. Obsérvese que se requiere un gestor o encargado de la distribución de las participaciones que puede ser ajeno a N. El ejemplo clásico es la distribución de claves entre los empleados para poder abrir la caja fuerte en un banco.

Un protocolo es perfecto para la estructura de acceso G anterior si permite que las agrupaciones autorizadas puedan acceder al secreto M, pero de manera que ninguna no autorizada pueda extraer ninguna información válida sobre M.

Uno de los tipos de esquema más conocidos y simples son los ***esquemas umbral*** (Shamir). Un (n,t)-esquema umbral consta de un conjunto de n participantes y su estructura de acceso está formada por todos los subconjuntos de cardinal mayor o igual que t. Por lo tanto cualquier agrupación de al menos t participaciones del secreto M debe ser capaz de recuperarlo. El protocolo propuesto por Shamir para resolver un (n,t)-esquema unbral es el siguiente:

1. Supongamos que $N = \{1,\ldots,n\}$ y tomemos un primo $p \geq \text{máx}\{M+1, n+1\}$.
2. Construimos aleatoriamente un polinomio de grado $t-1$ de la forma $q(X) = M + \sum_{i=1}^{t-1} a_i X^i$.
3. La participación del usuario i es $m_i = q(i) \bmod p$.

Es claro que mediante interpolación de Lagrange t usuarios cualesquiera pueden recuperar el polinomio $q(X)$ y como consecuencia el secreto $q(0) = M \bmod p$. Sin embargo $t-1$ usuarios no obtienen ninguna información sobre M, ya que cualquier término independiente es compatible con la información que reúnen $t-1$ usuarios. Evidentemente mediante el teorema chino de los restos sobre $\mathbb{Z}$ se obtiene otra solución satisfactoria del problema.

Se pueden plantear numerosas variantes en un esquema umbral, entre ellas las distribuciones jerarquizadas que consisten en diseñar un esquema de reparto de secretos que exija la presencia de t miembros pero de manera que algunos de ellos han de ser siempre los mismos.

Mencionemos que mediante circuitos booleanos monótonos (aquellos que no contienen puertas "NO") se puede diseñar un protocolo perfecto para cualquier estructura de acceso.

Problemas de la distribución de secretos. Cualquier distribución de claves secretas, en particular las de un esquema para compartir secretos, deben cumplir una serie de condiciones que eviten algunos problemas clásicos que aparecen de manera natural. Básicamente éstos son:

- Se debería poder verificar que el usuario i realmente dispone de una participación en el secreto K de forma previa a la recuperación del mismo y sin necesidad de que revele su participación. El método utilizado para esta tarea es el de las cápsulas criptográficas (ver [PS97]).
- El usuario debería poder verificar que realmente recibe una participación del secreto K y no otra cosa. Los esquemas de compromiso con un bit permiten resolver este problema (ver [PS97]).
- Se debería evitar que el administrador del secreto conozca la distribución concreta de participaciones. Una posibilidad es mediante el uso de transferencia inconsciente.

- Deberíamos poder comprobar la identidad de un usuario antes de proceder a recuperar el secreto. Los métodos de transferencia con conocimiento cero permiten resolver este apartado.

En particular, las dos últimas técnicas mencionadas tienen numerosas utilidades por lo que las trataremos a continuación.

Transferencia inconsciente

Plantearemos un método de transferencia inconsciente, mencionando al final alguna de sus aplicaciones.

El objetivo de la transferencia inconsciente es el siguiente: A envía a B dos mensajes cifrados sujetos a las siguientes condiciones:

- B sólo puede descifrar uno de ellos.
- A no sabe cual de los dos puede descifrar A.
- Ambos están seguros de que los dos apartados anteriores son ciertos.

Sistema de Rabin. Supongamos que el secreto, es decir, la información que A desea tranferir a B es la factorización de un entero n como producto de dos primos p y q. El sistema se basa en que si conocemos dos raices cuadradas modulo n, x y x' de un cierto número y que no son gemelas (es decir, $x' \neq \pm x \bmod n$) entonces podemos factorizar n. Los pasos que A y B deben seguir son los siguientes:

1. A envía a B en número n.
2. B escoge aleatoriamente un entero z, con $1 < z < n$ y z primo con n. Envía a A el entero $w \equiv x^2 \bmod n$.
3. A calcula las raíces cuadradas de w: $[x, N-x, y, N-y]$ (con $x \neq \pm y \bmod n$).
4. A envía a B aleatoriamente una de las cuatro raíces.

Si B recibe y ó $N-y$ (para lo cual tiene un 50 % de probabilidades) puede factorizar n. Por otro lado, A no sabe cual de las cuatro raíces conoce B de manera que ni puede engañar con el envío de una de las cuatro raíces ni tiene ninguna certeza de si B ha podido factorizar n. Así pues A sabe que B conoce los factores primos de n con una probabilidad de $1/2$.

Las técnicas de transferencia inconsciente se utilizan en muchos problemas criptográficos, entre ellos el más significativo es el problema de la distribución de secretos. Mediante dichas técnicas es posible diseñar técnicas de distribución de manera que el administrador desconozca cual es el secreto (por ejemplo el password) que corresponde a un usuario. Otros ejemplos significativos son el de la compra anónima, simultaneidad de firmas digitales o correo electrónico con acuse de recibo.

Distribución de claves. El sistema que describimos se basa en la técnica de cifrado probabilístico de Goldwasser y Micali. Supongamos que el administrador A dispone de claves $S = (s_1, \ldots, s_k)$ y cada una de ellas es una secuencia binaria $s_i = (s_{i1}, \ldots, s_{in}) \in \mathbb{Z}_2^n$.

A elige un número $n = pq$ producto de dos primos grandes y un entero y, $1 < y < n$ que no es un residuo cuadrático módulo n pero tal que $\left(\frac{y}{n}\right) = 1$. Los valores (n, y) son públicos, pero no lo son p y q. A continuación A cifra las claves mediante:

Para $i = 1, \ldots, k$, para $j = 1, \ldots, n$

Elige aleatoriamente x_{ij}, $1 < x_{ij} < n$ y primo con n.

$z_{ij} = x_{ij}^2 y^{s_{ij}}$

Obsérvese que el símbolo de Jacobi de z_{ij} es siempre 1, por lo que nadie que no conozca la factorización de n puede saber si z_{ij} es o no un cuadrado.

Supongamos ahora que B desea adquirir la clave s_i. Para ello elige aleatoriamente n bits $(a_1, \ldots, a_n)$ y n enteros primos con n: $(r_1, \ldots, r_n)$, $1 < r_i < n$. Para $j = 1, \ldots, n$ calcula

$$c_{ij} \equiv z_{ij} r_j^2 y^{a_j} \equiv (x_{ij} r_j)^2 y^{s_{ij}+a_j} \mod n \, .$$

Pregunta a A si cada uno de los c_{ij}, $j = 1, \ldots, n$, es o no un cuadrado módulo n. Con esta información que le proporciona A puede recuperar $s_i = (s_{i1}, \ldots, s_{in})$, ya que conoce $s_i + (a_1, \ldots, a_n)$ y conoce $(a_1, \ldots, a_n)$. Por otra parte A no sabe cual es el índice i elegido por B, ya que desconoce el valor de los bits $a_1, \ldots, a_n$.

En el sistema anterior hay que adoptar algunas precuaciones, por ejemplo para evitar que el vendedor A ofrezca siempre la misma clave a A enmascarada de k formas.

Pruebas de conocimiento Cero

Como ya mencionamos, las pruebas de conocimiento cero permiten a una persona A convencer a otra B de que posee un cierto *secreto* (por ejemplo los dos factores primos de un número de 200 cifras) sin que al final del protocolo B sepa más acerca del secreto de lo que sabía previamente.

Las soluciones toman la forma de demostración interactiva, también llamadas protocolo de ***desafío–respuesta***, y consisten en un cierto número de etapas del tipo

- B presenta un desafío a A.
- A realiza una cierta computación privada.
- A envía a B una respuesta.

Si alguna de las respuestas es incorrecta B rechaza que A posea el secreto. Si todas son correctas lo acepta.

Para que un protocolo de conocimiento cero sea válido es necesario que

a) Si A posee el secreto siempre puede conseguir que B acepte su demostración.

b) Si A no posee el secreto la probabilidad de que engañe a B pueda hacerse tan pequeña como se quiera.

Veamos algunos ejemplos de este tipo de pruebas.

Logaritmo discreto. Tomemos un primo grande p y dos enteros g, b con $1 < g, b < p$ y siendo g un generador del grupo multiplicativo $\mathbb{Z}_p^*$. Todos los valores anteriores se pueden considerar públicos y por tanto conocidos por A y B. Supongamos que A conoce el logaritmo discreto de b respecto a g, es decir, el entero x con $1 < x < p-1$ tal que $g^x \equiv b \bmod p$. Veamos como el desafiante A puede convencer a B de que realmente conoce x sin desvelar su valor.

1. A elige un entero aleatorio e, $1 < e < p$ y envía a B el entero $z \equiv g^e \bmod p$.
2. B elige aleatoriamente un bit $c \in \{0, 1\}$ y lo envía a A.
3. A calcula y envía a B el entero $y = e + cx \bmod p - 1$.
4. B verifica que $g^y \equiv zb^c$.

Se repite el protocolo anterior un número de veces suficiente hasta que B quede convencido de que A realmente conoce x.

Obsérvese que si A ha enviado realmente la información $z = g^e$ a B, en el caso $c = 1$ no podría responder correctamente al desafío de B si no conoce x. Si A decide hacer trampa y, en previsión de que $c = 1$, decide enviar a B z/b en lugar de z, entonces no podría responder en el caso $c = 0$. El usuario B no recibe ningna información útil para calcular x, ya que en el caso $c = 1$ (único caso en que algo relacionado con x se envía) desconoce también e.

En [Kob87] se pueden ver algunos ejemplos más de pruebas de conocimiento cero, por ejemplo usando transferencia inconsciente. Una de las aplicaciones inmediatas de las pruebas de conocimiento cero es que permiten verificar la identificación de un interlocutor en un sistema de comunicaciones.

Aplicaciones de la criptografía

Las aplicaciones de la criptografía en el mundo de intercambio digital en el que nos movemos son prácticamente infinitas. Intencionadamente nos hemos intentado mover dentro de sistemas y técnicas de uso general, sin entrar en los detalles de las aplicaciones concretas y como se realizan en la práctica.

A grandes rasgos podemos clasificar las aplicaciones en varios grupos cualitativamente diferentes. Por un lado están las aplicaciones a las comunicaciones entre ordenadores, se caracterizan porque solamente requieren del usuario una cierta información inicial, ocupándose a partir de ahí los mismos ordenadores de establecer

las normas. Entre ellas citemos los protocolos de autentificación tipo Kerberos, los de privacidad como PEM o PGP, los de intercambio de documentación, etc. Los problemas de seguridad en las redes y en las transmisiones junto con la aparición de sistemas de rastreo muy eficaces han propiciado la aparición de numerosos programas que, si no garantizan al cien por cien, al menos proporcionan un nivel de seguridad razonable ya que implementan protocolos muy seguros para casi todas las necesidades del usuario (por ejemplo el SSH que permite telnet y transferencia de ficheros seguros).

Otro gran grupo de aplicaciones se encuentra en el desarrollo de la telefonía móvil que precisa garantías suficientes de confidencialidad para el usuario y de identificación para el sistema.

Las transacciones comerciales y bancarias electrónicas son otra fuente de problemas a resolver y aplicaciones de la criptografía. Tanto en su faceta más tradicional de uso de tarjetas de crédito como en su versión más actual de comercio por Internet.

Finalmente mencionemos un amplio grupo de aplicaciones que afectan más directamente a la seguridad jurídica del individuo. Entre ellas la seguridad y confidencialidad de sus datos (jurídicos, médicos, fiscales,), la firma e identificación electrónica, las firmas de contratos y problemas de certificación y notaria, las elecciones electrónicas, Muchas de estas facetas involucran a expertos en varias áreas, desde el derecho a las matemáticas, pasando por economistas e informáticos por ejemplo.

Además de algunos textos como [PS97], [Sta99], [FdlGH$^+$00] que disponen de una amplia información sobre algunas de las aplicaciones, hay numerosa documentación en Internet que es bastante accesible y, en algunos casos, completa. Muchos de los protocolos, sobre todo los de uso público, están también suficientemente documentados en los mismos programas. De manera que es relativamente fácil desarrollar varias aplicaciones.

A modo de ejemplo describiremos el uso de la criptografía en el correo electrónico.

Criptografía y Correo Electrónico. El correo electrónico es posiblemente la aplicación más universalmente aceptada en la nueva era de las telecomunicaciones. Es un sistema rápido y fiable de transmisión de información, sin embargo es también vulnerable. Los dos tipos de riesgos más importantes son los derivados de interferencias y/o ataques por parte de superusuarios que abusan de su poder legítimo (como los administradores de sistemas o diferentes estancias gubernamentales) y por parte de usuarios no autorizados, que pueden interferir mensajes y falsificarlos en beneficio propio (por ejemplo con objeto de usar nuestra tarjeta de crédito o acceder a información personal). Otros tipos de riesgos provienen de un uso inadecuado, por ejemplo para la firma de un contrato sin cuidarse antes de extremos como la falta de autentificación del usuarios o la falta de un acuse de recibo.

Mediantes técnicas criptográficas específicas es posible asegurarse ante la mayoria de los riesgos en su uso. En particular es posible garantizar la autenticidad de los

mensajes, la integridad de los mismos o la identidad de remitente y receptor.

Actualmente se utilizan principalmente dos sistemas: el PEM (*Privacy Enhanced Mail*) standard de Internet, utilizado con SMTP (Internet Standard Simple Mail Transfer Protocol) y el PGP (*Pretty Good Privacy*), creado por P. Zimmermann, que ha desbancado a PEM, convirtiéndose en el standard de facto para la mensajeria segura en Internet.

Ambos sistemas, PEM y PGP, son muy similares: combinan criptografía pública y privada y proporcionan los mismos servicios de cifrado y firma digital (más otros como compresión y conversión a caracteres ASCII). Describiremos brevemente como funciona el PGP (las diferencias entre ambos no son significativas).

En ambos casos el usuario dispone de de dos ficheros donde están almacenadas las claves del sistema asimétrico (un sistema RSA tanto en PEM como en PGP): las públicas de los posibles destinatarios (en PGP se llama pubring.pkr) y sus propias claves privadas (secring.skr). También dispone de un generador aleatorio K encargado de generar las claves criptográficas del sistema simétrico (PGP usa el sistema IDEA, mientras que PEM usa DES), llamadas claves de sesión. Por supuesto también dispone de los programas de cifrado y descifrado para ambos sistemas criptográficos.

Envío de un mensaje: Supongamos que un usuario A desea enviar el mensaje m al usuario B. Los pasos que sigue el programa son los siguientes:

1. K genera una clave de sesion k para el sistema simétrico.
2. El mensaje m se cifra mediante k, $m' := C(m, k)$.
3. Se busca en el fichero de claves públicas la clave pública del usuario B, e_B, y se cifra la clave k con ella: $k' := RSA(k, e_B)$.
4. Se envía a B la pareja $M' := (k', m')$ (de hecho k' es la cabecera del mensaje a enviar).

Recepción del mensaje: Para descifrar el mensaje $M' = (k', m')$ se siguen los siguientes pasos:

1. El programa lee la cabecera k' y nos solicita una contraseña b del usuario B.
2. Si b es correcta, accede al fichero de claves privadas de B y lee su clave d_B.
3. Descifra la clave k con d_B.
4. Conocida la clave de sesión k descifra el mensaje m' con k, recuperando $m = D(m', k)$.

Para generar la firma digital de un mensaje ambos sistemas utilizan el sistema DSS con la función hash SHA (en las versiones antiguas PGP usaba RSA con la función resumen MD5). Dependiendo del tipo de mensaje que enviemos podemos decidir incluir la firma digital en el propio fichero o enviarla aparte (por ejemplo no se puede adherir a un fichero ejecutable).

Los objetivos de garantizar que la clave del destinatario B corresponde realmente a B (autenticidad del receptor), asi como garantizar que el mensaje que recibe B viene de A y no de un intruso (autenticidad de origen) se consiguen mediantes sistemas de certificación de claves. La mayor diferencia entre ambos sistemas (PEM y PGP) es el mecanismo de certificación que usan.

En el caso del PEM la certificación es jerarquizada, es decir, está a cargo de una autoridad certificadora en la que confían ambos usuarios para garantizar y certificar sus identidades y claves. El sistema seguido se llama X.509, la certificación consiste en que la autoridad certificadora garantiza con su firma digital los datos del usuario, entre ellos su clave pública. En el PGP la certificación de claves es un sistema distribuido y anárquico (es decir no existe una autoridad central como en los sistemas jerárquicos) en el que cada usuario puede certificar las claves e identidades de todos aquellos en los que confía. Así pues se trata de una distribución de tipo social en la que la certificación entre dos usuarios desconocidos se lleva a cabo a través de un amigo común (de un usuario en el que ambos confían).

6.. Apéndice: Complejidad

Para una correcta comprensión de los métodos, los problemas y el alcance de las posibles soluciones que se plantean y/o resuelven dentro del álgebra computacional en general y, en el contexto de este curso, en la criptología es conveniente conocer los rudimentos de la teoría de la complejidad.

En la práctica un problema matemático se puede interpretar como una terna (X, Y, f), donde X e Y son conjuntos y $f : X \to Y$ una aplicación de X en Y. Cada elemento $x \in X$ se llama una *instancia* del problema y, dado $x \in X$, $f(x)$ es la solución de la instancia x. Son especialmente importantes los problemas de ***decisión***, en ellos el conjunto Y tiene dos elementos que podemos representar como $\{0, 1\}$. En este caso también se dice que f es un predicado o una propiedad y se interpreta $f(x) = 1$ como que x satisface la propiedad f y $f(x) = 0$ como que x no satisface la propiedad f. La importancia de los problemas de decisión (también llamados dicotómicos) radica en que la mayoria de los problemas se pueden reducir a un problema de decisión. Nótese que formular un problema de decisión consiste en definir un subconjunto Z de X, el formado por las instancias $x \in X$ para las que la propiedad es verdad, es decir, $Z = f^{-1}(1)$.

Veamos cómo se formulan en estos términos dos problemas bien conocidos. El problema de factorizar un número entero positivo se puede adaptar a nuestra notación tomando $X = \mathbb{N}$ y, definiendo, para un número entero $x \in X$, $f(x)$ como la lista de pares de enteros positivos $[(p_1, n_1), \ldots, (p_r, n_r)]$ de manera que $p_1, \ldots, p_r$ son números primos distintos y $x = \prod_{i=1}^{r} p_i^{r_i}$. Por tanto Y se podría tomar como conjunto formado por todas las listas finitas de enteros positivos. De la misma forma,

el problema de determinar si un número es primo o no es un problema de decisión en el que $f(x) = 0$ si y sólo si x es compuesto y $f(x) = 1$ si y sólo si x es primo.

Cuando estamos interesados en problemas computacionales, se exige que los conjuntos X e Y sean constructibles en un número finito de operaciones a partir de un conjunto primitivo (normalmente $\{0, 1\}$) y decimos que el problema es computable (o resoluble) si existe un ***algoritmo***, A, de manera que $A(x) = f(x)$ para cada instancia x del problema. Un *algoritmo* (también llamdo un procedimiento efectivo) es una sucesión finita de etapas (operaciones) que se inician a partir de los datos de entrada (es decir, de la instancia x) y de manera que cada una de ellas depende exclusivamente de x y de las etapas anteriores. No se admiten por tanto decisiones subjetivas o elecciones arbitrarias en una etapa del algoritmo. El concepto de algoritmo se puede formalizar completamente mediante el concepto de máquina de Turing determinista.

Complejidad computacional. Sea F un conjunto finito (alfabeto) y F^* el conjunto de palabras formadas por símbolos de F. La longitud $\lambda(x)$ de una palabra es el número de símbolos que la forman. Supondremos para simplificar que las instancias de un problema computacional son elementos de F^* ya que la generalización a varias entradas en diferentes alfabetos es relativamente simple.

Dado un algoritmo A, su complejidad (o complejidad temporal), es la función $t_A(n)$ que expresa el máximo tiempo invertido por A sobre instancias (inputs) de longitud n. Claramente la definición que hemos dado es ambigüa ya que los tiempos de ejecución dependerán de muchos factores, en particular de la máquina en la que los ejecutemos. Por ello, normalmente, $t_A(n)$ representa el número de operaciones *elementales* del proceso. Por supuesto esto nos lleva a otra imprecisión: ¿qué es una operación elemental? La forma más pura de resolver el conflicto es referir todos los datos al lenguaje binario, que al fin y al cabo es que se utlizará. En este caso la longitud de los inputs es la longitud binaria, (es decir, F=$\{0,1\}$) y una operación elemental es la adición de dos bits, llamada *operación bit* (en el caso de una máquina de Turing cada paso de la misma). Sin embargo en la práctica es más cómodo referirse a unidades de longitud y de medida más adaptadas al problema concreto que estemos tratando, en este caso se requiere conocer perfectamente la complejidad de dichas operaciones elementales. Veamos algunos ejemplos que clarifican los comentarios anteriores y que alertan sobre lo que se puede entender por operaciones elementales

1. En el problema de ordenar una lista de n números enteros es más razonable tomar n como la longitud del dato de entrada y como operación elemental la comparación entre dos de ellos.

2. Un algoritmo clásico de polinomios sobre un cuerpo K puede tomar como longitud de la entrada el grado del mismo y como operaciones elementales las del cuerpo base. Esto requiere que las operaciones del cuerpo base sean computables y su complejidad bien determinada.

3. Admitamos como operaciones elementales de números enteros la suma, el producto, la división entera y el calculo de factoriales. En este caso el teorema de Wilson nos proporciona un test de primalidad extraordinario: n es primo si y sólo si $(n-1)!$ es congruente con -1 módulo n precisa de dos operaciones elementales. La trampa es que el cálculo del factorial dista mucho de ser elemental, su tiempo de ejecución es $c \cdot 2^{2m} m^2$ veces mas lento que una suma binaria.

Para expresar el comportamiento asintótico (cuando $n \to \infty$) de la complejidad de los algoritmos es común utilizar la que se conoce como notación $\mathcal{O}(-)$. Dadas dos funciones $f, g : \mathbb{N} \to \mathbb{R}_+$ diremos que f es del orden de g (lo que expresaremos como $f(n) \leq \mathcal{O}(g(n))$ o también $f(n) \in \mathcal{O}(g(n))$) si existe una constante $c > 0$ tal que $f(n) \leq c \cdot g(n)$ para todo n mayor o igual que un cierto n_0. Diremos que un algoritmo A es de complejidad ***polinómica*** si $t_A(n) \leq \mathcal{O}(P(n))$ para un cierto polinomio $P(n)$. Obsérvese que si k es el grado del polinomio P entonces la condición anterior es equivalente a decir que $t_A(n) \leq \mathcal{O}(n^k)$. En general los algoritmos de complejidad polinómica son "rápidos", por lo que también se denominan algoritmos *eficientes* y un problema computacional que admita un algoritmo eficiente se considera un problema ***tratable***. Por contra un problema se considera ***intratable*** si no admite (en la práctica, si no se conoce) ningún algoritmo eficiente que lo resuelva. Algoritmos polinómicos son, por ejemplo, el algoritmo multiplicación de enteros o de polinomios con coeficientes números enteros o elementos de un cuerpo finito y el algoritmo de Euclides en los mismos contextos. El calculo de $n!$ es un problema intratable, ya que su tiempo de ejecución es del orden $\mathcal{O}(n^2 \log^2 n)$, es decir, en función de la longitud binaria m del entero n es del orden $\mathcal{O}(2^{2m} m^2)$. Los algoritmos conocidos de factorización de un número natural son todos ellos no polinómicos. Aunque se desconoce si existe algún algoritmo polinómico que resuelva el problema, se cree que es un problema intratable.

La notación $\mathcal{O}$ para medir la complejidad no es la única posible, aunque sí la más extendida. Nótese que en dicha notación no se tienen en cuenta ni el producto ni la suma de constantes. En cierto modo este hecho contribuye a que la medidad de la complejidad sea independiente de la máquina, el programador y el alfabeto de entrada. Desde el punto de vista informático esta característica se recoge con el nombre de principio de invariancia, que afirma que dos implementaciones distintas de un mismo algoritmo no diferirán en su eficiencia en más de una constante multiplicativa.

Clases de complejidad. La clase de complejidad $\mathcal{P}$ (polinómica) está formada por aquellos problemas computacionales para los que existe un algoritmo polinómico que los resuelve. La clase $\mathcal{EXP}$ está formada por aquellos problemas de complejidad exponencial, es decir los problemas que admitan un algoritmo del orden $\mathcal{O}(a^n)$ para inputs de longitud n, siendo a una constante.

Una clase de gran interés es la denominada $\mathcal{NP}$ (non-deterministic polynomial

time). Está formada por los problemas de decisión $Z \subset X$ para los que existen un conjunto Q y un algoritmo polinómico $A : X \times Q \to \{0, 1\}$ de manera que para cada instancia $x \in X$ se tiene la condición

$$x \in Z \iff A(x, q) = 1$$

para algún $q \in Q$ de longitud polinómica en la longitud de x. Dicho elemento q se llama un *certificado* para x. Los elementos del conjunto Q se interpretan como "soluciones"posibles al problema, por tanto, un problema está en la clase $\mathcal{NP}$ si, dada una posible solución (obtenida por cualquier método, en particular el método usado no tiene porque ser polinómico en tiempo), es posible determinar si es correcta o no en tiempo polinomial. Otra caracterización establece que un problema es $\mathcal{NP}$ si su complejidad temporal es polinómica para una computación en paralelo (la computación en paralelo consiste en la ejecución en paralelo de tantos procesos como sean necesarios, el número de los mismos es)

También se define la complejidad espacial, s-complejidad, como la función $s_A(n)$ que expresa el máximo espacio de almacenamiento requerido por A durante su ejecución para instancias de longitud n. En función de ella se puede definir la clase $\mathcal{PSPACE}$ como la de aquellos problemas con complejidad espacial polinómica y la clase $\mathcal{EXPSPACE}$ de los problemas de complejidad espacial exponencial.

La diferencia entre algoritmos polinómicos y otros no polinómicos (por ejemplo los exponenciales) es muy significativa asintóticamente, aunque para grados k del polinomio grandes, así como para valores grandes de la constante "oculta", c, conviene ser precavido. Por ejemplo, imaginemos tres algoritmos A, B, C para resolver un problema de manera que $t_A(n) = 10^n$, $t_B(n) = n^{10^3}$ y $t_C(n) = 10^{10000}n^2$. Es claro que $\mathcal{O}(t_C(n)) \leq \mathcal{O}(t_B(n)) \leq \mathcal{O}(t_A(n))$, por lo que asintóticamente el algoritmo C es preferible. Sin embargo A es más rápido que B para enteros de hasta 3000 cifras decimales, A es más rápido que C hasta 10^4 cifras y B es más rápido que C hasta enteros con 10^{10} cifras decimales. No cabe duda de que si limitamos nuestros cálculos a números menores que 10^{3000} el mejor algoritmo será el A. Curiosamente practicamente todos los algoritmos polinómicos útiles son de grados bajos, lo que da lugar a que la eficiencia de los algoritmos (o la tratabilidad de los problemas) se interprete hoy día como que *un problema natural se puede resolver mediante un algoritmo eficiente si y sólo si tiene un algoritmo polinómico* (Cook-Thesis).

El ejemplo anterior justifica el hecho de que a veces la identificación ciega de algoritmos eficientes o polinómicos con "buenos" algoritmos es engañosa. Por otro lado, un algoritmo eficiente puede ser lento para nuestras necesidades, mientras que un algoritmo no eficiente que mejore sensiblemente otros conocidos se puede considerar un gran avance. Por ejemplo, es sencillo convencerse que el algoritmo clásico de multiplicación de números naturales de longitud n es un algoritmo del orden $\mathcal{O}(n^2)$. No cabe duda que se trata de un algoritmo eficiente, sin embargo en cualquier proceso computacional presumiblemente será necesario hacer gran cantidad de multiplicaciones, por ello es deseable disponer de algoritmos lo más rápidos posibles.

El algoritmo de multiplicación más rápido que se conoce actualmente se debe a Schönhage y Strassen y es, para inputs de longitud n, de orden $\mathcal{O}(n \log n \log\log n)$, lo que supone un avance significativo: si $n = 1000$ el segundo algoritmo será unas 30 veces más rápido que el primero, mientras que si $n = 10^6$ ya es casi 12000 veces más rápido.

Algoritmos probabilísticos. Un algoritmo se dice que es probabilístico si en en ciertos estados bien definidos se permiten elecciones aleatorias, dicho de otro modo, el estado siguiente no está unívocamente definido por el actual sino que se puede elegir entre dos (o más) posibilidades con la misma probabilidad. Las elecciones aleatorias realizadas en la ejecución del algoritmo se pueden entender como un nuevo input elegido al azar de un conjunto Q bien definido. De manera que un algoritmo probabilistico AL consiste en un algoritmo (determinista) $A : X \times Q \to Y$ de manera que para un input $x \in X$, AL hace una elección al azar de un elemento $q \in Q$. Nótese que para una instancia $x \in X$ y para una elección concreta de $q \in Q$ se tiene un resultado concreto, sin ambigüedades. Los primeros algoritmos probabilísticos fueron el algoritmo de Berlekamp de factorización de polinomios sobre un cuerpo finito y el algoritmo de primalidad de Solovay-Strassen.

La clase de complejidad $\mathcal{RP}$ (random polynomial time) está formada por los problemas de decisión $Z \subset X$ para los que existe un algoritmo probabilístico, AL, de manera que A es polinómico, la longitud de los elementos de Q es polinómica en la longitud del input $x \in X$ y de manera que

1. Si $x \in Z$ entonces $A(x, q) = 1$ con probabilidad mayor que $1/2$
2. Si $x \notin Z$ entonces $A(x, q) = 0$ para todo $q \in Q$.

Los algoritmos que responden a la resolución de problemas de clase $\mathcal{RP}$ se llaman de tipo *Montecarlo*. La clase $\mathcal{RP}$ es una subclase de la clase $\mathcal{NP}$. Nótese que si se encuentra $q \in Q$ tal que $A(x, q) = 1$ entonces forzosamente $x \in Z$ (se dice que q es un testigo), por lo que iterando k veces el algoritmo la probablidad de que $z \in Z$, pero $A(x, -) = 0$ para todas las elecciones de q efectuadas es menor que $1/2^k$.

La clase de los problemas $Z \subset X$ tales que tanto Z como $X \setminus Z$ pertenecen a la clase $\mathcal{RP}$ se denomina $\mathcal{ZPP}$ (zero-error probabilistic polynomial type). Los problemas de la clase $\mathcal{ZPP}$ se resuelven siempre de manera satisfactoria, siendo el tiempo empleado en los mismos una variable aleatoria con un valor esperado polinómico en la longitud de x. Los algoritmos que responden a esta definición se llaman también tipo *Las Vegas*.

El problema $\mathcal{P} = ?\mathcal{NP}$. La siguiente cadena de contenciones aclara la situación relativa de las clases descritas:

$$\mathcal{P} \subset \mathcal{ZPP} \subset \mathcal{RP} \subset \mathcal{NP} \subset \mathcal{PSPACE} \subset \mathcal{EXP} \subset \mathcal{EXPSPACE} \subset$$

De ellas, solo se conoce que la contención $\mathcal{P} \subset \mathcal{EXP}$ es estricta. Actualmente el problema más importante de la teoría de la complejidad (y uno de los importantes tanto en matemáticas como en computación) es saber si la contención $\mathcal{P} \subset \mathcal{NP}$ es estricta, dicho problema se conoce como el problema $\mathcal{P} = ?\mathcal{NP}$. Aparece en casi todas las listas de grandes problemas de matemáticas para este siglo; por ejemplo la fundación Clay premia con un millón de dólares la solución del mismo.

Problemas $\mathcal{NP}$-completos. Dados dos problemas de decisión $Z \subset X$ y $Z' \subset X'$, se dice que (Z, X) es una reducción polinómica de (Z', X') si existe un algoritmo polinómico $A : X' \to X$ de manera que $x \in Z'$ si y sólo si $A(x) \in Z$. Si cualquier problema (Z', X') de una clase de complejidad $\mathcal{C}$ se puede reducir polinómicamente a un problema fijo (Z, X) se dice que éste último es un problema **$\mathcal{C}$-hard**. Si además el problema (Z, X) está en la clase $\mathcal{C}$ se dice que es un problema **$\mathcal{C}$-completo**. Obsérvese que si para un problema $\mathcal{NP}$-completo se prueba su pertenencia a la clase $\mathcal{P}$ entonces $\mathcal{P} = \mathcal{NP}$. De la msima forma, si probamos que dicho problema no está en $\mathcal{P}$ necesariamente $\mathcal{P} \neq \mathcal{NP}$. Actualmente se conocen miles de problemas $\mathcal{NP}$-completos (vease por ejemplo Garey-Johnson, [GJ79]). Obsérvese que una vez que se conocen varios problemas $\mathcal{NP}$-completos es relativamente fácil encontrar otros, ya que basta demostrar que el nuevo problema es de clase $\mathcal{NP}$ y que es una reducción de alguno de los conocidos. La noción de problema $\mathcal{NP}$-completo se debe a Cook (1971) y las reducciones polinómicas a Karp (1972). El primer problema $\mathcal{NP}$-completo conocido fue el problema de saber si una expresión booleana es satisfactoria, es decir, si es posible asignar valores de verdad o falsedad a las variables de manera que la expresión sea verdadera (Cook, 1971). Citemos algunos ejemplos sencillos más:

1. El problema de las subsumas (base del sistema criptográfico de las mochilas) consiste en, dados un número finito de números naturales $a_1, \ldots, a_n, s \in \mathbb{N}$, saber si existen $x_1, \ldots, x_n \in \{0, 1\}$ tales que $\sum_{i=1}^{n} x_i a_i = s$.

2. El problema de decidir si los vértices de un grafo se pueden colorear usando sólo 3 colores y de manera que no haya dos vértices del mismo color conectados por una arista.

3. El problema del viajante: dado un grafo metrizado y un entero k saber si existe un camino que recorre todo el grafo (sin repetir vértices) con un coste menor o igual que k.

4. El problema de la congruencias cuadráticas: dados enteros positivos a, b y c, ¿existe un entero positivo $x < c$ tal que $x^2 \equiv a(\mod b)$?.

5. El problema de resolver ecuaciones diofánticas cuadráticas: dados enteros positivos a, b y c, ¿existen enteros positivos x, y tales que $ax^2 + by = c$?.

Se cree que el problema de factorizar un número es también $\mathcal{NP}$-completo, pero por el momento no se ha probado su pertenencia a la clase $\mathcal{NP}$. Una presentación del problema por Stephan Cook que se puede encontrar en la página web: `http://www.claymath.org/prizeproblems/` contiene información precisa sobre la historia y el estado del mismo.

Bibliografía

[AKS02] M. Agrawall, N. Kayal, and N. Saxena. Primes is in $\mathcal{P}$. *Preprint, disponible en* http://www.cse.iitk.ac.in/news/primality.html, 2002.

[Chi95] L. Childs. *A Concrete Introduction to Higher Algebra (2nd Ed.)*. UTM. Springer-Verlag, 1995.

[Coh93] H. Cohen. *A course in computational algebraic number theory*. GTM. Springer-Verlag, 1993.

[CP01] R. Crandall and C. Pomerance. *Prime numbers. A computational perspective*. Springer-Verlag, 2001.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptogrhphy. *IEEE Trans. Inform. Theory*, 22:644–654, 1976.

[FdlGH$^+$00] A. Fúster, D de la Guía, L. Hernández, F. Montoya, and J. Muñoz. *Técnicas criptográficas de protección de datos. 2^a Ed.* RA-MA, 2000.

[Gib93] P. Giblin. *Primes and programming*. Cambridge Univ. Press, 1993.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the theory of $\mathcal{NP}$-completeness*. Freeman, 1979.

[Gor84] J Gordon. Strong prmes are easy to find. *Advances in Criptology. EUROCRYPT'84-Proceedings. Springer Verlag*, pages 216–223, 1984.

[Kah96] D. Kahn. *The codebreakers*. Scribner, New York, 1996.

[Kob87] N. Koblitz. *A Course in Number Theory and Cryptograhy*. Springer-Verlag, 1987.

[Kob98] N. Koblitz. *Algebraic aspects of cryptography*. Springer-Verlag, 1998.

[LT90] F. López and J. Tena. *Introducción a la teoría de los números primos*. Publicaciones de la Universidad de Valladolid, 1990.

[MT97] C. Munuera and J. Tena. *Codificación de la Información*. UVA, 1997.

[PS97] J. Pastor and M.A Sarasa. *Criptografía digital. Fundamentos y aplicaciones*. Prensas de la U. de Zaragoza, 1997.

[Sin00] S. Singh. *Los Códigos Secr3tos*. Ed. Debate, 2000.

[Sta99] W. Stallings. *Cryptography and network security. Principles and practice*. Prentice Hall, 1999.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1999.

[Yan00] S.Y. Yan. *Number Theory for Computing*. Springer-Verlag, 2000.