Elementos de criptografía

Llorenç Huguet Rotger
Josep Rifà Coma
Juan Gabriel Tena Ayuso

PID_00200951

UOC
Universitat Oberta de Catalunya

Los textos e imágenes publicados en esta obra están sujetos –excepto que se indique lo contrario– a una licencia de Reconocimiento-NoComercial-SinObraDerivada (BY-NC-ND) v.3.0 España de Creative Commons. Podéis copiarlos, distribuirlos y transmitirlos públicamente siempre que citéis el autor y la fuente (FUOC. Fundació per a la Universitat Oberta de Catalunya), no hagáis un uso comercial y no hagáis una obra derivada. La licencia completa se puede consultar en http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.es

Índice

Introducción 5

Objetivos 8

1. Criptosistemas simétricos o de clave privada 9
1.1. Criptosistema DES 9
1.2. Criptosistema IDEA 12
1.3. Criptosistema AES 13
1.4. Modos de operación de los criptosistemas de clave privada ... 14

2. Criptosistemas de clave pública 16
2.1. Funciones unidireccionales 17
2.2. Criptosistema RSA 17
2.2.1. Descripción del criptosistema 18
2.2.2. Firma digital, basada en el RSA 19
2.3. Criptosistema ElGamal 21
2.3.1. Descripción del criptosistema ElGamal 21
2.3.2. Firma digital, basada en el ElGamal 22
2.4. Algoritmo DSA como alternativa a la firma digital RSA 23
2.5. Funciones *hash*: MD5 y SHA-1 25
2.5.1. El algoritmo MD5 26
2.5.2. El algoritmo SHA-1 27
2.6. Infraestructura de clave pública: PKI 29
2.6.1. Sistemas gestores de certificados electrónicos: la recomendación X.509 31
2.6.2. Listas de certificados revocados: CRL 32

3. Criptografía cuántica y post-cuántica 33
3.1. Criptografía cuántica 35
3.2. Los códigos correctores de errores en la criptografía post-cuántica 37
3.2.1. Nociones básicas de códigos correctores de errores ... 37
3.2.2. Códigos lineales 39
3.2.3. Los códigos lineales cíclicos: *BCH* y *RS* 43
3.2.4. Los códigos cíclicos BCH 45
3.2.5. Los códigos cíclicos RS 46
3.3. Los criptosistemas de McEliece y de Niederreiter 47
3.3.1. Criptosistema de McEliece 47
3.3.2. Criptosistema de Niederreiter 49

Ejercicios de autoevaluación 51

Solucionario 52

Bibliografía 56

Introducción

Tradicionalmente, la criptografía tiene como objetivo la transmisión o almacenamiento de mensajes indescifrables para todo receptor que no disponga de la clave del algoritmo de descifrado.

Lectura recomendada

Para hacer más comprensible este módulo didáctico se puede acompañar del libro de Criptografía de J. Domingo, J. Herrera y H. Rifà-Pous de los estudios de Informática y Multimedia, de la UOC.

Hoy, la criptografía se presenta como la solución al problema de la vulnerabilidad de los sistemas de transmisión, o de almacenamiento, con respecto al secreto y a la autenticidad de la información transmitida, o almacenada. El objetivo concerniente a la privacidad y autenticidad asociados a una red de sistemas es evitar que un espía pueda violar o eliminar la protección del sistema en referencia a las líneas de comunicación, a la conexión de acceso a la red (contraseñas) y a la utilización de los recursos de un determinado sistema.

En tiempos pasados, la criptografía ha sido una actividad casi exclusivamente utilizada en la diplomacia y en la guerra, pero, a partir de la Segunda Guerra Mundial, la aparición de los ordenadores ha hecho que todos los sistemas criptográficos utilizados antes, excepto el método de Vernam (basado en claves de un solo uso y del cual se puede demostrar matemáticamente su inviolabilidad), formen parte de la historia puesto que la velocidad en el tratamiento de la información hace que sea un juego de niños el problema de encontrar sus correspondientes claves (criptoanálisis).

De esta simplicidad de los métodos clásicos es un ejemplo el sistema criptográfico, llamado de Julio César, por ser él su primer usuario, utilizado todavía durante la segunda guerra mundial, que consistía en numerar los caracteres alfabéticos y cifrar el mensaje m como el criptograma c, mediante una traslación cíclica que hoy enunciaríamos como $c = (m + k) \pmod{25}$, donde m es el valor numérico asignado a cada letra del alfabeto $\{A = 0, B = 1, \ldots, Z = 24\}$, por ejemplo, y para un cierto valor de k previamente elegido (Cesar escogía $k = 3$). El texto AMOR quedaría, en valores numéricos, 0,12,14,17 que se cifrarían en 3,15,17,20; es decir, se transmitiría el mensaje cifrado (criptograma): DPRU.

Desde siempre, toda técnica criptográfica que opera sobre un mensaje, sin tener en cuenta su estructura lingüística, está basada en una operación ejecutada por el emisor, transformando el mensaje original en un mensaje cifrado, mediante un algoritmo que implementa esta operación, ligado a una clave k. Al mismo tiempo, esta operación posee la operación inversa, ejecutada por el receptor, que permite encontrar el mensaje original.

Todo sistema criptográfico, también denominado criptosistema, consta de cinco componentes: $\{M, C, K, E$ y $D\}$, donde M es el conjunto de todos los mensa-

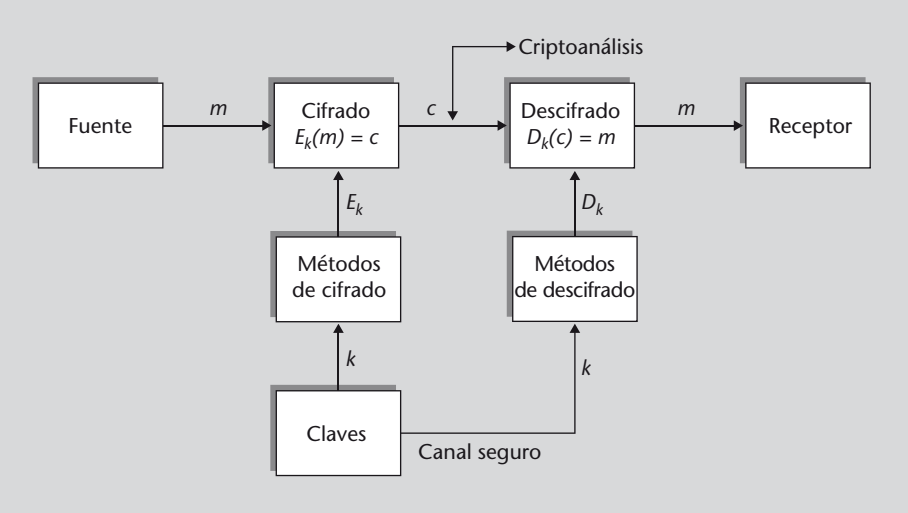
jes a transmitir, C el de todos los mensajes cifrados, K el de las claves a utilizar, E el de todos los métodos de cifrado: $E = \{E_k | E_k(m) \in C, \forall m \in M, \forall k \in K\}$ y D el de todos los métodos de descifrado: $D = \{D_k | D_k(c) \in M, \forall c \in C, \forall k \in K\}$.

Cada método de cifrado de E, y cada método de descifrado de D, está definido mediante un algoritmo, que es común a todos los métodos, y donde cada clave $k \in K$, distinguirá la instancia correspondiente a cada transformación E_k, y D_k, respectivamente.

Para toda clave $k \in K$, la transformación D_k es la inversa de E_k; es decir:

$$D_k(E_k(m)) = m, \forall m \in M$$

Observar sin embargo, que esto no quiere decir que E_k sea el inverso de D_k, en sentido matemático.



Todo criptosistema, tal y como muestra la figura anterior, debe cumplir al menos estos tres requisitos:

1) Todos los algoritmos de cifrado y descifrado E_k y D_k deben ser computacionalmente eficientes.

2) Los algoritmos E_k y D_k deben ser fácilmente implementables.

3) La seguridad del sistema solo debe depender del secreto de las claves $k \in K$, y no de los algoritmos correspondientes de E y D.

Además, siempre se deberá tener en cuenta los objetivos de privacidad y autenticidad, donde se considera:

Principios de Kerchoff

En criptografía, las propiedades deseables de un criptosistema constituyen los principios de Kerckhoff; de entre ellos, los más importantes: Si el criptosistema no es teóricamente irrompible, al menos lo debe ser en la práctica. La efectividad del criptosistema no debe depender de que su diseño permanezca en secreto. El criptosistema debe ser fácil de usar. La clave debe ser fácilmente memorizable, para evitar recurrir a notas escritas. Los criptogramas deberían ser alfanuméricos.

- **Privacidad**: como la incapacidad, para un criptoanalista, de determinar un mensaje original a partir del criptograma que haya podido interceptar.

- **Autenticidad**: como la incapacidad, para un criptoanalista, de sustituir un criptograma falso c', en lugar del criptograma real c, sin que sea detectado.

Consideraciones a los criptosistemas

Actualmente, se consideran dos tipos de criptosistemas, según la utilización y administración de los algoritmos de cifrado y descifrado. El criptosistema clásico o convencional, en el cual la clave correspondiente a ambos algoritmos es la misma o, en su defecto, una fácilmente deducible de la otra. En este caso, cada usuario dispone de su pareja de algoritmos E_k y D_k particulares y ningún otro usuario puede disponer de ellos sin conocer la clave k. Son los llamados **criptosistemas de clave privada o simétricos**.

En el año 1976 entra en juego un nuevo concepto de criptosistema, propuesto por W. Diffie y M. Hellman, llamados **criptosistemas de clave pública o asimétricos**, caracterizados por el hecho de que conocer el algoritmo E_k no revela ninguna información sobre D_k o viceversa. Es decir, una de las claves puede ser revelada públicamente sin peligro de que la otra pueda ser deducida.

Lectura recomendada

W. Diffie; M. Hellman. "New Directions in Criptography". *IEEE Transactions on Information Theory* (vol. IT-22).

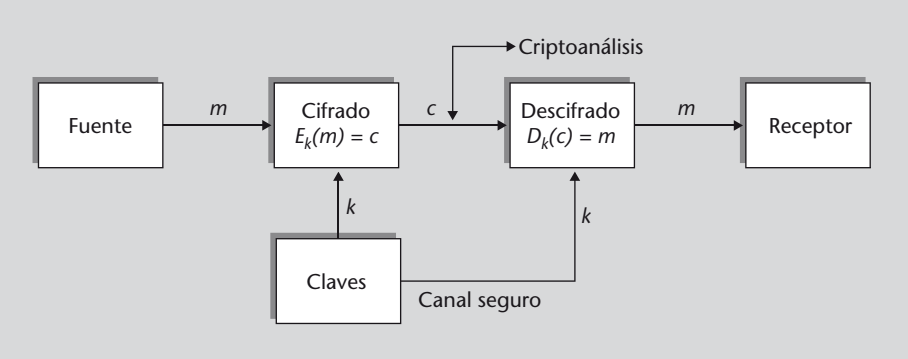
Objetivos

En los materiales didácticos de este módulo el estudiante encontrará los contenidos necesarios para alcanzar los objetivos siguientes:

1. Conocer los sistemas criptográficos de clave simétrica más comunes (DES, IDEA, AES).
2. Conocer los sistemas criptográficos de clave asimétrica más comunes (RSA, ElGamal).
3. Conocer los principales algoritmos usados en las funciones *hash* (MD5, SHA).
4. Conocer los fundamentos y algún ejemplo de sistema criptográfico cuántico.
5. Conocer los fundamentos de la teoría de la codificación para la corrección de errores y los sistemas criptográficos postcuánticos basados en esta (McEliece, Niederreiter).

1. Criptosistemas simétricos o de clave privada

Nos referiremos a los criptosistemas como simétricos o de clave privada cuando el emisor y el receptor comparten una única clave *k*. Por esto, establecemos como característica principal la existencia de un canal seguro a través del cual el emisor transmite al legítimo receptor su clave privada *k* de forma que queda protegida ante un criptoanalista.



El emisor quiere enviar cifrado el mensaje *m*, por lo cual, mediante el algoritmo de cifrado calcula el criptograma *c* a partir de *m* y de la clave *k*:

$$E_k(m) = c$$

El receptor debe ser capaz de descifrar el criptograma *c*, a partir del conocimiento de la clave *k*, es decir, reencontrar el mensaje *m* mediante:

$$D_k(c) = m$$

1.1. Criptosistema DES

En el año 1977 el NBS (National Bureau of Standards), de los EE. UU., anunció un algoritmo de cifrado estándar, el DES (*Data Encryption Standard*, FIPS pub. 46, National Bureau of Standards, (enero 1977)), para que fuera utilizado por todas las agencias federales, con el propósito de hacer compatibles todos los sistemas de protección de la información utilizados en los diferentes estados, bajo un sistema criptográfico común admitido como estándar. La nueva agencia que sustituyó a la NBS, el NIST (National Institute of Standards and

Technology), certificó el DES en el año 1987 y otra vez en 1993, hasta que en 1997 ya no lo certificó. Durante estos años fue considerado estándar a nivel mundial y hoy todavía es utilizado en el intercambio de información entre los cajeros automáticos y los bancos respectivos. Aunque sea por motivos históricos nos parece interesante hacerle referencia.

El DES consiste en un algoritmo de cifrado-descifrado en bloques de 64 bits, mediante una clave k, también de 64 bits (de los cuales solo 56 bits son efectivos).

Los 64 bits de entrada (mensaje original) se transforman a través de una permutación inicial PI, cuya salida se divide en dos sub-bloques L_0 y R_0 de 32 bits cada uno, los cuales están sujetos a un conjunto de 16 transformaciones, de acuerdo con una cierta función f y 16 sub-claves k_i ($i = 1, \ldots, 16$). Tras realizar las 16 transformaciones se reunifican los sub-bloques R_{16} y L_{16} y se le aplica la inversa de la permutación inicial: PI^{-1}.

Enlace de interés

Se puede encontrar un aplicativo de simulación del DES, de uso libre, en la dirección: www.criptored.upm.es.

Si T_i es el resultado de la i-ésima iteración, entonces T_i está formada por dos partes la L_i, que denota los 32 bits más a la izquierda de T_i, y R_i, que denota los 32 bits más a la derecha de T_i; o sea, T_i es la concatenación de L_i con R_i. El cálculo se hace de este modo:

$$L_i = R_{i-1},$$

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

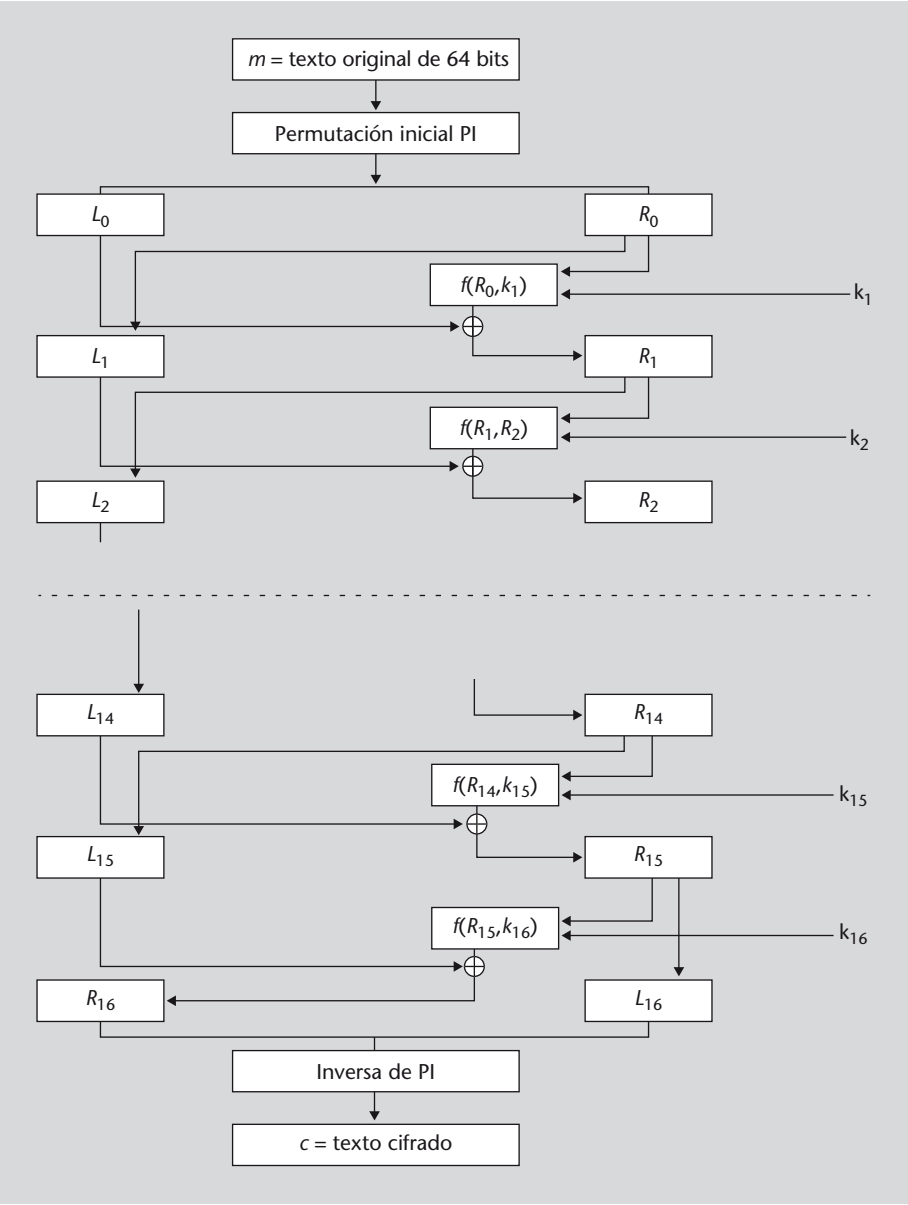
donde $\oplus$ es la operación or-exclusiva y k_i es una sub-clave de 48 bits obtenida a partir de la clave original k.

La función f transforma los 32 bits del bloque R_{i-1}, mediante la sub-clave k_i en los 32 bits del bloque R_i. Para hacer esto, primero se expanden los 32 bits de R_{i-1} en un bloque de 48 bits, utilizando una tabla de expansión E para calcular la or-exclusiva de $E(R_{i-1})$ y k_i, el resultado de la cual se divide en ocho bloques B_i de 6 bits de entrada y cuatro de salida. Estos bits de salida son concatenados por dar un nuevo bloque de 32 bits. Finalmente, la salida de la función f es el resultado de aplicar una cierta permutación P al bloque de 32 bits anterior. Es decir:

$$f(R_{i-1}, k_i) = P(S_1(B_1), S_2(B_2) \ldots S_8(B_8))$$

Cada una de las 16 iteraciones del algoritmo DES utiliza una clave diferente de 48 bits, k_i, calculada a partir de la clave k de 64 bits, la cual posee 8 bits de control en las posiciones 8,16,24,32,40,48,56,63, mediante una permutación P_1 de 56 bits. El resultado $P_1(k)$ se divide en dos partes de 28 bits cada uno, a los que se aplica un desplazamiento a la izquierda diferente para cada sub-clave k_i.

La siguiente figura detalla los pasos del algoritmo de cifrado DES:



Nota

El hecho de que no sean intercambiados R_{16} y L_{16} antes de aplicarles la permutación PI^{-1} obedece a que el algoritmo que describimos también se utilizará para el descifrado.

Para aumentar la fortaleza del DES ante posibles criptoanálisis se propuso la reiteración de los procesos de cifrado/descifrado operando, sucesivamente, sobre el mismo bloque con varias claves independientes. Este es el caso del doble o triple cifrado.

1) Doble cifrado. Dadas las claves independientes k_1 y k_2, los algoritmos de cifrado y descifrado vienen dados por:

$$c = E_{k_2}(E_{k_1}(m))$$

$$m = D_{k_1}(D_{k_2}(c))$$

2) Triple cifrado. Dadas las claves independientes k_1, k_2 y k_3, los algoritmos de cifrado y descifrado vienen dados por:

$$c = E_{k_3}(D_{k_2}(E_{k_1}(m)))$$
$$m = D_{k_1}(E_{k_2}(D_{k_3}(c)))$$

El método de triple cifrado puede ser utilizado para evitar el ataque criptoanalítico del *meet in the middle*.

1.2. Criptosistema IDEA

El International Data Encryption Algorithm es un criptosistema de clave simétrica, que fue diseñado para corregir las debilidades detectadas en el DES. Hoy en día es utilizado en correo electrónico seguro (PGP)

El algoritmo IDEA opera en bloques de 64 bits como texto original, dando un texto cifrado de 64 bits, mediante una clave de 128 bits. Es decir, opera sobre la misma longitud de bloques de bits que el DES, pero con una longitud de clave doble, lo cual añade complejidad al criptoanálisis.

Las operaciones se realizan sobre sub-bloques de 16 bits, cosa que permite su adaptación a arquitecturas de 16 bits. En el algoritmo encontramos tres tipos de operaciones: la or-exclusiva $\oplus$, la suma módulo 2^{16} y el producto módulo $2^{16} + 1$ (este valor, $2^{16} + 1$ es un número primo, por lo cual podemos calcular inversos dentro del cuerpo finito correspondiente).

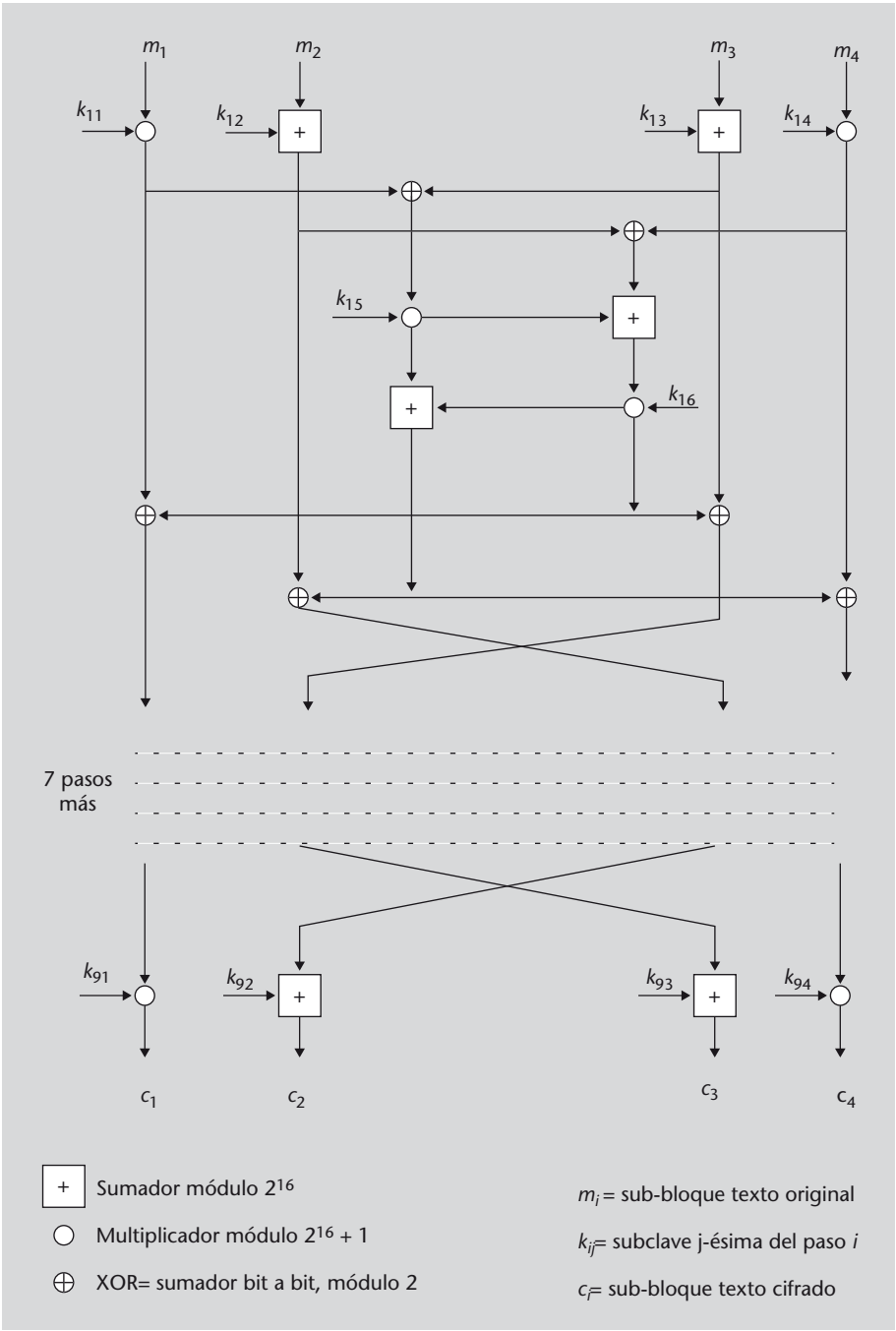
Enlace de interés

Se puede encontrar un aplicativo para la simulación del IDEA, de uso libre, en la dirección: www.criptred.upm.es.

Sea m un bloque de 64 bits que queremos cifrar; éste se dividirá en 4 sub-bloques de 16 bits: m_1, m_2, m_3, m_4, que serán la entrada al algoritmo de cifrado. El algoritmo consta de ocho pasos con las mismas características. En cada paso, intervienen los cuatro sub-bloques de texto y seis sub-claves, también de 16 bits, de forma que en cada paso los bloques segundo y tercero se intercambian. Para acabar, hay un noveno paso, en el cual intervienen los cuatro sub-bloques de texto y solo cuatro sub-claves.

La clave de 128 bits se subdivide en ocho sub-bloques de 16 bits cada uno, los cuales constituyen las primeras ocho sub-claves utilizadas por el algoritmo: seis de estas en el primer paso y las dos restantes son las dos primeras del segundo paso. Para poder continuar, se le aplica a la clave inicial una rotación de 25 bits a la izquierda y la nueva clave resultante se subdivide en ocho sub-bloques de 16 bits. Ahora el algoritmo utilizará las dos sub-claves anteriores seguidas de las cuatro primeras sub-claves provenientes de la subdivisión actual. Las cuatro restantes se utilizarán en el tercer paso, y así sucesivamente

El diagrama de bloques del criptosistema IDEA se muestra en la figura siguiente.



1.3. Criptosistema AES

Durante el periodo que el DES estuvo en vigor (1977-2001) se propusieron decenas de cifrados alternativos, muchos de ellos de dominio público. La NSA (National Security Agency) también dio a conocer en 1994 un nuevo cifrado para ser usado en telefonía y comercio electrónico: Skipjack.

El Skipjack es un algoritmo de cifrado en bloques de 64 bits, fue declarado secreto, y los chips que se han implementado impiden acceder a su código fuente. Quizás proceda así la agencia NSA porque no desea que uno de sus criptosistemas circule abiertamente por todo el mundo, o para poder escuchar conversaciones telefónicas cifradas, o ambas cosas.

Enlace de interés

Se puede encontrar un aplicativo de simulación del AES, de uso libre, denominado *AES Inspector* en la dirección: www.formaestudio.como/rijndaelinspector.

En enero de 1997, el NIST (National Institute of Standards and Technology), viendo que la seguridad del DES estaba ya comprometida (por motivos estrictamente computacionales, porque el DES nunca ha sido roto), convocó a concurso público la adjudicación del nuevo estándar de cifrado. Se denominaría AES, acrónimo de Advanced Encryption Standard. En la convocatoria se especificaba una serie de requisitos mínimos: un cifrado en bloque de 128 bits; con claves de 128, 192 y 256 bits; la posibilidad de ser implementado tanto en *hardware* como en *software* y estar disponible gratuitamente. Pero lo más destacado de la convocatoria era que el concurso estaba abierto a todo el mundo y que el proceso de selección iba a ser totalmente transparente.

Con esta idea se inició una etapa larga de selección que culminó, el año 2000, en la elección del criptosistema Rijndael de los investigadores belgas Vincent Rijmen y Joan Daemen.

Desde este momento, DES ya tiene sustituto: AES. Naturalmente, la decisión del NIST de adoptar el nuevo criptosistema solo obliga a la administración federal americana, y en lo concerniente a la información no clasificada; pero con toda seguridad, AES va a ser el cifrado más usado en los próximos años. La recomendación del cifrado por parte del NIST es todo un certificado de garantía para empresas y organizaciones. Prueba de ello es que la misma NSA ha aprobado el uso de AES para cifrar información clasificada: la **secreta** con claves de 128 bits y la **de alto secreto** con claves de 192 y 256 bits.

El proceso de cifrado de cada bloque de 128 bits de texto original consta de tres transformaciones, o capas distintas donde se tratan los bits, que constan de:

- **Capa de Mezcla Lineal**: difusión de los bits: *ShiftRow* y *MixColumns*.
- **Capa No Lineal**: *ByteSub* (similar a las *S-boxes* del DES).
- **Capa de Adición de Clave**: operaciones con la función or-exclusiva entre el estado intermedio y la sub-clave de cada ronda.

Las operaciones implicadas en AES se expresan en términos algebraicos empleando cierta aritmética de bytes. En esta aritmética, la suma y el producto de bytes son justamente la suma y el producto en el cuerpo finito $\mathbb{F}_{2^8}$, construido a partir del polinomio primitivo: $p(X) = X^8 + X^4 + X^3 + X + 1$.

Criptoanálisis

El disponer de un estándar de cifrado de uso generalizado tiene un gran inconveniente: que todo el mundo quiere romperlo. Al exigir claves con una longitud mínima de 128 bits, el NIST dotaba a su cifrado de una seguridad más que suficiente contra un ataque de fuerza bruta; el único método que venció al anterior estándar, el DES. Descubrir una clave de 128 bits comprobando una a una todas las posibles es una tarea prácticamente eterna para cualquier ordenador de hoy en día. Incluso para todos ellos trabajando juntos. Y 256 bits son garantía de sobra contra todos los ordenadores electrónicos que se construyan en las próximas décadas. Será necesario algo más que fuerza bruta para derrotar al AES. Durante el concurso, el vencedor Rijndael (igual que el resto de los finalistas) probó ser inmune a todos los métodos de criptoanálisis conocidos hasta aquel momento.

1.4. Modos de operación de los criptosistemas de clave privada

Por aumentar la seguridad del algoritmo de cifrado y el de descifrado se utilizan de diferentes modos:

- **Cifrado en bloque**: donde el texto original se procesa en bloques disjuntos de 64 bits, los bloques de salida de los cuales, también de 64 bits, se concatenan para formar el texto cifrado. Este modo suele llamarse **ECB** (*Electronic Code-Book*). Esta manera de cifrar/descifrar impide, por un lado, la supresión y/o inserción de bloques de texto cifrado, porque en cualquier caso el receptor sería incapaz de descifrar el criptograma recibido y, por lo tanto, quedaría alertado de las posibles intrusiones. Por otro lado, los ataques estadísticos también se complican, debido a la interdependencia del texto cifrado a lo largo de todo el proceso. Una alternativa es el denominado cifrado en bloques encadenados, consistente en dividir el texto que hay que cifrar en bloques y hacer depender el bloque n-ésimo de texto cifrado/descifrado del bloque $(n-1)$-ésimo. Es decir:

$$c_n = E_k(m_n \oplus c_{n-1})$$
$$m_n = D_k(c_n) \oplus c_{n-1}$$

  El primer bloque de entrada al proceso de cifrado está formado por la *or* exclusiva entre el primer bloque del mensaje y los 64 bits del vector inicial $c_0 = VI$, el cual es compartido por los algoritmos de cifrado y de descifrado. Este modo suele denotarse **CBC** (*Cipher Block Chaining*).

- **Cifrado en flujo**: operando sobre uno o más bits, desplazando previamente un conjunto de bits de la operación anterior en número suficiente para guardar los nuevos (*stream cipher*).

  Dentro de este modo de cifrado, se considera el modo **CFB** *Cipher Feedback*, que consiste en una or-exclusiva entre los n bits más a la izquierda de la información de salida del proceso de cifrado y los n bits de la información de entrada produciendo n bits de texto cifrado (n es el número bits a cifrar). El criptograma se obtiene a partir de un valor inicial VI y del criptograma anterior. Para cargar estos n bits en el dispositivo de cifrado, desplazaremos el contenido del valor inicial VI n bits a la izquierda.

  Esta modalidad permite el tratamiento de bloques de menos de 64 bits. Normalmente se utiliza para la seguridad de mensajes muy repetitivos y para cifrar/descifrar ficheros donde no conviene almacenar información inútil.

  Existen otros métodos, implementados con registros de desplazamiento; LFSR (*Linear Feedback Shift Register*), en los cuales la clave k se usa para controlar un generador de claves variables (*running key genarator*), que produce una secuencia binaria $k_1,k_2...k_n$ (n debe ser mucho más grande que la longitud de la clave). Así, los dígitos del texto cifrado se forman a partir del texto original, en binario:

$$c_i = m_i \oplus k_i$$

  Evidentemente, el descifrado se hará de forma simétrica: $m_i = c_i \oplus k_i$

Secreto perfecto de Shannon

El hecho de que los registros de desplazamiento que generan la secuencia binaria $k_1,k_2...k_n$ tengan un periodo finito está en contraposición con los requerimiento del secreto perfecto de Shannon, porque, si el texto a cifrar es muy largo, se repetirá la clave de forma determinista.

2. Criptosistemas de clave pública

En el año 1976 entra en juego el nuevo concepto de criptosistema propuesto por Diffie y Hellman, con dos claves, una de las cuales puede ser revelada públicamente sin peligro de que se pueda deducir la otra.

En un criptosistema con claves públicas cada usuario tiene un algoritmo de cifrado E_k, registrado en un directorio público, y un algoritmo de descifrado D_k que solo conoce el usuario. Mientras D_k se define en función de la clave privada, E_k se define mediante un algoritmo o función que no permita, desde el punto de vista computacional, la revelación de D_k. Estos tipos de algoritmos o funciones reciben el nombre de unidireccionales.

Separabilidad escritura/lectura

Este tipo de criptosistemas son muy indicados para la protección de ficheros públicos, puesto que el hecho de poder escribir información sobre el fichero no implica poderla leer y viceversa, porque las claves de escritura y lectura son independientes, pese a que están relacionadas.

En este caso, dos algoritmos diferentes proporcionan el secreto y la autenticidad. Por ejemplo, si el usuario A quiere transmitir el mensaje m al usuario B, ambos conectados al mismo directorio público, solo debe buscar el algoritmo de cifrado E_{k_B} de B en el directorio público y transmitir el mensaje cifrado $c = E_{k_B}(m)$. Cuando el usuario B recibe c tiene que aplicar D_{k_B}, que solo él conoce, y encuentra el mensaje original $D_{k_B}(c) = m$, (ya que $D_{k_B}(c) = D_{k_B}(E_{k_B}(m)) = m$).

Con la utilización de la criptografía de clave pública queda asegurado el secreto, pero no queda protegida la autenticidad, puesto que todos los usuarios pueden conocer E_{k_B}.

Firma digital

En general, se envía la firma por una parte y el mensaje, cifrado o no, por otra parte; según la necesidad de privacidad del mensaje. Así, si A quiere enviar un mensaje firmado a B, enviará m, o $E_{k_B}(m)$ y la firma correspondiente: $s = D_{k_A}(h(m))$. Entonces, B recupera m con su clave privada que compara con el resultado de aplicar el cifrado con la clave pública de A a s. Si ambos resultados coinciden se acepta la autenticación y de lo contrario se rechaza. Incluso, como veremos más adelante, lo que se enviará es el mensaje m, cifrado o no, y la firma de un resumen del mensaje: $h(m)$ (función de *hash*). En tal caso la firma será: $s = D_{k_A} h(m)$ (ver el algoritmo de firma DSA).

Para conseguir la autenticidad tendremos que exigir a las transformaciones E_k y D_k, de cada usuario, que verifiquen que E_k sea la transformación inversa de D_k; es decir, que para todo mensaje m y para todo usuario con clave k, tengamos: $E_k(D_k(m)) = m$,

En este caso el usuario A podrá firmar sus mensajes mediante su transformación secreta D_{k_A}. En efecto, si A quiere enviar el mensaje m a B, autenticando su procedencia, podrá firmar digitalmente este mensaje haciendo $s = D_{k_A}(m)$, donde s será la firma y D_{k_A} la transformación de descifrado del usuario A; que solo él conoce y, por lo tanto, solo él podrá hacer esta operación.

Ahora, una vez firmado el mensaje, A enviará el mensaje cifrado c=$E_{k_B}(s)$. Cuando B reciba c; puede encontrar s, puesto que $s = D_{k_B}(c)$, que será ininteligible para él. Ya que ha sido prevenido por A que le enviaba un mensaje, para descifrar el contenido de s, solo debe buscar en el directorio público el algoritmo E_{k_A} de A. Y en efecto, $E_{k_A}(s) = m$ era el mensaje que pretendía transmitirle A.

2.1. Funciones unidireccionales

La idea fundamental de Diffie y Hellman para la definición de criptosistema con clave pública se basa en la existencia de las funciones unidireccionales (*One way functions*).

Definición 2.1 (Función unidireccional).

Una función f sobre un dominio U se llama unidireccional si $\forall x \in U$, $f(x)$ es fácilmente calculable, mientras que para casi todo $y \in f(U)$, no es computacionalmente factible encontrar $x \in U$, tal que $f(x) = y$.

Observar que la definición no es muy precisa: los términos *fácilmente calculable*, *para casi todo* y *computacionalmente factible* son muy imprecisos, aunque se pueden definir matemáticamente para que tengan un sentido perfectamente preciso.

Definición 2.2 (Función unidireccional con trampilla).

Una familia de funciones invertibles f_k con dominio U_k, con índice k, se llama función unidireccional con trampilla si, dado k, se pueden encontrar algoritmos E_k y D_k que calculen fácilmente $f_k(x)$ y $f_k^{-1}(y)$ $\forall x \in U_k$ y $\forall y \in f(U_k)$; y, sin embargo, para casi todo k y $\forall y \in f(U_k)$, no es computacionalmente eficiente encontrar $f_k^{-1}(y)$, con el único conocimiento de E_k.

Una de las primeras candidatas a función unidireccional fue la del logaritmo discreto, propuesta por los mismos Diffie y Hellman. En efecto, dados un número primo grande, p, y α un elemento primitivo del cuerpo $\mathbb{F}_p$, la función exponencial discreta: $f(x) = \alpha^x \pmod p$, donde $1 < x < p$ es computacionalmente eficiente calcularla.

En cambio, la inversa de la exponencial discreta; el logaritmo discreto $x = log_\alpha(y)$ no es computacionalmente eficiente calcularlo si p–1 tiene un factor primo grande.

Algoritmo de multiplicar y elevar

Para valores de x grandes, podemos usar el método binario de exponenciación (D. E. Knuth (1981). *The Art of Computer Programming. vol. 2 Semi-Numerical Algorithms*. Addisson Wesley). Por ejemplo (véase el algoritmo 3.2 del módulo "Cuerpos finitos" de esta asignatura), para calcular α^{25} se puede realizar de esta forma: $\alpha^{25} = \alpha^{16+8+1} = (((\alpha^2)^2)^2)^2 \cdot ((\alpha^2)^2)^2 \cdot \alpha$.

2.2. Criptosistema RSA

A partir del concepto de función unidireccional, Diffie y Hellman definen la estructura de un criptosistema de clave pública. Sin embargo, no proporcionan ninguna implementación concreta de tal estructura, excepto para el caso del protocolo de distribución de claves privadas que veremos más adelante.

R. L. Rivest, A. Shamir y L. Adleman, del MIT, en su artículo "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Comm. of ACM 21(2); 120-126, February 1978, presentaron un criptosistema de clave pública que cumplía todas las condiciones enumeradas anteriormente (el criptosistema RSA), basado en el teorema de Euler y en la dificultad de factorizar un valor $n = p \cdot q$, donde p y q son primos.

La función unidireccional del RSA es la exponencial discreta: $f_k(x) = x^e \pmod n$; donde $0 < x < n = p \cdot q$ y donde $k = (e,n)$; p y q son dos números primos muy grandes y e cumple $0 < e < \varphi(n)$ y $mcd(e,\varphi(n)) = 1$. El algoritmo E_k para calcular $f_k(x)$ es relativamente fácil; es la exponenciación por el método de multiplicar y elevar mencionado anteriormente. Hacer público este algoritmo requiere divulgar n y e.

Cálculo de $\varphi(n)$

Para valores grandes de p y q, no es computacionalmente eficiente el cálculo de $\varphi(n)$, para quien no conoce los valores de p y q.

Teorema de Euler

El teorema de Euler asegura que la función f_k^{-1} es la inversa de f_k, es decir: $f_k^{-1}(f_k(x)) = x \pmod n$, si x es relativamente primo con n.

La función inversa es:

$$f_k^{-1}(y) = y^d \pmod n$$

donde d es el único $0 < d < n$ tal que $e \cdot d = 1 \pmod{\varphi(n)}$. El algoritmo f_k^{-1} es fácil de calcular para quien conoce la clave (d,n). Pero solo conocerá d, quien conozca $\varphi(n)$, difícilmente calculable para quien no conoce la factorización de n en p y q (esta es la trampilla de la función unidireccional utilizada).

Trampilla

Dado $\varphi(n)$ es fácil generar el par de números e y d que satisfacen la condición de inversos $\pmod{\varphi(n)}$, cuando e o d son relativamente primos con $\varphi(n)$. Es decir, dado e, es fácil calcular d (o viceversa) si conocemos $\varphi(n)$. Sin embargo, si e y n son conocidos, sin revelar $\varphi(n)$, no es computacionalmente eficiente calcular d.

2.2.1. Descripción del criptosistema

El criptosistema RSA consiste en asociar a cada carácter del alfabeto, en qué están escritos los mensajes originales, un valor numérico y entonces cifrar el mensaje por bloques de la misma longitud y con un valor numérico comprendido en un cierto rango.

Supongamos $m \in [2,n-1]$ correspondiente a un cierto bloque a cifrar. El **algoritmo de cifrado** se reduce al cálculo de una exponencial donde la clave es el par de números (e,n):

$$c = E_{(e,n)}(m) = m^e \pmod n$$

La **algoritmo de descifrado**, para poder obtener m a partir de c, consiste también en una exponenciación, donde la clave es ahora otro par de números (d,n):

$$m = D_{(d,n)}(c) = c^d \pmod n$$

La manera de obtener un buen esquema de cifrado y descifrado recae en la posibilidad de obtener $\varphi(n)$. Rivest, Shamir y Adleman sugieren este tratamiento:

1) Encontrar el valor $n = p \cdot q$, donde p y q son dos números primos grandes (en los inicios del RSA ya se sugerían de un centenar de dígitos cada uno).
2) Conociendo p y q, calcular: $\varphi(n) = (p-1) \cdot (q-1)$.
3) Tomar e relativamente primo con $\varphi(n)$.
4) Calcular $d = e^{-1} \pmod{\varphi(n)}$.

Veamos un ejemplo aunque los valores empleados no son los que se podrían usar en la realidad.

Ejemplo 2.1. Supongamos $p = 13$ y $q = 17$. Entonces $n = 13 \cdot 17 = 221$ y $\varphi(n) = 12 \cdot 16 = 192$. Escogiendo $e = 11$ $((e, \varphi(n))=(11, 192)=1)$, calculamos el valor de d, tal que $d \cdot e = 1 \pmod{\varphi(n)}$ y encontramos $d = 35$.* La clave pública será (11,221) y la clave privada será (35,221).

*** Por ejemplo, usando el algoritmo de Euclides extendido, según el algoritmo 1.4 del módulo "Cuerpos finitos".**

Entonces el cifrado será: $c = E_{(11,221)}(m) = m^{11} \pmod{221}$ y el descifrado: $m = D_{(35,221)}(c) = c^{35} \pmod{221}$.

Si suponemos que el conjunto de mensajes originales es $M = \{A,B,...,Y,Z\}$ y la correspondiente asignación numérica es $M_n = \{2,3,...,26,27\}$, y queremos cifrar el mensaje $m = EDI$ o, numéricamente, 060510, haremos sucesivamente:

- $6^{11} \pmod{221} = 141$
- $5^{11} \pmod{221} = 164$
- $10^{11} \pmod{221} = 173$

y nos dará el criptograma $c = 141164173$.

Para el descifrado iremos cogiendo sucesivamente bloques de tres dígitos y haremos:

- $141^{35} \pmod{221} = 6$
- $164^{35} \pmod{221} = 5$
- $173^{35} \pmod{221} = 10$

que nos dará el mensaje original $m = 060510$ o, en caracteres: $m = EDI$.

2.2.2. Firma digital, basada en el RSA

Los algoritmos de cifrado y descifrado del algoritmo RSA son conmutativos; es decir: $D_k(E_k(m) = E_k(D_k(m))$, y por lo tanto, el esquema RSA puede ser utilizado para ambos objetivos de privacidad y autenticidad. En base a esta conmutatividad se puede utilizar el RSA para construir firmas digitales.

Firma digital, basada en el RSA

En general, dado que los valores de m serán muy grandes, se firmará un resumen de m y la firma será: $s = D_{(d_A,n_A)}(h(m))$, donde $h(m)$ es la función resumen (*hash*) de m.
En este caso, en lugar de proceder cómo indica el segundo paso del cifrado, el mensaje m se enviará aparte, cifrado o no.
La función resumen deberá ser conocida, también por el usuario B, para poder proceder a hacer la verificación del segundo paso del descifrado.

En este caso, si suponemos dos usuarios A y B, con claves públicas (e_A,n_A) y (e_B,n_B) y claves privadas (d_A,n_A) y (d_B,n_B), respectivamente, podremos arbitrar un sistema de firma digital como se ha mencionado anteriormente.

Si el usuario A quiere enviar el mensaje m, firmado digitalmente, al usuario B, procederá de la forma siguiente:

Por parte del usuario A:

1) Firmar m con su clave privada: $s = D_{(d_A,n_A)}(m)$

2) Cifrar la firma con la clave pública de B: $c = E_{(e_B,n_B)}(s)$

Por parte de *B*, una vez recibido el criptograma *c*:

1) Descifrar el criptograma *c* con su clave privada: $D_{(d_B,n_B)}(c) = s$

2) Verificar, a partir de la firma *s*, si el criptograma *c* ha sido enviado realmente por *A*: $E_{(e_A,n_A)}(s) = m$

Ejemplo 2.2.

Usuario A:

Sean $p = 29$ y $q = 7$ los valores escogidos por *A*. Entonces $n_A = 29 \cdot 7 = 203$ y $\varphi(n_A) = 28 \cdot 6 = 168$.

Supongamos que *A* escoge: ($e_A = 19$,$n_A = 203$) como clave pública, entonces ($d_A = 115$,$n_A = 203$) será su clave privada.

Usuario B:

Sean $p = 13$ y $q = 17$ los valores escogido por *B*. Entonces $n_B = 13 \cdot 17 = 221$ y $\varphi(n_B) = 12 \cdot 16 = 192$.

Supongamos que *B* escoge: ($e_B = 11$,$n_B = 221$) como clave pública, entonces ($d_B = 35$,$n_B = 221$) será su clave privada.

Si suponemos que el conjunto de mensajes originales es $M = \{A,B,...,Y,Z\}$ y la correspondiente asignación numérica es $M_n = \{2,3,...,26,27\}$, y queremos firmar digitalmente el texto original EDI o, numéricamente, 060510, haremos sucesivamente:

- $6^{115} \pmod{203} = 13$
- $5^{115} \pmod{203} = 96$
- $10^{115} \pmod{203} = 101$

y nos dará la firma $s = 013096101$.

Para el cifrado, iremos cogiendo sucesivamente bloques de tres dígitos de la firma *s* y haremos:

- $013^{11} \pmod{221} = 208$
- $096^{11} \pmod{221} = 216$
- $101^{11} \pmod{221} = 186$

que nos dará el criptograma $c = 208216186$.

Cuando el usuario *B* ha recibido *c*, lo divide en bloques de tres dígitos y los descifra con su clave privada:

- $208^{35} \pmod{221} = 013$
- $216^{35} \pmod{221} = 096$
- $186^{35} \pmod{221} = 101$

el mensaje *s* que recupera es ininteligible (13 equivale al carácter *L*, 96 y 101 no tienen equivalencia dentro del conjunto de mensajes originales). Ahora, el usuario *B* debe comprobar que 013096101 es la firma digital del usuario *A*, para esto debe cifrar el resultado obtenido de la anterior operación con la clave pública de *A*.

- $013^{19} \pmod{203} = 6$
- $096^{19} \pmod{203} = 5$
- $101^{19} \pmod{203} = 10$

que nos dará el texto original 060510 o, en caracteres, EDI.

Nota

El usuario *B* acepta EDI como el mensaje que le ha enviado A, sencillamente porque para él es inteligible; pero esto en muchos casos no sería suficiente. Tal y como veremos, en otros casos, la estrategia de firma será diferente.

2.3. Criptosistema ElGamal

El criptosistema de clave pública ElGamal se basa en la función unidireccional exponencial discreta. Este criptosistema ha servido de base para la definición de un algoritmo de firma alternativo al RSA, el DSA.

Ver también

El algoritmo DSA se estudia en el subapartado 2.4 de este módulo.

2.3.1. Descripción del criptosistema ElGamal

Si fijamos un cuerpo finito $\mathbb{F}_p$ y un elemento primitivo α, supondremos que el mensaje que queremos cifrar se corresponde con un elemento $m \in \mathbb{F}_p$. A partir de aquí, cada usuario U escoge al azar un entero $r_U \in [2,p-1]$ que será su clave privada. Su clave pública será $y_U = \alpha^{r_U} \pmod p \in \mathbb{F}_p$.

Entonces, si un usuario A quiere transmitir al usuario B el criptograma correspondiente al mensaje original $m \in \mathbb{F}_p$, deberá hacer las siguientes operaciones dentro de $\mathbb{F}_p$:

Algoritmo 2.3 (Algoritmo ElGamal):

1) Escoger al azar un entero k y calcular $K = \alpha^k \pmod p$

2) Cifrar m como: $c = E_{y_B}(m) = m \cdot (y_B)^k \pmod p$

3) Transmitir el par de números (K,c)

Fortaleza

Cualquier enemigo que quiera calcular K^{r_B}, a partir del conocimiento de K y y_B, pero sin conocer r_B, deberá calcular previamente el logaritmo discreto $log_\alpha(y_B)$. En la dificultad de este cálculo se basa la fortaleza del criptosistema ElGamal.

Entonces, el usuario B podrá recuperar m a partir del par de números recibidos, haciendo las siguientes operaciones dentro de $\mathbb{F}_p$:

1) Calcular $\beta = K^{r_B} \pmod p$

2) Calcular $c/\beta \pmod p = m$

Nota

El resultado de esta operación ha de ser, efectivamente, igual a m, ya que $K^{r_B} \pmod p = (\alpha^{r_B})^k \pmod p = (y_B)^k \pmod p$.

A continuación veremos un ejemplo de utilización del algoritmo, para poder seguirlo mejor, aunque los valores del ejemplo son pequeños comparados con los que se tendrían que usar en la realidad.

Ejemplo 2.3.

Supongamos el cuerpo finito $\mathbb{F}_{23}$ y sea $\alpha = 5$ el elemento primitivo elegido*. Si $r_A = 13$, la clave pública del usuario A será $y_A = 5^{13} \pmod{23} = 21$. Si $r_B = 17$, la clave pública del usuario B será $y_B = 5^{17} \pmod{23} = 15$.

Si el usuario A quiere transmitir al usuario B el mensaje $m = 18$ efectuará los siguientes cálculos:

1) Tomará al azar un entero, por ejemplo $k = 7$, y calculará $K = 5^7 \pmod{23} = 17$
2) Cifrará $m = 18$, sabiendo que $y_B = 15$, como: $c = E_{15}(18) = 18 \cdot 15^7 \pmod{23} = 14$.
3) Transmitirá el par de números (17,14).

*** Tal como se ha visto en el módulo "Cuerpos finitos".**

Entonces el usuario B podrá recuperar m a partir del par de números recibidos, (K,c), haciendo las siguientes operaciones dentro de $\mathbb{F}_{23}$:

1) Calculará $\beta = 17^{17} \pmod{23} = 11$
2) Calculará $c/\beta = 14/11 \pmod{23} = 18$; $(11^{-1} \pmod{23} = 21)$.

Este resultado coincide con el valor del mensaje original, $m = 18$.

2.3.2. Firma digital, basada en el ElGamal

El criptosistema ElGamal no cumple la condición de conmutatividad que hemos visto en el subapartado de firma digital basada en el RSA. O sea, no se cumple $E_k(D_k(m)) = m$. Sin embargo, se puede adaptar el sistema para la autenticación mediante otro tipo de firma digital.

Supongamos que el mensaje a firmar por un usuario A, para ser trasmitido a un usuario B, sea $m \in \mathbb{F}_p$. Para la firma, el usuario A debe realizar las siguientes operaciones en $\mathbb{F}_p$:

1) Elegir un entero k tal que $mcd(k,p-1) = 1$ y calcular $K = \alpha^k \pmod{p}$.

2) Encontrar un entero s tal que: $m = r_A \cdot K + k \cdot s \pmod{p-1}$, donde r_A y k son valores que solo conoce el usuario A; por lo tanto solo él será capaz de calcular la firma s.

3) La firma digital es el par de números (K,s). Entonces transmitirá (K,s,m), aunque, opcionalmente, quizá pueda querer también cifrar el mensaje m, $c = E_{y_B}(m)$, y enviar (K,s,c).

Firma ElGamal

Se puede calcular, de forma directa, haciendo $s = (m - r_A \cdot K) \cdot k^{-1} \pmod{q-1}$.

Para la validación de la firma del mensaje m, el usuario B debe comprobar que se cumple la siguiente igualdad:

$$\alpha^m = (y_A)^K \cdot K^s \pmod{p}$$

donde y_A es la clave pública del usuario A y, por lo tanto, disponible por el usuario B.

En caso de que la igualdad se cumpla, el usuario B aceptará como auténtico el mensaje m que le ha enviado el usuario A. De lo contrario, el mensaje m no será validado.

Verificación

Efectivamente: $(y_A)^K = (\alpha^K)^{r_A}$
$K^s = (\alpha^k)^s = \alpha^{(m-r_A \cdot K)} =$
$\alpha^m \cdot (\alpha^K)^{-r_A}$
El producto de ambas igualdades resulta ser, α^m.

Ejemplo 2.4. Suponemos que continuamos con las mismas hipótesis del ejemplo anterior: $\mathbb{F}_{23}$, $\alpha = 5$, $r_A = 13$ y $r_B = 17$ La clave pública del usuario A es $y_A = 21$ y la clave pública del usuario B es $y_B = 15$.

Si A quiere transmitir el mensaje $m = 18$ al usuario B de forma secreta y autenticada, hará los siguientes cálculos:

1) Escogerá al azar un entero, por ejemplo $k = 7$ ($mcd(7,22) = 1$), y calculará $K = 5^7 \pmod{23} = 17$.

2) Calculará s tal que $m = r_A \cdot K + k \cdot s \pmod{22}$. Es decir:

$$s = (m - r_A \cdot K) \cdot k^{-1} \pmod{22} = (18 - 13 \cdot 17) \cdot 7^{-1} \pmod{22} = 15; (7^{-1} \pmod{22} = 19).$$

3) Para transmitir el mensaje cifrado (en el ejemplo anterior $c = 14$) y firmado, el usuario A envía: $(K,s,c) = (17,15,14)$.

Entonces, el usuario B puede validar la transmisión a partir de m (en el ejemplo anterior, a partir de c, había reencontrado $m = 18$) y el par de números (K,s).

Efectuando el cálculo $(y_A)^K \cdot K^s \pmod{p} = 21^{17} \cdot 17^{15} \pmod{23} = 6$, que coincide con $\alpha^m \pmod{p} = 5^{18} \pmod{23} = 6$.

Por lo tanto, el usuario B daría por válida la firma s del mensaje m.

2.4. Algoritmo DSA como alternativa a la firma digital RSA

En 1991, el NIST (National Institute of Standards and Technology) hizo la propuesta del algoritmo DSA (*Digital Signature Algorithm*) como un estándar de firma digital DSS (*Digital Signature Standard*).

Este algoritmo DSA fue desarrollado por la NSA (National Security Agency), a partir de la firma digital de ElGamal, pero con el propósito añadido de reducir la longitud. Este algoritmo contiene los parámetros siguientes:

- p, un número primo de $2 \cdot L$ bits, donde L es un múltiplo de 64 y está comprendido entre 512 y 1024 bits.
- q, un factor primo de $p - 1$ de unos 160 bits. Sea $n = (p - 1)/q$
- α, tal que $\alpha = g^n \pmod{p}$, donde g es un número menor que $p - 1$ y de forma que $\alpha \pmod{p} > 1$.
- x, un número cualquiera menor que q.
- y, tal que $y = \alpha^x \pmod{p}$.
- $h(.)$, una función unidireccional *hash*.

Función unidireccional *hash*

Una función *hash*, h, viene definida por una serie de operaciones que transforman un mensaje m, de longitud variable, en una secuencia de pocos bits, $h(m)$, de longitud fija, como veremos en el próximo subapartado.

Los números p, q y α son públicos para todos los usuarios de la red, mientras que x es la clave privada e y es la clave pública.

Sea m el mensaje que A quiere enviar a B, firmado para que B pueda autenticarlo.

El usuario A, del cual se suponen conocidos los parámetros anteriores, excepto x, deberá realizar las siguientes operaciones:

1) Escoger un número aleatorio k, menor que q.

2) Generar dos valores r y s, tales que:

$r = (\alpha^k \pmod p) \pmod q$

$s = ((h(m) + x \cdot r) \cdot k^{-1}) \pmod q$

3) Enviar el mensaje m y su firma digital (r,s).

El usuario B, al recibir el mensaje m, y su firma digital correspondiente (r,s), podrá hacer el siguiente proceso de autenticación:

1) Seleccionar, en el directorio público, los parámetros de A: p,q,α y $h(.)$

2) Calcular: $w = s^{-1} \pmod q$

$u_1 = (h(m) \cdot w) \pmod q$

$u_2 = (r \cdot w) \pmod q$

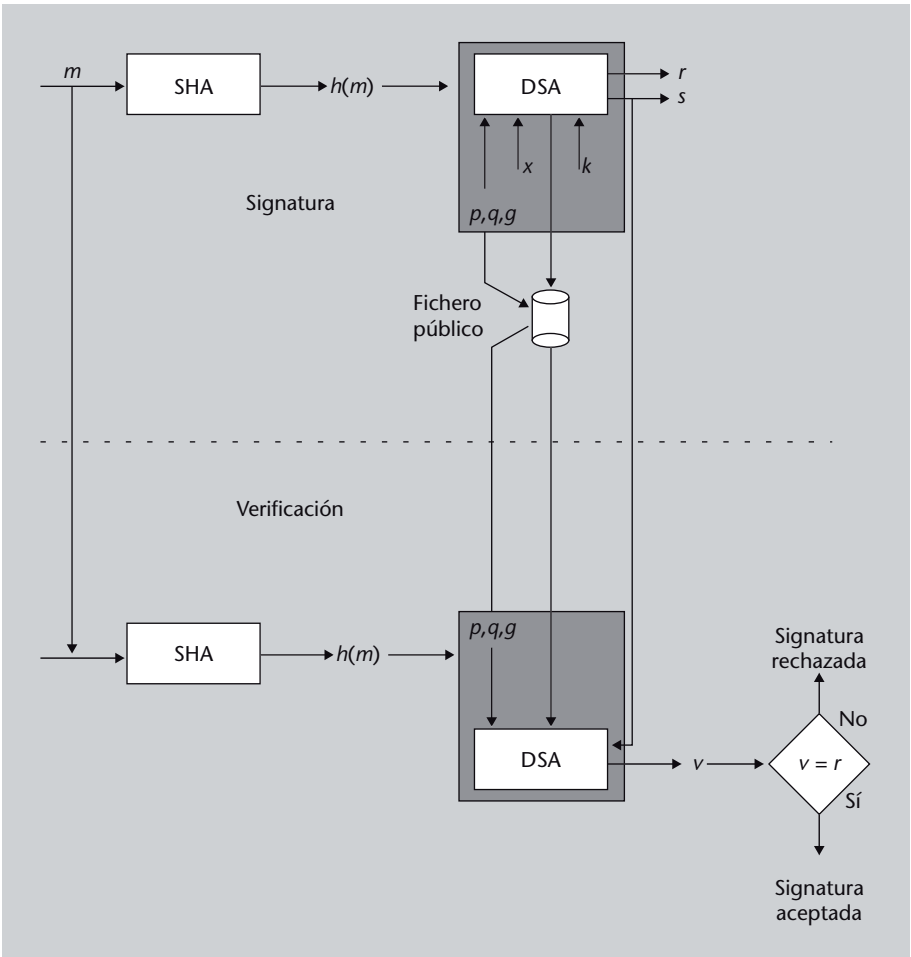
3) Calcular: $v = ((\alpha^{u_1} \cdot y^{u_2}) \pmod p) \pmod q$

4) Autenticar: Si $v = r$ entonces la firma digital (r,s) de A, es aceptada por B.

Verificación de firma

Por simplicidad, y sin perder rigor en la prueba, obviaremos los cálculos modulares.
$v = \alpha^{u_1} \cdot y^{u_2} = (\alpha^{h(m)\cdot s^{-1}}) \cdot (\alpha^x)^{r\cdot s^{-1}} = \alpha^{(h(m)+x\cdot r)\cdot s^{-1}}$,
y según la definición de s, este resultado es, efectivamente, $\alpha^k = r$.

El siguiente diagrama representa el proceso de esta firma digital:



Ejemplo 2.5.

Suponemos que el conjunto de mensajes originales es $M = \{A,B,...,Y,Z\}$ y la correspondiente asignación numérica $M_n = \{2,3,...,26,27\}$. Si el usuario A quiere firmar digitalmente el texto original EDI o, numéricamente, 060510, teniendo en cuenta los parámetros del criptosistema, hará las siguientes operaciones:

Parámetros públicos:
$p = 29, q = 7$ $(n = (p-1)/q = 28/7 = 4)$
$\alpha = 5^4 \pmod{29} = 16$; $(g = 5)$
Clave privada: $x = 3$
Clave pública $y = 16^3 \pmod{29} = 7$
Parámetro aleatorio secreto: $k = 6$

Entonces, el usuario A calcula (suponiendo la función ficticia de *hash* anterior):
$h(m) = h(06,05,10) = 60 \pmod{29} = 2$
$r = (16^6 \pmod{29}) \pmod 7 = 20 \pmod 7 = 6$
$s = (2 + 3 \cdot 6) \cdot 1/6 \pmod 7 = 6 \cdot 1/6 \pmod 7 = 1$

Por lo tanto, $DSA(EDI) = DSA(060510) = (6,1)$ y enviará al usuario B: $(EDI,6,1)$.

El usuario B, a la recepción de esta tripla, realiza la verificación, tras seleccionar los parámetros p,q,g y $h(.)$:

Calcula:

$h(EDI) = 2$, puesto que $(h(06,05,10) = 6 \cdot 10 \pmod{29})$
$w = 1^{-1} \pmod 7 = 1$
$u_1 = 2 \cdot 1 \pmod 7 = 2$
$u_2 = 6 \cdot 1 \pmod 7 = 6$

Verifica que $v = (16^2 \cdot 7^6 \pmod{29}) \pmod 7 = 20 \pmod 7 = 6 (= r)$ y por lo tanto, valida la firma.

Función *hash* ficticia

Por simplicidad, en el ejemplo hemos escogido como función *hash*, ficticia, el producto de los valores numéricos de los caracteres impares (mod 29). Esta propuesta no tiene nada que ver con una función de *hash* real, como las que detallaremos en el próximo apartado.

Criticidad de la función de *hash*

Si suponemos que enviamos un texto diferente con la misma firma (EDJ, 6, 1), el usuario B haría los siguientes cálculos:
$h(EDJ) = 8$;
$(h(06,05,11) = 6 \cdot 11 \pmod{29} = 8)$
$w = 1^{-1} \pmod 7 = 1$
$u_1 = 8 \cdot 1 \pmod 7 = 1$
$u_2 = 6 \cdot 1 \pmod 7 = 6$
$v = (16^1 \cdot 7^6 \pmod{29}) \pmod 7 = 23 \pmod 7 = 2 (\neq r = 6)$, con lo cual no se autenticaría la firma.
Sin embargo, hay que observar que si el mensaje firmado hubiera sido BUS, como que $h(BUS) = h(03,22,20) = 3 \cdot 20 \pmod{29} = 2$, derivaría la misma firma. Con lo cual, se puede ver que la elección de la función de *hash* es crítica respecto a este algoritmo.

2.5. Funciones *hash*: MD5 y SHA-1

El propósito de las funciones *hash* es el de proporcionar una huella, una impronta, de pocos bits, de un fichero, mensaje o cualquier otro bloque de datos, por grande que sea, el cual se quiere autenticar.

Una función *hash*, h, debe tener las siguientes propiedades:

1) Se debe poder aplicar la función h a un bloque de datos de cualquier longitud.

2) El resultado de la función h debe ser de longitud fija, de pocos bits.

3) Debe ser computacionalmente eficiente calcular $h(m)$ para cualquier m, tanto en implementaciones de hardware como de software.

4) Para un bloque determinado x, no debe ser computacionalmente eficiente encontrar un mensaje m tal que $h(m) = x$.

5) Para un mensaje determinado m, no debe ser computacionalmente eficiente encontrar otro mensaje $m' \neq m$ tal que $h(m) = h(m')$.

6) No debe ser computacionalmente eficiente encontrar un par (m,m') tal que $h(m) = h(m')$.

Enlace de interés

Se puede encontrar un aplicativo de simulación de las funciones *hash* MD5 y SHA-1, de uso libre, en la dirección: www.criptored.upm.es.

Utilidad de las propiedades

Las primeras tres propiedades son tres requisitos para llevar a la práctica las funciones *hash* y la autenticación de mensajes.

La cuarta propiedad sirve para asegurar que no debe ser computacionalmente eficiente encontrar la inversa de las funciones *hash* (funciones unidireccionales, *one-way functions*).

La quinta y sexta propiedad garantizan que no debe ser posible encontrar un mensaje alternativo con la misma secuencia *hash* que el original.

A continuación, pasaremos a examinar los dos algoritmos que implementan las funciones *hash* más utilizadas actualmente: el algoritmo *message-digest*, MD5, y el algoritmo *secure hash*, SHA.

2.5.1. El algoritmo MD5

El *message-digest* MD5 lo desarrolló Ron Rivest y consiste en un algoritmo que tiene como entrada un mensaje de una longitud arbitraria y produce una salida de 128 bits (el *message digest*).

Eficiencia del MD5

Rivest muestra que, con el MD5, la dificultad de encontrar dos mensajes con la misma secuencia hash es del orden de 2^{64} operaciones, y, por otra parte, que la dificultad de encontrar un mensaje con una secuencia *hash* determinada es del orden de 2^{128} operaciones.

El mensaje de entrada se procesa en bloques de 512 bits, siguiendo los pasos:

- **Paso 1: Añadir bits de *padding***. Se insertan bits en el mensaje para que su longitud final sea congruente con 448 $\pmod{512}$. Esta secuencia de bits de *padding* consiste en un 1 seguido de la cantidad necesaria de 0's.

Efecto del *padding*

Los bits de *padding* se añaden siempre, incluso si el mensaje ya tiene la longitud deseada. Por ejemplo, si el mensaje tiene 448 bits de longitud, entonces se añaden 512 bits de padding para llegar a una longitud de 960 bits. Esto quiere decir que añadimos una cantidad de bits de padding que oscilará entre 1 y 512.

- **Paso 2: Añadir la longitud**. Al resultado del paso anterior le añadimos la representación en 64 bits de la longitud del mensaje antes de añadir los bits de *padding*. Por lo tanto, este campo contiene la longitud del mensaje original $\pmod{2^{64}}$. Tras los dos primeros pasos tenemos un mensaje con una longitud en bits igual a un múltiplo de 512. El mensaje se divide en bloques de 512 bits, denominados $Y_0, Y_1 \cdots Y_{L-1}$; por lo tanto, podemos expresar la longitud del mensaje como $L \cdot 512$ bits, o también ver el mensaje como un múltiplo de 16 palabras de 32 bits.

- **Paso 3: Iniciar el buffer MD**. Para guardar tanto los resultados intermedios como el resultado final, se utiliza un *buffer* de 128 bits, el cual se representa por cuatro palabras de 32 bits, A, B, C, D, que se inician con los siguientes valores hexadecimales:

$$A = 01234567$$

$$B = 89ABCDEF$$

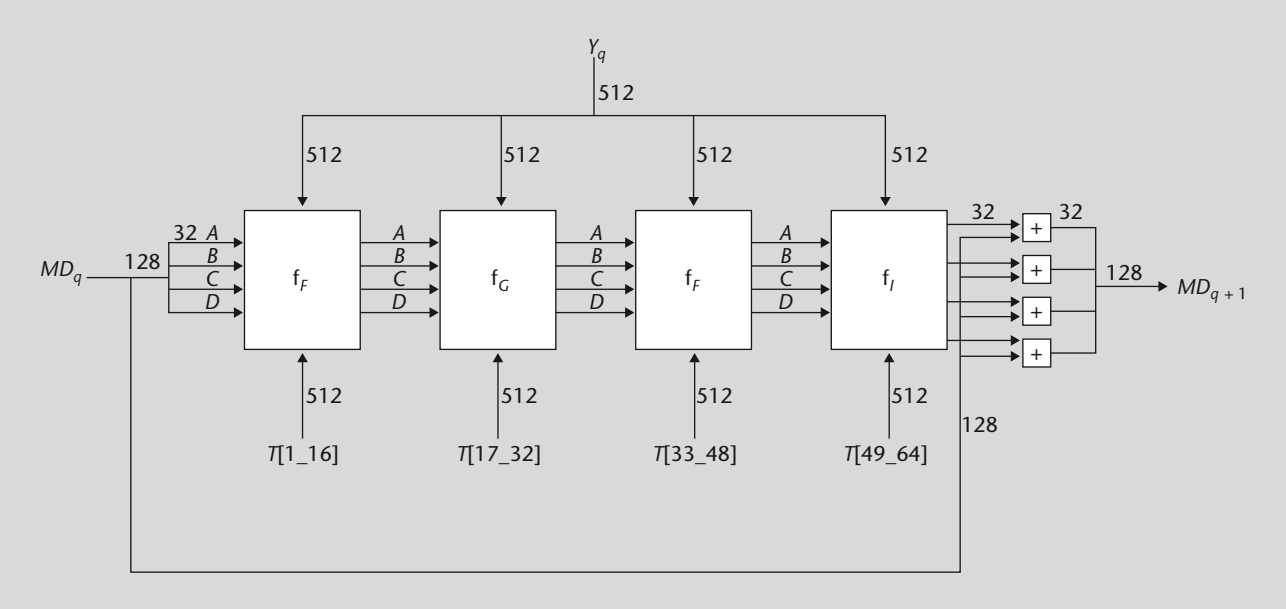
$$C = FEDCBA98$$

$$D = 76543210$$

- **Paso 4: Procesamiento del mensaje en bloques de 512 bits (16 palabras de 32 bits)**. La parte central del algoritmo es un módulo que tiene cuatro etapas de funcionamiento.

  Las cuatro etapas tienen una estructura similar, pero utilizan funciones lógicas primitivas diferentes.

  En la figura, se ha etiquetado cada etapa con f_F, f_G, f_H, f_Y para indicar que todas tienen la misma estructura, f, pero con una función primitiva diferente: F,G,H,I.



  Cada etapa tiene como entrada el respectivo bloque de 512 bits (Y_q) y el valor del *buffer* A,B,C,D de 128 bits y actualiza el contenido del *buffer*. En cada etapa también se utiliza una cuarta parte de los 64 elementos de una tabla T[1..,64] que proporciona un conjunto seudo-aleatorio de secuencias de 32 bits, que sirven para eliminar cualquier regularidad en los datos de entrada. Por lo tanto, el proceso del bloque Y_q consiste en coger como entrada el mismo Y_q y el resultado (*message digest*) intermedio correspondiente MD_q para producir el MD_{q+1}. Las sumas que se hacen al final de las cuatro etapas son sumas (mod 2^{32}).

- **Paso 5: Salida**. Tras procesar los L bloques de 512 bits, la salida MD_{L-1} del L-ésimo bloque de proceso es la secuencia *hash* de 128 bits.

2.5.2. El algoritmo SHA-1

El algoritmo SHA-1 fue desarrollado por el NIST y publicado como estándar de procesamiento de información federal FIPS (*federal information process standard*, PUB 180) en el año 1993.

El algoritmo tiene como entrada un mensaje de longitud menor que 2^{64} bits y produce una salida de 160 bits (el *message digest*). El mensaje de entrada se procesa en bloques de 512 bits, siguiendo los pasos:

- **Paso 1: Añadir bits de Padding y Paso 2: Añadir la longitud**. Igual que en el algoritmo MD5.

- **Paso 3: Iniciar el *buffer* SHA**. Para guardar tanto los resultados intermedios como el resultado final, se utiliza un *buffer* de 160 bits. Podemos representar el *buffer* como cinco palabras de 32 bits, A,B,C,D,E, que se inician con los siguientes valores hexadecimales:

$$A = 67452301$$
$$B = EFCDAB89$$
$$C = 98BADCFE$$
$$D = 10325476$$
$$E = C3D2E1F0$$

- **Paso 4: Procesamiento del mensaje en bloques de 512 bits (16 palabras de 32 bits)**. La parte central del algoritmo es un módulo que tiene 80 etapas de procesamiento. La lógica de este módulo es la siguiente:

  Cada etapa tiene como entrada el actual bloque de 512 bits, Y_q, y los 160 bits del *buffer ABCDE* y, como resultado, actualiza el contenido del *buffer*.

  En cada etapa se hace uso de la suma por una constante K_t. De hecho, solo se utilizan cuatro constantes diferentes. Los valores hexadecimales son los siguientes:

  $0 \leq t \leq 19$, entonces $K_t = 5A827999$
  $20 \leq t \leq 39$, entonces $K_t = 6ED9EBA1$
  $40 \leq t \leq 59$, entonces $K_t = 8F1BBCDC$
  $60 \leq t \leq 79$, entonces $K_t = CAN62C1D6$

  Es decir, el algoritmo SHA para procesar un bloque Y_q de 512 bits, toma como entrada el bloque Y_q y el valor intermedio en aquel momento de la secuencia *hash* SHA_q que se toma del *buffer ABCDE*. Después, el resultado de las 80 etapas se suma a SHA_q, dando SHA_{q+1} que se coloca en el *buffer ABCDE*. Esta última suma se hace con dos sumandos de 160 bits cada uno, la operación se hace de forma independiente para cada una de las cinco palabras de 32 bits que tiene cada sumando (suma módulo 2^{32}).

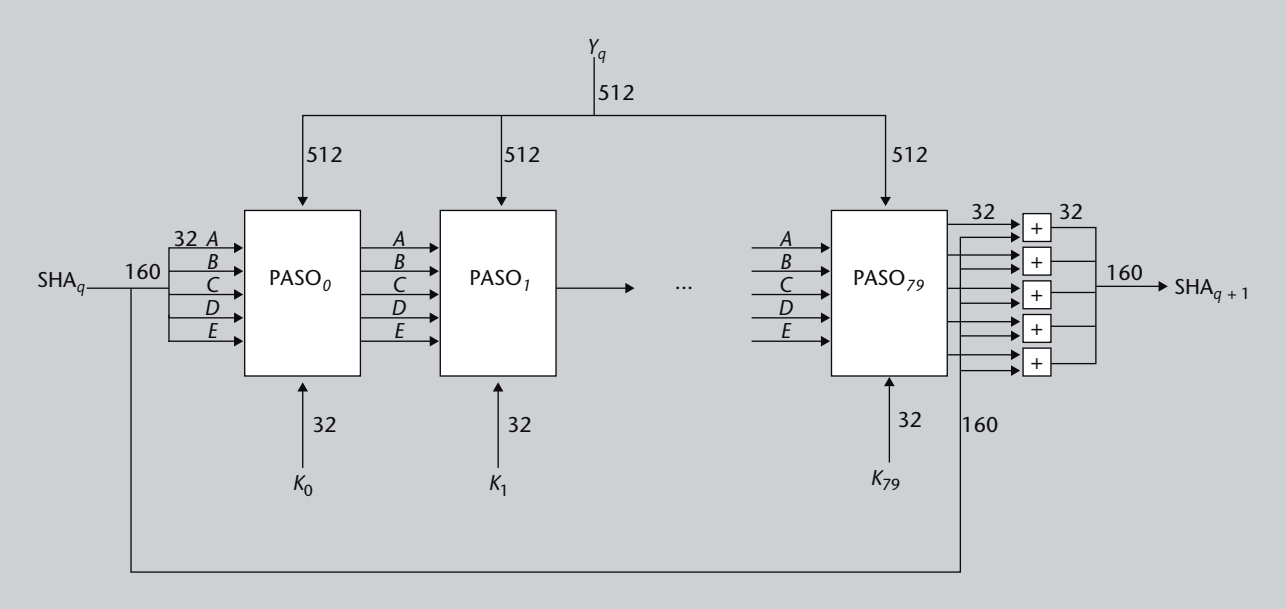
- **Paso 5: Salida**. Tras procesar los L bloques de 512 bits, la salida SHA_{L-1} del L-ésimo bloque de proceso es la secuencia *hash* de 160 bits.

Comparación entre SHA y MD5

Los algoritmos MD5 y SHA son bastante similares, ya que ambos derivan del mismo algoritmo MD3.
La diferencia más obvia, y la más importante, es que la secuencia resultante de aplicar el SHA es 32 bits más larga que la del MD5. Entonces el SHA es un algoritmo más fuerte, desde el punto de vista del criptoanálisis, que el MD5. En cambio, el SHA consta de 80 pasos y el MD5 solo de 64, cosa que hace que el SHA se ejecute del orden de un 25 % más lento. Debemos notar que los dos algoritmos tienen una gran cantidad de sumas módulo 2^{32}, por lo tanto ambos funcionan muy bien en arquitecturas de 32 bits.

SHA-2 y SHA-3

Actualmente, se utiliza la variante SHA-2, desarrollada en el 2005 a partir del la SHA-1, donde la salida puede ser de 224, 256, 384 o 512 bits, para aumentar la dificultad de ser roto. Además, está en marcha un concurso público para diseñar el nuevo estándar SHA-3, que seguramente se hará público durante 2012.



2.6. Infraestructura de clave pública: PKI

En el ámbito de las comunicaciones abiertas es imprescindible garantizar la identidad de los usuarios, además de los servicios de seguridad: confidencialidad, integridad, autenticación y no repudio.

Para que estas operaciones se puedan realizar de forma fiable, se deben cumplir dos condiciones:

- Que la clave privada se guarde de forma segura y no se desvele a nadie. Para conseguir esto, la clave privada se almacena en un soporte físico imposible de duplicar, como una tarjeta inteligente. Además, para acceder al contenido de la tarjeta se necesita un número personal que solamente el propietario legítimo conoce.

- Que se pueda determinar a qué persona pertenece una clave pública. De esta forma se puede saber, por ejemplo, quién ha llevado a cabo la firma electrónica de un documento.

Lectura recomendada

Para hacer más comprensible este subapartado es recomendable el módulo 7 del libro de Criptografía de J. Domingo, J. Herrera y H. Rifà-Pous de los estudios de Informática y Multimedia de la UOC.

Para dar cumplimiento a estas dos condiciones, se utiliza el certificado electrónico, emitido por una autoridad de certificación (CA). El apoyo tecnológico del certificado electrónico es la criptografía de clave pública. Así, puede verse un certificado electrónico como un documento electrónico que asocia una clave pública con su propietario.

Por esto, el certificado digital contendrá la clave pública junto con datos de carácter personal del poseedor de la clave (nombre, DNI...). Normalmente contiene más información (fechas de validez y otras), así como también se refiere

al ámbito de utilización del certificado, lo que se conoce como política de certificación. Por ejemplo, si es un certificado de uso personal o nos acredita para actuar en una empresa.

Al realizar una firma electrónica se suele adjuntar el certificado electrónico del firmante de forma que se puede extraer su clave pública para verificar la firma y a la vez comprobar la identidad del firmante.

Una infraestructura de clave pública (PKI, *public key infrastructure*) es una estructura de sistemas informáticos, procedimientos de operación, protocolos, políticas de certificación, repositorios de información, estándares, declaraciones de prácticas y recursos humanos, la finalidad de los cuales es ofrecer a los usuarios una plataforma para la gestión de la identidad digital.

Una PKI dispone de los elementos y de la arquitectura necesarios para integrar todos los procedimientos de solicitud de certificados, verificación de identidades, generación de claves, almacenamiento y publicación de certificados electrónicos, renovación, revocación, etc.

Las infraestructuras de clave pública se fundamentan en la interacción de diversos subsistemas, de los cuales destacan los siguientes:

- **Autoridad de certificación, CA**. Una autoridad de certificación (**CA: *certificate authority***), es una entidad de confianza, cuya finalidad es emitir, renovar y revocar certificados electrónicos. Las autoridades de certificación constituyen el núcleo de las infraestructuras de clave pública, que permiten utilizar los **certificados electrónicos** con total seguridad.

  Ejemplos de autoridad de certificación

  En la actualidad, un usuario puede escoger entre múltiples CA para conseguir un certificado electrónico, pero las más utilizadas son; a nivel internacional Verisign, a nivel estatal FNMT y en el ámbito catalán CATCert.

  Verisign* es una de las empresas de mayor reputación internacional y prestigio en el mundo de la certificación digital y la seguridad de la información. Aunque su abanico de servicios es muy amplio (soluciones comerciales para comercio electrónico, servidores seguros, tarjetas inteligentes, servidores de nombres de dominio, consultoría,...), el más conocido es el de CA para la expedición de certificados electrónicos, ampliamente utilizados en Internet.

  * http://www.verisign.com

  Fábrica Nacional de Moneda y Timbre (FNMT)**, es un organismo público nacional español que depende del Ministerio de Economía que tiene establecida una arquitectura de certificación, CERES, para autenticar y garantizar la confidencialidad de las comunicaciones entre ciudadanos, empresas u otras instituciones y administraciones públicas a través de redes abiertas de comunicación.

  ** http://www.fnmt.es

  CATCert*** es la Agencia Catalana de Certificación que emite y gestiona la idCAT que es un certificado electrónico que garantiza la identidad de las personas en Internet y permite operar con diferentes administraciones.

  *** http://www.catcert.cat

- **Autoridades de registro, RA**. Una autoridad de registro (**RA, *registration authority***) es una entidad encargada de llevar a cabo los procesos de veri-

ficación de identidad, solicitud y distribución de certificados electrónicos. Normalmente, en una PKI, los usuarios finales no interactúan directamente sobre la CA, sino que canalizan sus operaciones a través de una o varias RA.

Sin embargo, estos subsistemas no pueden expedir certificados electrónicos por sí mismos.

- **Certificados electrónicos**. Un certificado electrónico es un archivo o documento electrónico expedido y firmado por una CA en el cual se vincula una identidad a una clave pública, ligado, a su vez, a la correspondiente clave privada.

  Para obtener un certificado electrónico, el usuario se dirige a una RA (autoridad de registro); ésta verifica la identidad del usuario y pide a la CA que expida el certificado.

- **Directorio *lightweight directory access protocol*, LDAP**. Su finalidad, dentro de una PKI, es la de mantener un registro de usuarios y actuar como almacén para los certificados electrónicos y la lista de certificados revocados (CRL), que veremos más adelante. El protocolo LDAP es una versión simplificada del protocolo X.500 que especifica tanto el modelo de información como los mecanismos de acceso a la misma.

2.6.1. Sistemas gestores de certificados electrónicos: la recomendación X.509

El auge de la certificación electrónica nace de la consolidación internacional del protocolo estándar X.509. A partir de este momento, no solo aparecen en el mercado multitud de aplicaciones que aprovechan los servicios de la identidad digital, sino también un gran número de paquetes de software que implementan las funciones básicas de una PKI. Estos paquetes reciben el nombre de sistemas gestores de certificados electrónicos.

Las soluciones de implementación para los sistemas gestores de certificados electrónicos pueden clasificarse en tres categorías: Las integradas en el sistema operativo, las libres (de código abierto) y las comerciales.

La recomendación X.509 de la ITU-T forma parte de la serie de recomendaciones X.500, la finalidad de las cuales es definir un servicio de directorio. Por directorio se entiende un servidor o conjunto distribuido de servidores que gestionan una base de datos de información sobre usuarios. En la actualidad, hablar de X.509 y certificados electrónicos es hablar del mismo concepto. Actualmente, este estándar de certificación electrónica se utiliza en protocolos de Internet de uso tan extendido como SSL, SMIME (*Secure Multipurpose Internet Mail Extensiones*) y IPSec (*IP Security*).

2.6.2. Listas de certificados revocados: CRL

Una lista de certificados revocados (CRL, *Certificate revocation list*), es un documento electrónico expedido y firmado por una CA, en el cual se incluyen los números de serie de todos aquellos certificados que, sin haber expirado, han sido revocados por algún motivo.

Al recibir un certificado electrónico, el usuario debe consultar la CRL de la CA que firma el certificado para verificar la validez del mismo. El protocolo OCSP (*online certificate status protocol*) permite realizar consultas en tiempo real sobre la base de datos de certificados revocados de una CA. Algunos navegadores web ya incluyen apoyo para OCSP.

La validación de certificados en tiempo real es imprescindible para el desarrollo del comercio electrónico.

La recomendación X.509 define un formato estándar para las listas de certificados revocados, de forma análoga a la estructura sugerida para los certificados electrónicos.

3. Criptografía cuántica y post-cuántica

De manera muy resumida podríamos decir que, hoy en día, la criptografía de clave pública se utiliza para la distribución de claves privadas, para la firma digital y otros protocolos de autenticación y, en cambio, la criptografía de clave privada se utiliza para conseguir la privacidad de los datos. Los algoritmos de clave pública más utilizados son RSA, DSA y ECDSA (el algoritmo DSA utilizando curvas elípticas). ¿Qué pasaría si en pocos años alguien anunciara la construcción de un gran ordenador cuántico?

Dado que la criptografía de clave pública se basa en la factorización o en el problema del logaritmo discreto o del logaritmo elíptico y que no existe el algoritmo de Shor para los ordenadores clásicos, parece ser que con el advenimiento de la computación cuántica la seguridad basada en RSA, DSA y ECDSA se verá comprometida. Aún así, no podemos decir que la criptografía está bajo sospecha de desaparición con el advenimiento de la computación cuántica. Hay todo un campo criptográfico post-cuántico que podrá resistir los grandes ordenadores clásicos y a los ordenadores cuánticos.

Algoritmo de Shor

En 1994, Shor descubrió un algoritmo capaz de factorizar un número producto de dos primos con una complejidad polinomial sobre un hipotético ordenador cuántico. La base del algoritmo es una transformada discreta de Fourier que en un ordenador cuántico, y de manera probabilística, da el resultado. Se han hecho pruebas reales de este algoritmo con prototipos de ordenador cuántico y han funcionado. En el 2001, en *IBM's Almaden Research Center*, usando un prototipo de ordenador cuántico que funcionaba con registros de 7 qubits basados en NMR (*nuclear magnetic resonance*) se consiguó factorizar el número 15 = 3·5.

Algunos de estos sistemas de cifrado que se cree que son, y serán, resistentes a la computación clásica y cuántica son:

- **Criptografía basada en funciones *hash***
- **Criptografía basada en la teoría de códigos**
- **Criptografía basada en la combinatoria**
- **Criptografía de clave privada como AES**. Hay un algoritmo que puede simplificar los cálculos para romper el AES en un ordenador cuántico. Es el algoritmo de Grove, pero no es tan dramático como el algoritmo de Shor. En el caso de Grove se pasa de una complejidad $O(n)$ en el caso clásico a una complejidad $O(\sqrt{n})$ en el caso cuántico.

Tanto el sistema RSA como el de McEliece (basado en la teoría de códigos, que veremos más adelante) han sido propuestos el mismo año 1978. Los dos han conseguido resistir durante más de 30 años el esfuerzo del criptoanálisis para romperlos.

RSA se basa en la dificultad de la factorización. En el año 1978, el mejor algoritmo conocido para factorizar tenía una complejidad exponencial del orden

$$O(exp(\log(n)^{1/2} \log\log(n)^{1/2})).$$

Con el tiempo, esta complejidad ha sido mejorada, básicamente utilizando nuevos conceptos matemáticos y, actualmente, parece ser que ya no puede conseguirse ninguna mejoría, manteniéndose una complejidad del orden

$$O(exp(\log(n)^{1/3}\log\log(n)^{1/3})).$$

La complejidad del sistema McEliece era del orden $O(exp(n/(2\cdot\log(n))))$, en 1978. Ha habido mejoras y parece ser que la mejor complejidad posible seguirá siendo más o menos la misma, salvo que quizás en lugar del factor 2 habrá una constante algo menor.

Complejidad

La complejidad del RSA se llama *subexponecial* y la del sistema Mc.Eliece se llama *exponencial*.

La pregunta, ahora, es obvia. ¿Por qué actualmente no usamos el sistema de McEliece en lugar de RSA? La respuesta rápida es que el tamaño de la clave hace que optemos por RSA en lugar de McEliece. Para el mismo nivel de seguridad, RSA utiliza claves de algunos miles de bits, mientras que en McEliece habría que utilizar del orden del millón de bits.

Todavía falta tiempo para que la computación cuántica sea una realidad. En el camino, podemos pensar en ordenadores clásicos cada vez más potentes (no hace falta que sean potentes trabajando aislados, pero podemos pensar en grandes sistemas de computación distribuida).

Algunos sistemas de cifrado, como el RSA, con cuatro mil bits de clave se cree que serán resistentes a los ataques con grandes ordenadores clásicos, pero no lo serán a los ataques con grandes computadoras cuánticas. Algunas alternativas, como por ejemplo McEliece con una clave de cuatro millones de bits, se cree que será capaz de resistir los ataques de los grandes ordenadores clásicos y cuánticos.

Nos hemos centrado en la criptografía de clave pública, puesto que los ordenadores cuánticos parece que tienen muy poco efecto sobre la criptografía de clave privada y la criptografía basada en funciones *hash*.

Aunque los ordenadores cuánticos no son todavía una realidad, sí que hay ciertos fenómenos físicos basados en la mecánica cuántica que pueden usarse en criptografía. Un ejemplo claro es el sistema de distribución de claves basado en el algoritmo de Bennet y Brassard. Aparte de este algoritmo criptográfico basado en la mecánica cuántica, si ahora nos situásemos en la época en que ya funcionen los ordenadores cuánticos tendríamos algunos planteamientos que deberíamos empezar a estudiar:

Ver también

El algoritmo de Bennet y Brassard se estudia en el subapartado 3.1 de este módulo.

- **Eficiencia**: El software de criptografía post-cuántica es más lento que el software criptográfico de hoy en día.

- **Confianza**: A día de hoy, estamos utilizando sistemas criptográficos clásicos que han sobrevivido a muchos años de esfuerzos criptoanalíticos para

romperlos. Al considerar nuevos sistemas cuánticos, que son recientes y no del todo probados, es necesario que también los criptoanalistas dispongan de tiempo para buscar los ataques a estos sistemas y depurarlos.

- **Usabilidad**: Es importante desarrollar software y hardware adaptando las implementaciones a los nuevos conceptos con mucho cuidado, para evitar pérdidas de tiempo o efectos colaterales. Implementaciones de conceptos como los de aleatoriedad o relleno (*padding*) se deben mejorar en las nuevas tecnologías y seguramente habrá sistemas híbridos, compartiendo las nuevas tecnologías con las actuales que necesitarán estandarizaciones.

3.1. Criptografía cuántica

A diferencia de los sistemas criptográficos convencionales, ya sean de clave pública o privada (que basan su seguridad en el hecho de mantener una clave de manera privada) los sistemas criptográficos cuánticos basan su fortaleza en un fenómeno físico. En 1984, Bennet y Brassard diseñaron (a nivel teórico) un protocolo criptográfico basado en un hecho de la física cuántica: la imposibilidad de medir simultáneamente un par de observables (principio de incertidumbre de Heisenberg, 1927).

La seguridad tradicional de un método de distribución de claves se ha basado en problemas intratables debido a su complejidad computacional. El método de Benet y Brassart se basa en un hecho físico inviolable.

Podemos imaginar una nube de fotones que vibran en todas las direcciones perpendiculares a su línea de propagación. Si los hacemos pasar por un filtro polarizado en vertical, los fotones que vibran verticalmente pasarán por el filtro y, los otros, lo harán con una probabilidad $\cos^2(\varphi)$, donde φ es el ángulo que separa su dirección de vibración de la vertical. Sólo los fotones que vibran horizontalmente ($\cos(\varphi) = 0$) no pasarán por el filtro.

El protocolo que describimos aquí permite a dos usuarios A y B compartir una clave secreta sobre un canal no seguro. Al final del protocolo, A y B tendrán la seguridad que ningún enemigo criptoanalista ha interceptado su comunicación y podrán usar la clave compartida para cifrar los siguientes mensajes (normalmente con un sistema como el AES).

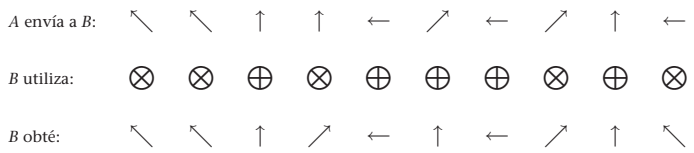
Protocolo de Ekert

En 1991 Artur Ekert presentó otro protocolo diferente al de Bennet y Brasard. En este caso, A y B reciben los fotones de una pareja "entrelazada". En este caso, la seguridad se basa en el efecto Einstein-Podolsky-Rosen.

- A envía a B una secuencia de pulsos de fotones. Cada pulso está polarizado aleatoriamente en una de las cuatro direcciones: vertical ($\uparrow$), horizontal ($\leftarrow$), según la diagonal primaria ($\nwarrow$) y según la diagonal secundaria ($\nearrow$).

- B utiliza, aleatoriamente, un detector polarizado para detectar polarizaciones vertical-horizontal o polarizaciones en diagonal, pero no las dos al mismo tiempo.

Por ejemplo:



Los pulsos 4, 6 y 10 también podían haber sido ↖, ← y ↗, respectivamente.

- *A* y *B* se comunican a través de un canal no seguro para ver qué detectores ha usado *B* que no haya usado *A*. Los dos, *A* y *B* se guardan solo los bits que corresponden a los detectores correctos.

  En nuestro caso, los dos usuarios están de acuerdo en que el bit 1 se representara por ← y ↖, mientras que el bit 0 sería ↑ y ↗. O sea que *A* y *B* habrán generado, entre ellos, la secuencia:

  1101100

  Esta secuencia no es conocida por el espía que intenta interferir en la comunicación entre *A* y *B*, puesto que la conversación (sobre un canal inseguro) entre *A* y *B* solo decía qué detectores se habían usado correctamente. Y, cada detector puede dar, indistintamente, ambos resultados 1 o 0.

Cualquier espía que intercepte los fotones que envía *A* los deberá re-enviar a *B* y, básicamente, tiene dos grandes problemas:

- Cuando escucha la transmisión entre *A* y *B*, el espía solo puede deducir con seguridad los bits que corresponden a detectores que él mismo está usando y que, a la vez, coincidan con los detectores que están utilizando *A* y *B*.

- En una de cada cuatro veces, el detector que está usando el espía no coincide ni con el de *A* ni con el de *B* (que, a su vez, coincide con el de *A*). En estos casos, *A* y *B* están de acuerdo en el detector que han usado, pero el bit que obtendrán uno y otro será distinto. Si *A* y *B* descubren (por un canal que no hace falta que sea secreto) algunos de los bits obtenidos en el protocolo podrán deducir la presencia del espía y abortar el proceso.

Hay algunos prototipos de este protocolo que están funcionando sobre fibra óptica entre distancias del orden de los 200 km (Toshiba Research-2003).

Aun cuando la física cuántica nos asegura la validez del protocolo anterior, desde los primeros intentos de construcción de prototipos que lo implementarán ha habido varios problemas que han aplazado su comercialización. No es tan sencilla la construcción "segura" de todos los dispositivos implicados

en el protocolo. Los países asiáticos están al frente de la investigación en esta área. Japón ha anunciado planes para tener satélites con comunicaciones de tipo cuántico para el 2013 y China para el 2016.

3.2. Los códigos correctores de errores en la criptografía post-cuántica

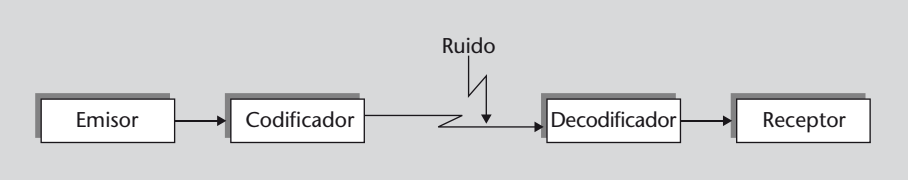
En este subapartado presentaremos dos sistemas criptográficos, basados en la teoría de códigos correctores de errores. En particular nos centraremos en los criptosistemas de McEliece y de Niederreiter, definidos a partir de códigos para los cuales se conoce un sistema de descodificación computacionalmente eficiente. En ambos casos, la función unidireccional utilizada para el cifrado se basa en una transformación de la matriz generadora del código, en el caso de McEliece, o de la matriz de control, en el caso de Niederreiter. Esta función unidireccional hará que el criptoanalista deba trabajar con un código equivalente para el que el algoritmo de descodificación es computacionalmente ineficiente.

Por comprender mejor estos dos criptosistemas, veamos unas nociones básicas de códigos correctores de errores.

3.2.1. Nociones básicas de códigos correctores de errores

Los códigos detectores y correctores de errores tienen como objetivo principal reducir la probabilidad de error en la descodificación, mediante la incorporación de redundancia en la transmisión.

Consideremos ahora el sistema de transmisión de información, representado por el diagrama:



El principio general de la detección de errores es el siguiente: el codificador añade, según una regla C, una cierta cantidad de información suplementaria a la información útil, y el descodificador, en primer lugar, verifica si esta ley es respetada. Si no lo es, estamos seguros de que, al menos, se ha producido un error. Esta ley C que caracteriza el código consiste en una correspondencia biunívoca entre la información a enviar y el mensaje enviado.

En lo que sigue, consideramos el cuerpo finito $\mathbb{F} = \mathbb{F}_q$, donde $q = p^m$ y p un número primo.

Definición 3.1 (Código-bloque).

Dada una fuente de información $S = \{A_1, A_2, \cdots, A_k\}$ y un alfabeto $\mathbb{F}$, se considera el producto cartesiano $\mathbb{F}^n$.

Llamaremos código-bloque, de longitud n, a cualquier subconjunto $C \subset \mathbb{F}^n$ de manera que a cada elemento $A_i \in S$, le hacemos corresponder, de manera única, un vector $v_i \in \mathbb{F}^n$, al que llamaremos **palabra-código**.

Si el cardinal de S es M nos referiremos al código C como:

$C(M,n) = \{v_i = (v_{i_1}, v_{i_2}, \cdots, v_{i_n})$, tal que $v_{i_j} \in \mathbb{F}\}$.

Podemos considerar un código-bloque como un conjunto de secuencias de elementos de un cuerpo finito $\mathbb{F}$ (palabras-código) de la misma longitud. Un código-bloque es binario cuando $q = 2$.

Los parámetros a considerar en un código-bloque $C(M,n)$, además de su longitud, n, y su número de palabras-código M, son: la tasa de transmisión: $(1/n)log_q(M)$ y la distancia mínima entre todas las palabras-código, d.

Ejemplo 3.1.

Supongamos el caso en el que tenemos que transmitir dos posibles mensajes, $S = \{A_1, A_2\}$ donde A_1= "Hace sol" y A_2 = "Llueve". El canal por el cual debe circular la transmisión es binario, es decir, el alfabeto será $\mathbb{F}_2 = \{0,1\}$.

Un código-bloque para S puede ser $C(2,3) = \{A_1 \rightarrow (0,0,0);\ A_2 \rightarrow (1,1,1)\}$.

Definición 3.2 (Distancia de Hamming).

Dados dos elementos $x,y \in \mathbb{F}^n$, definimos su distancia de Hamming como:

$$d_H(x,y) = \#\{i : 1 \leq y \leq n, x_i \neq y_i\},$$

es decir, la distancia entre dos vectores x e y es la cantidad de componentes diferentes que tienen entre uno u otro.

Distancia y métrica

La distancia de Hamming satisface las propiedades de la definición matemática de distancia y define una métrica.

$\forall x,y,z \in \mathbb{F}^n$:

1) $d_H(x,x) = 0$

2) $d_H(x,y) = d_H(y,x)$

3) $d_H(x,y) + d_H(y,z) \leq d_H(x,z)$

Definición 3.3 (Distancia mínima).

Dado un código $C(M,n)$, definiremos la distancia mínima d, del código, como:

$$d = min\{d_H(x,y) : x \neq y, x,y \in C\}.$$

Definición 3.4 (Regla de descodificación a distancia mínima).

Dado un código $C(M,n)$, definiremos la regla de descodificación a distancia mínima como la que descodifica un vector recibido $u \in \mathbb{F}^n$ por la palabra-código que está a mínima distancia de él. Es decir establece una aplicación $\mathbb{F}^n \longrightarrow C$, tal que si $u \longrightarrow v$, resulta que v es la palabra-código para la cual $d_H(v,u) = min\{d_H(v',u), \forall v' \in C\}$.

Definición 3.5 (Capacidad correctora).

Podemos considerar en $\mathbb{F}^n$ las bolas centradas en las palabras-código de radio el máximo valor posible de forma que las bolas sean disjuntas. El radio de estas bolas se puede calcular como $c = \lfloor \frac{d-1}{2} \rfloor$ y este valor se denomina capacidad correctora del código.

Diremos que el código es **c-corrector**.

3.2.2. Códigos lineales

Los códigos lineales permiten una buena solución al problema de la codificación/descodificación, considerando la regla de descodificación a distancia mínima.

Consideramos la estructura de espacio vectorial de $\mathbb{F}^n = \{u = (u_1, \cdots, u_n) : u_i \in \mathbb{F}\}$, de dimensión n, sobre $\mathbb{F}$, con las operaciones suma y producto por escalares habituales en un espacio vectorial:

$$u + v = (u_1 + v_1, \cdots, u_n + v_n),\ u,v \in \mathbb{F}^n$$

$$k \cdot (u_1, \cdots, u_n) = (k \cdot u_1, ..., k \cdot u_n);$$

Definición 3.6 (Código lineal).

Un código-bloque se llama lineal si es subespacio vectorial de $\mathbb{F}^n$. Si la dimensión de este subespacio es k, el código tendrá $M = q^k$ palabras-código, de longitud n, y será denotado por $C(n,k)$.

Definición 3.7 (Peso de un vector).

El peso $wt(v)$ de un vector $v \in \mathbb{F}^n$ es el número de coordenadas no nulas de este vector. Es decir:

$$wt(v) = \#\{v_i \neq 0 : v_i \in \mathbb{F}\}.$$

Definición 3.8 (Error de transmisión).

Un error en la transmisión se corresponde con un cambio de coordenada entre la palabra-código de entrada y el vector de salida.

Estas características nos dan la capacidad detectora y correctora de errores.

Lema 3.9.

Un código lineal, con distancia mínima $d \geq 2c+1$, puede detectar hasta $d-1$ errores y corregir hasta c si se utiliza el esquema de descodificación a distancia mínima.

Definición 3.10 (Matriz generadora).

De la definición de código lineal resulta que todo conjunto de k vectores de $\mathbb{F}^n$, linealmente independientes, constituye una base de un código lineal $C(n,k)$. Así, todo código lineal puede ser definido por la matriz $k \times n$, donde las k filas son k vectores independientes, de una cierta base de C.

A tal matriz, denotada $G_{k\times n}$, se la llama matriz generadora del código, ya que toda palabra-código v es una combinación lineal de las k filas de esta matriz.

Podemos escribir, matricialmente:

$$v = a \cdot G$$

y dando a $a = (a_1, a_2, \cdots, a_k)$ todos los valores posibles (q^k en total), obtendremos todas las palabras-código de C.

Coincidencia en los valores del peso mínimo y de distancia mínima

Un código lineal tiene la propiedad que la suma de dos palabras-código es una palabra-código, por lo tanto:
$d_H(u,v) =$
$\#\{i : u_i \neq v_i\} =$
$\#\{i : u_i - v_i \neq 0\} =$
$wt(u-v)$
Así, en un código lineal, la distancia entre dos palabras-código es igual al peso de otra palabra-código y, en consecuencia, la distancia mínima, no nula, coincide con el peso mínimo del conjunto de palabras-código no nulas.

Definición 3.11 (Matriz de control).

Un código lineal $C(n,k)$ puede ser descrito, además, por otra matriz. En efecto, el subespacio ortogonal al código, que es de dimensión $n-k$, puede ser descrito por una matriz, H, donde las $n-k$ filas son los $n-k$ vectores linealmente independientes de este subespacio, ortogonales a todos los vectores de C. O sea, $H^T_{(n-k)\times n} \cdot G_{k\times n} = 0_{(n-k)\times k}$.

Entonces, todo vector v del código tiene la propiedad de ser ortogonal a esta matriz, es decir, verifica:

$$H \cdot v^T = 0 \Longleftrightarrow v \in C(n,k)$$

Esta matriz H permite controlar si un vector determinado pertenece o no al código y, por esto, se llama matriz de control.

Ortogonalidad

Ortogonal, o perpendicular, va a significar que el producto escalar sea cero.

Ejemplo 3.2.

Para construir un código lineal binario $C(6,3)$ podemos tomar la siguiente matriz generadora:

$$G_{3\times 6} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

que nos va a permitir codificar de la siguiente manera:

a	$v = a \cdot G$	$wt(v)$
(0,0,0)	(0,0,0,0,0,0)	0
(1,0,0)	(1,1,1,0,0,0)	3
(0,1,0)	(0,1,0,0,1,1)	4
(0,1,1)	(0,1,0,0,1,1)	3
(0,0,1)	(0,0,1,1,1,0)	3
(1,1,0)	(1,0,0,1,0,1)	3
(1,0,1)	(1,1,0,1,1,0)	4
(1,1,1)	(1,0,1,0,1,1)	4

El peso mínimo del código, que coincide con la distancia mínima, es 3 y, por lo tanto, podrá detectar hasta 2 errores en la transmisión, pero solo podrá corregir 1.

Por otra parte, se puede verificar que la matriz $H_{3\times 6}$ es una matriz de control para el código $C(6,3)$ anterior, puesto que las tres filas de H son una base del subespacio ortogonal a C.

$$H_{3\times 6} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Dado un código lineal $C(n,k)$ y un vector $e \in \mathbb{F}^n - C$ tomemos el conjunto $C+e = \{v+e : v \in C\}$, llamado **coset** (o trasladado) de C, con líder (o representante) e, que contiene q^k vectores distintos de $\mathbb{F}^n$.

Teorema 3.12.

Un código lineal $C(n,k)$ admite q^{n-k} cosets diferentes, que constituyen una partición de todo el espacio $\mathbb{F}^n$.

Demostración: Consideremos la relación de equivalencia sobre $\mathbb{F}^n$, inducida por el código C, definida como:

$$e,u \in \mathbb{F}^n;\ u \backsim e \Longleftrightarrow u - e \in C$$

que nos induce la partición de $\mathbb{F}^n$ en clases de equivalencia, donde dos vectores de $\mathbb{F}^n$ son equivalentes si y solo si pertenecen a la misma clase. De hecho tenemos:

$$u - e \in C \Longleftrightarrow u \in C + e$$

de donde deducimos que las clases de equivalencia coinciden con los cosets. ∎

Obviamente, cada coset contiene q^k elementos (tantos como palabras-código).

Definición 3.13 (La aplicación síndrome).

Dado un código lineal $C(n,k)$, consideremos su matriz de control H. Esta $(n-k) \times n$ matriz nos permite definir una aplicación lineal, del espacio $\mathbb{F}^n$ en el subespacio de las $(n-k)$-uplas de $\mathbb{F}$, o sea $\mathbb{F}^{n-k}$:

$$S : \mathbb{F}^n \longrightarrow \mathbb{F}^{n-k},$$

tal que cada vector $u \in \mathbb{F}^n$ se transforma en $S(u) = H \cdot u^T$. Este valor $S(u)$ recibe el nombre de **síndrome del vector u**.

Lema 3.14.

Existe una correspondencia biunívoca entre los q^{n-k} cosets posibles y los q^{n-k} síndromes posibles.

A causa de esta correspondencia biunívoca entre cosets y síndromes, y habiendo hecho la elección de los líderes de los cosets tomando un vector de peso

mínimo en cada coset, el síndrome de un vector cualquiera caracteriza el líder del coset de manera única, y por lo tanto podemos considerar la re-definición de la regla de descodificación a distancia mínima:

Definición 3.15 (Regla de descodificación vía síndrome).

Si $u \in \mathbb{F}^n$ es el vector recibido:

1) Calcular $S(u) = H \cdot u^T$

2) Determinar el líder e tal que $S(e) = H \cdot e^T = S(u)$

3) Descodificar u por la palabra-código $v^* = u - e$.
(Efectivamente, $v^* \in C$, ya que $H \cdot (v^*)^T = H \cdot (u-e)^T = H \cdot u^T - H \cdot e^T = S(u) - S(e) = 0$).

Corrección del error

Si solo se ha producido un error, entonces estamos seguros de que $v^* = v$ es realmente la palabra-código que se había enviado. De lo contrario, si se ha producido más de un error, entonces la descodificación, aun cuando $v^* \in C$, sería incorrecta.

Ejemplo 3.3.

Descodificar el vector $u = (1,1,0,1,0,1)$, usando la descodificación vía síndrome, por el código $C(6,3)$ del ejercicio anterior (este código es 1-corrector).

$$S(u) = H \cdot u^T = H \cdot (1,1,0,1,0,1)^T = (1,0,0)$$

El vector $e = (0,1,0,0,0,0) \in \mathbb{F}^n - C$, cumple que $S(e) = S(u)$:

$$S(e) = H \cdot e^T = H \cdot (0,1,0,0,0,0)^T = (1,0,0)$$

Así, la estimación que hacemos de la palabra-código enviada es:

$$v^* = u - e = (1,1,0,1,0,1) - (,0,1,0,0,0,0) = (1,0,0,1,0,1) \in C$$

3.2.3. Los códigos lineales cíclicos: *BCH* y *RS*

Para hacer el estudio de estas familias de códigos asociaremos a los vectores del espacio $\mathbb{F}^n$ un polinomio, de grado inferior o igual a $n-1$, tal que sus coeficientes coincidan con las coordenadas del vector. Entonces un código lineal cíclico podrá ser considerado como un ideal del anillo de polinomios de grado inferior o igual a $n-1$ y coeficientes en $\mathbb{F}$ (notaremos a este anillo por $\mathbb{F}^n[X] \approx \mathbb{F}[X]/X^n - 1$).

Definición 3.16 (Códigos cíclicos).

Un código C de longitud n se llama cíclico si toda permutación cíclica de una palabra-código es también una palabra-código. Es decir:

$$\forall v = (v_0, v_1, \cdots, v_{n-1}) \in C \implies v^{\pi} = (v_{n-1}, v_0, \cdots, v_{n-2}) \in C$$

Para estudiar las propiedades algebraicas de estos códigos, es cómodo utilizar una descripción polinómica. A toda palabra-código le asociamos un polinomio de la siguiente manera:

$$v = (v_0, v_1 \ldots, v_{n-1}) \in C \implies v(X) = v_0 + v_1 X + \ldots + v_{n-1} X^{n-1}$$

Si llamamos $C(X)$ al conjunto de los polinomios asociados a las palabras-código de C, observaremos que: $v^{\pi}(X) = X \cdot v(X) - v_{n-1} X^n$; es decir, que: $v^{\pi}(X) = X \cdot v(X) \pmod{X^n - 1}$.

Lema 3.17.

$\mathbb{F}^n[X]$, con la suma habitual de polinomios y el producto $a(X)^* b(X) = a(X) \cdot b(X) \pmod{X^n - 1}$ tiene estructura de anillo conmutativo.

Lema 3.18.

Un código lineal $C(n,k)$ es cíclico $\iff$ $C(X)$ es un ideal de $\mathbb{F}^n[X]/(X^n - 1)$.

Los dos lemas anteriores nos permiten escribir el siguiente resultado, que es la base de la construcción de los códigos lineales y cíclicos.

Teorema 3.19.

Sea C un código lineal y cíclico de longitud n (ideal de $\mathbb{F}^n[X]/(X^n - 1)$). Sea $g(X)$ un polinomio mónico (el coeficiente del término de mayor grado es 1) de grado más pequeño dentro de $C(X)$. Sea r el grado de $g(X)$. Entonces se cumple:

1) $g(X)$ es el único polinomio mónico de grado r en $C(X)$.

2) $g(X)$ es el generador de $C(X)$ como ideal principal de $\mathbb{F}^n[X]/(X^n - 1)$ (es decir, $\forall v(X) \in C(X)$ existe $h(X)$ tal que $v(X) = g(X) * h(X)$).

3) $g(X)$ divide $X^n - 1$ (con n longitud del código).

4) $\{X^i \cdot g(X), 0 \leq i \leq (n - r - 1)\}$, genera $C(X)$ como subespacio vectorial. Es decir, $\forall v(X) \in C(X)$, existen ciertos coeficientes a_i para los que $v(X) = \sum_{i=0}^{n-r-1} a_i X^i \cdot g(X) \pmod{X^n - 1}$.

Este teorema nos permite asegurar que todo polinomio $g(X) \in \mathbb{F}^n[X]$, de grado r, divisor de $X^n - 1$ genera un código lineal y cíclico $C(n,k)$ que tiene por **matriz generadora**:

$$G_{k\times n} = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-1} & \cdots & 0 & 0 & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-1} & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots & g_0 & g_1 & \cdots & g_{n-1} \end{pmatrix}.$$

Es decir, las filas de $G_{k\times n}$, donde $k = n - r$, son los coeficientes de $X^i \cdot g(X)$ (mod $X^n - 1$).

Observar que multiplicar $g(X)$ por X^i (mod $X^n - 1$) equivale a desplazar a la derecha (con retroalimentación) i posiciones los coeficientes de $g(X)$, $i = 1, \cdots, n - r - 1$, visto como un vector de $\mathbb{F}^n$.

3.2.4. Los códigos cíclicos BCH

Los códigos BCH fueron introducidos por Hocquenghem (1959) y Chaudhuri, Bose (1960), donde la estructura algebraica se basa en los cuerpos finitos, $\mathbb{F}_{p^m}$. En nuestro caso tomaremos $p = 2$.

Este tipo de códigos vienen definidos por dos parámetros m y c y verifican:

Teorema 3.20.

Para todo entero n de la forma $n = 2^m - 1$, $m \geq 3$, y todo entero c tal que $n - c \cdot m > 0$, existe un código cíclico c-corrector, de longitud n, dimensión $k \geq n - c \cdot m$ y distancia mínima $d \geq 2 \cdot c + 1$, que tiene por polinomio generador:

$$g(X) = mcm(m_1(X), m_3(X), \ldots, m_{2\cdot c-1}(X))$$

siendo $m_i(X)$ el polinomio mínimo de α^i y α un elemento primitivo de $\mathbb{F}_{2^m}$.

Polinomio mínimo de α^i

Sea α un elemento primitivo de $\mathbb{F}_{2^m}$, Sea t tal que $(\alpha^i)^t = 1$ y sea s el más pequeño entero tal que t divide $2^s - 1$. Entonces, el polinomio mínimo de α^i se puede calcular como:

$$m_i(X) = \Pi_{j=0}^{s-1}(X - (\alpha^i)^{2^j})$$

Definición 3.21 (Los códigos cíclicos BCH).

Un código con las características del teorema anterior se llama **código BCH primitivo**.

La matriz de control de los códigos BCH es:

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \alpha^{2c-1} & \alpha^{(2c-1)\cdot 2} & \cdots & \alpha^{(2c-1)\cdot(n-1)} \end{pmatrix},$$

ya que que $v(X) = v_0+v_1X+...+v_{n-1}X^{n-1}$, $v_i \in \mathbb{F}$, será el polinomio asociado a una palabra-código v, si y solo si, $v(\alpha^i) = 0, \forall i = 1,3,\cdots,2c-1$. Es decir, las palabras código son múltiplos del polinomio $g(X)$ que tiene, por construcción, como ceros los elementos $\alpha,\alpha^3,\cdots,\alpha^{2c-1}$ (y, también, $\alpha^2,\alpha^6,\cdots,\alpha^{2c}$).

En la tabla siguiente podemos ver los parámetros de algunos códigos BCH.

n	t	k	$g(X)$
7	1	4	$X^3 + X + 1$
–	2	1	$X^6 + X^5 + X^4 + X^3 + X^2 + X + 1$
15	1	11	$X^4 + X + 1$
–	2	7	$X^8 + X^7 + X^6 + X^4 + 1$
–	3	5	$X^{10} + X^8 + X^5 + X^4 + X^2 + X + 1$
31	1	26	$X^5 + X^4 + X^2 + 1$
–	2	21	$X^{10} + X^9 + X^8 + X^6 + X^5 + X^3 + 1$
–	3	16	$X^{15} + X^{11} + X^{10} + X^9 + X^8 + X^7 + X^5 + X^3 + X^2 + X + 1$
–	5	11	$X^{20} + X^{18} + X^{17} + X^{13} + X^{10} + X^9 + X^7 + X^6 + X^4 + X^2 + 1$
–	7	6	$X^{25} + X^{24} + X^{21} + X^{19} + X^{18} + X^{16} + X^{15} + X^{14} + X^{13} + X^{11} + X^9 + X^5 + X^2 + X + 1$

3.2.5. Los códigos cíclicos RS

Reed y Solomon introdujeron en 1960, una clase particular de códigos BCH que mejoraba, todavía más, las prestaciones de corrección y la facilidad de descodificación. En particular, los códigos de esta familia tienen la mejor capacidad detectora y correctora, dado el par de parámetros n y k.

Definición 3.22 (Los códigos cíclicos RS).

Un código de Reed-Solomon (RS), primitivo, es un código cíclico sobre $\mathbb{F}_{2^m}$, de longitud $n = 2^m - 1$ y dimensión k, con distancia mínima $d = n - k + 1$, y que tiene por polinomio generador:

$$g(X) = (X - \alpha) \cdot (X - \alpha^2) \cdots (X - \alpha^{d-1})$$

donde α es un elemento primitivo de $\mathbb{F}_{2^m}$.

Corrección de paquetes de errores

Los símbolos de las palabras-código son elementos $\alpha^i \in \mathbb{F}_{2^m}$, lo cual quiere decir que cada símbolo que se transmite por el canal es un elemento de m coordenadas binarias. Por lo tanto, un código de Reed-Solomon (n,k), en realidad transmite $m \cdot k$ bits de información mediante una palabra-código de $n \cdot m$ bits. En consecuencia, es importante señalar que, con estos códigos, los errores no son bits aleatorios sino que pueden ser paquetes de m bits, que será considerado como un solo error. Esta característica mejora la capacidad correctora global, puesto que en realidad puede corregir hasta c paquetes de m errores (cada error es un cambio de un α^i por un α^j).

Nota

Un código lineal siempre satisface que $d \leq n-k+1$ (cota de Singleton). Un código que satisface la igualdad se llama MDS **(máxima distancia separable)**. Evidentemente, los códigos RS son de máxima distancia separable.

La matriz de control de los códigos RS es:

$$H_{(n-k)\times n} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ 1 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \alpha^{d-1} & \alpha^{2(d-1)} & \cdots & \alpha^{(d-1)\cdot(n-1)} \end{pmatrix},$$

ya que $v(X) = v_0+v_1X+...+v_{n-1}X^{n-1}$, $v_i \in \mathbb{F}^m$. En este caso, $v(X)$ será el polinomio asociado a la palabra-código v, si y solo si, $v(\alpha^i) = 0, \forall i = 1,2,\cdots,d-1$. Es decir, las palabras código son múltiplos del polinomio $g(X)$ que, por construcción, tiene por ceros los elementos $\alpha,\alpha^2,\cdots,\alpha^{d-1}$.

En 1969, Berlekamp y Massey dieron un algoritmo muy eficiente de descodificación, basado en el teorema de Dirichlet. Estos códigos han sido ampliamente utilizados en sistemas de almacenamiento de datos (CDs, DVDs,...), en módems de alta velocidad (ADSL, DSL,...), en televisión digital (TDT, MPEG2, MPEG4, ...) y también propuestos para ser usados en criptografía.

3.3. Los criptosistemas de McEliece y de Niederreiter

En este subapartado veremos los dos criptosistemas de McEliece y de Niederreiter, donde la función unidireccional se basa en códigos correctores de errores.

3.3.1. Criptosistema de McEliece

Robert McEliece propuso en 1978 un criptosistema de clave pública fundamentado en que el algoritmo de descodificación de un código lineal, en general, no es computacionalmente eficiente (R. J. McEliece (1978). "A public-key cryptosystem based on algebraic coding theory". *DSN Progress Report*, 42-44 (1978)).

En su propuesta utilizó los códigos de Goppa (de la misma familia de códigos que los BCH y RS, los llamados **códigos alternantes**), de los cuales se conoce un algoritmo de descodificación computacionalmente eficiente. En particular,

el algoritmo de descodificación de Patterson que tiene un funcionamiento muy parecido al de Berlekamp-Massey.

La idea principal del criptosistema consiste en transformar la matriz generadora, $G_{k\times n}$, del código alternante escogido, $C(n,k)$, c-corrector, y transformarla en una matriz generadora de un nuevo código lineal $C'(n,k)$, que notaremos por $G'_{k\times n}$ (que también será c-corrector).

Para llevar a término esta transformación, se utilizan dos matrices; una matriz binaria, no singular, $S_{k\times k}$, por lo tanto invertible, y una matriz de permutación $P_{n\times n}$.

Así, la **clave privada** será $G_{k\times n}$ y las matrices $S_{k\times k}$ y $P_{n\times n}$, mientras que la **clave pública** será: $G'_{k\times n} = S_{k\times k}\cdot G_{k\times n}\cdot P_{n\times n}$ y el parámetro c (la capacidad correctora del código $C(n,k)$).

Supongamos un usuario B que quiere enviar un mensaje cifrado al usuario A, que tiene G', como hemos descrito antes, como clave pública:

- **Algoritmo de cifrado**. Si m es el mensaje a cifrar por B, éste lo dividirá en bloques de k símbolos. Sea m_i un de estos bloques; entonces el criptograma correspondiente será:

$$c_i = E_{G'}(m_i) = m_i\cdot G' + e,$$

  siendo $e\in\mathbb{F}^n$ un vector de error arbitrario, escogido por B, tal que $wt(e)\leq c$.

- **Algoritmo de descifrado**. Este algoritmo hace uso del conocimiento, por parte del usuario A, de S^{-1} y P^{-1}. Así, recibido c_i, el usuario A hará:

  1) Calcular $c'_i = c_i\cdot P^{-1} = (m_i\cdot G' + e)\cdot P^{-1} = m_i\cdot S\cdot G + e\cdot P^{-1}$, donde $e\cdot P^{-1}$ es un error que se puede corregir por G, puesto que evidentemente $wt(e\cdot P^{-1}) = wt(e)$, ya que la permutación no cambiará el número de coordenadas no nulas.

  2) Aplicar a c'_i el algoritmo de descodificación del código $C(n,k)$. Este corregirá el error $e\cdot P^{-1}$ y nos devolverá el vector $m_i\cdot S\in\mathbb{F}^k$.

  3) Multiplicando, ahora, el vector recibido por S^{-1} reencontraremos m_i. Efectivamente, $m_i = m_i\cdot S\cdot S^{-1}$.

Pese a tratarse de un criptosistema en el cual los procesos de cifrado y descifrado son relativamente rápidos, en la actualidad se utiliza escasamente. Esto es debido, principalmente, a los tamaños de clave (2^{19} bits para la clave pública) y al factor de expansión del mensaje que hace que el criptograma tenga un tamaño un 60 % mayor que el mensaje original. Sin embargo, dado que el algoritmo de Shor no afecta a este criptosistema, parece ofrecer resistencia al criptoanálisis basado en computación cuántica.

3.3.2. Criptosistema de Niederreiter

H. Niederreiter, en 1986, propone un criptosistema, dual al de McEliece, basado en el uso de códigos *GRS* (*generalized Reed-Solomon)*, estrechamente ligados a los códigos de Goppa, y definidos por su matriz de control. La propuesta de utilización de estos códigos se basa en la posibilidad de reducir el tamaño de los parámetros respecto a los de Goppa (H. Niederreiter (1986). *Knapsack-type cryptosystem and algebraic coding theory, Problems of Control and Information Theory*).

En su propuesta, Niederreiter, utilizó los códigos *GRS* para los cuales, se conoce un algoritmo de descodificación computacionalmente eficiente. La idea principal consiste en transformar la matriz de control, $H_{(n-k)\times n}$, del código escogido, sobre $\mathbb{F}_{2^m}$, c-corrector ($c = \lfloor (d-1)/2 \rfloor$), y transformarla en una matriz de control de un nuevo código código lineal, que denotaremos por $H'_{(n-k)\times n}$ (que también será c-corrector).

Para llevar a término esta transformación, se utilizan dos matrices; una matriz binaria, no singular, $S_{(n-k)\times(n-k)}$, por lo tanto invertible, y una matriz de permutación $P_{n\times n}$.

Así, la **clave privada** será $H_{(n-k)\times n}$ y las matrices $S_{(n-k)\times(n-k)}$ y $P_{n\times n}$, mientras la **clave pública** será: $H'_{(n-k)\times n} = S_{(n-k)\times(n-k)} \cdot H_{(n-k)\times n} \cdot P_{n\times n}$ y el parámetro c (la capacidad correctora del código $C(n,k)$).

Supongamos que el un usuario B quiere enviar un mensaje cifrado a otro usuario A, que tiene H', como hemos descrito antes, como clave pública:

- **Algoritmo de cifrado**. Si m es el mensaje a cifrar por B, éste lo dividirá en bloques m_i de n símbolos, tales que $wt(m_i) \leq c$. Entonces el criptograma correspondiente será:

$$c_i = (H') \cdot (m_i)^T \in \mathbb{F}_{2^m},$$

  que da el síndrome de m_i.

- **Algoritmo de descifrado**. Este algoritmo hace uso del conocimiento, por parte del usuario A, de S^{-1} y P^{-1}. Así, recibido c_i, el usuario A hará:

  1) Calcular $c'_i = S^{-1} \cdot c_i = S^{-1} \cdot H' \cdot (m_i)^T = H \cdot P \cdot (m_i)^T = H \cdot (m'_i)^T$, donde c'_i es el síndrome de $m'_i = m_i \cdot P^T$, calculado a partir de H.

  2) Aplicar el algoritmo de descodificación a c'_i, para encontrar $m_i \cdot P^T$.

  3) $m_i = m_i \cdot P^T \cdot (P^{-1})^T$

Equivalencia de los dos criptosistemas

Yuan Xing Le y otros demuestran que los criptosistemas de McEliece y Niederreiter son equivalentes, en términos de seguridad (Y. Xing Le; R. H. Deng; X. Mei Wang (1994). "On the equivalence of McEliece's and Niederreiter public-key cryptosystems". *IEEE Transaction on Information Theory*).

La siguiente tabla muestra algunos resultados interesantes sobra la eficiencia de los criptosistemas de McEliece y Niederreiter.

	(n,c)	(2048,32)	(2048,40)	(4096,22)	(4096,45)
McEliece	texto original	1928	1888	4024	3904
	criptograma	2048	2048	4096	4096
	tamaño clave pública	73 *kb*	86 *kb*	123 *kb*	234 *kb*
Niederreiter	texto original	232	280	192	352
	criptograma	352	4408	264	540
	tamaño clave pública	73 *kb*	86 *kb*	123 *kb*	234 *kb*

Lectura recomendada

R. Overbeack; N. Sendrier. (2009). "Code-based cryptography". A: D. Bernstein; J. Buchmann; J. Ding (eds.). *Post-Quantum Cryptography* (págs. 95-145). Springer.

Ejercicios de autoevaluación

1. Un usuario de una red ha recibido el criptograma: 1611,3556,4744,3504 resultado de cifrar caracteres, individualmente, de $M = \{A,B,\cdots,Y,Z\} \approx \{A = 02, B = 03,\cdots,Y = 26, Z = 27\}$; empleando el criptosistema RSA con la clave pública $n = 7597$ y $e = 4947$. Encontrar el mensaje original.

2. Utilizar un criptosistema RSA para dos usuarios A y B, con el mismo valor de

$$n = 151953280470109$$

y claves públicas, respectivamente $e_A = 17$ y $e_B = 19$. Suponer que el usuario A quiere cifrar, para enviar a B, el texto:

m = Los ordenadores cuánticos pueden dejar obsoletos los métodos actuales de cifrado

a) ¿Cuál será el resultado del cifrado?
b) ¿Cuál sería la firma *RSA* del mensaje?
c) Simular la verificación de la firma por parte de B.

3. A partir de un cuerpo finito $\mathbb{F}_{71}$ de 71 elementos y un elemento primitivo $\alpha = 7$, consideramos un criptosistema ElGamal.
a) Tomando $k = 2$, si el criptograma correspondiente a $m = 30$ es $c = (57,49)$, encontrar el valor de la clave pública utilizada.
b) Si se toma otro valor de k, y resulta que para el mismo valor de m obtenemos $c = (b,59)$, ¿cuál es el valor de b?

4. Construir un criptosistema ElGamal para dos usuarios A y B, en un cuerpo $\mathbb{F}_p$, con $p = 1231451311$ y $\alpha = 21$. Suponer que el usuario A tiene por clave privada $x_A = 113$ y que el usuario B tiene por clave privada $x_B = 97$. Suponer el texto: m =*HOLA*, empleando los símbolos de $M = \{A,B,\cdots,Y,Z\} \approx \{A = 02, B = 03,\cdots,Y = 26, Z = 27\}$
a) ¿Cuál será el resultado del cifrado de m, tomando el valor de $k = 247$, que A enviará a B? Hacer el correspondiente descifrado para reencontrar m.
b) ¿Cuál sería la firma del mensaje, por parte de A?
c) Simular la verificación de la firma por parte de B.

5. Sobre la firma DSA:
a) Supongamos $p = 124540019$, $q = 17389$,$g = 10083255$, $x = 12496$ y $k = 9557$. Tomando como función de hash ficticia, $h(m) = m$, comprobar que la firma de $m = 5246$ es $(r,s) = (13752,9137)$.
b) Simular su verificación.

6. Ana y Bernardo deciden utilizar el protocolo cuántico de Bennett-Brassard para intercambiarse una clave de sesión. Al comienzo deciden que las polarizaciones \ y | indicarán un **1** y las polarizaciones / y — indicarán un **0**.

Ana envía 20 fotones a Bernardo. Ernesto, que es un espía que está observando la comunicación entre A y B, los intercepta todos y los reenvía a B con la misma polarización con que los ha detectado.

Posteriormente A y B se comunican públicamente entre ellos y deciden que los bits 1, 2, 3, 4, 5, 7, 8, 9, 10, 20 serán desestimados puesto que, en ellos, no han usado la misma polarización. La polarización de los bits restantes, como ha deducido B, es: / / / / / | | | | |.

Sabiendo que los bits 13, 14, 15, 16, que inicialmente había enviado A son, respectivamente, 1, 0, 0, 1, ¿sabríais decir con qué polaridad A ha enviado los fotones que representan estos bits? y, ¿qué polarizador ha utilizado Ernesto en estos bits? ¿por qué?

Solucionario

1. Para resolver el problema nos ayudaremos del software SAGE. En primer lugar hace falta factorizar n:

```
sage: n = 7597
sage: factor(n)
71 * 107
```

A continuación encontraremos el inverso $e = 4947$ módulo $\varphi(n) = 70 * 106 = 7420$:

```
sage: e = 4947
sage: phi = 7420
sage: d = inverse_mod(e,phi)
sage: print d
3
```

Con la función *power_mod*(C,d,n) encontraremos el mensaje original descifrando el mensaje recibido:

```
sage: power_mod(1611,d,n)
2
sage: power_mod(3556,d,n)
3
sage: power_mod(4744,d,n)
4
sage: power_mod(3504,d,n)
5
```

Finalmente, $m = ABCD$.

2. Utilizaremos el software SAGE para simplificar los cálculos. Para empezar, necesitamos pasar del texto alfabético a mensajes numéricos que nos permitan efectuar las operaciones *RSA*. Para hacerlo necesitamos definir algunas operaciones previas, las de codificar/descodificar letras, codificar/descodificar cadenas, cifrar/descifrar números, cifrar/descifrar mensajes. Damos por hecho que la programación elemental utilizada es conocida por el estudiante:

```
alphabet = ' abcdefghijklmnopqrstuvwxyz'
L = len(alphabet)
def codifica_char(lletra):
    return alphabet.index(lletra)

def descodifica_char(n):
    return alphabet[n]

def codifica_text(missatge):
    return [codifica_char(c) for c in text]

def descodifica_chain(llista):
    return ''.join([descodifica_char(v) for v in llista])

def chiper_RSA(llista,n,e):
    return [power_mod(valor,e,n) for valor in llista]

def unchiper_RSA(llista,n,d):
    return [power_mod(valor,d,n) for valor in llista]
```

a) Con estas definiciones previas podemos cifrar el mensaje que nos dan:

```
sage: m = 'Los ordenadores cuánticos pueden dejar obsoletos los métodos actuales de cifrado'
sage: n = 151953280470109
sage: e = 17
sage: chiper_RSA(codifica_text(m),n,e)

[762939453125, 93211876845592, 32590367823381, 0, 74138716094102, 103704942061406, 17179869184,
114274128424688, 97563638183746, 1, 17179869184, 74138716094102, 103704942061406, 32590367823381,
```

```
0, 69016003153490, 33625452816007, 1, 97563638183746, 105460815098081, 114274128424688, 129140163,
32590367823381, 0, 83278712378725, 74138716094102, 17179869184, 762939453125, 97563638183746, 0,
17179869184, 762939453125, 114274128424688, 119465547730806, 1, 103704942061406, 0, 74138716094102,
131072, 32590367823381, 74138716094102, 93211876845592, 762939453125, 105460815098081, 32590367823381,
0, 762939453125, 93211876845592, 32590367823381, 0, 19568778972781, 762939453125, 105460815098081,
74138716094102, 17179869184, 762939453125, 32590367823381, 0, 1, 129140163, 105460815098081,
33625452816007, 1, 93211876845592, 32590367823381, 0, 17179869184, 762939453125, 0, 119465547730806,
114274128424688, 16926659444736, 103704942061406, 1, 105460815098081]
```

b) Para firmar el mensaje debemos tener calculado el valor d que forma parte de la clave privada y que el enunciado del problema no nos da. Como el número n no es muy largo, lo podemos factorizar:

```
sage: factor(151953280470109)
1738934123 * 87383
```

Ahora podemos calcular $\varphi(n) = 1738934122 * 87382$ y el inverso de e módulo $phi(n)$.

```
sage: e = 17
sage: phi = 1738934122 * 87382
sage: d = inverse_mod(e,phi)
print d
8938325967565
```

Entonces la firma la podemos encontrar con la misma función que el cifrado, pero sustituyendo el parámetro e por d:

```
sage: missatge_signat = chiper_RSA(codifica_text(m),n,d)

[42782583250323, 30308012709953, 132858054847474, 0, 94150478556282,
30541430930646, 108907931565622, 76520780070238, 18215677713314, 1,
108907931565622, 94150478556282, 30541430930646, 132858054847474, 0,
44035504519670, 121144087766341, 1, 18215677713314, 16213728552100,
76520780070238, 84208423477578, 132858054847474, 0, 26232893409272,
94150478556282, 108907931565622, 42782583250323, 18215677713314, 0,
108907931565622, 42782583250323, 76520780070238, 54058519904057, 1,
30541430930646, 0, 94150478556282, 77779196084924, 132858054847474,
94150478556282, 30308012709953, 42782583250323, 16213728552100,
132858054847474, 0, 42782583250323, 30308012709953, 132858054847474, 0,
21980253538582, 42782583250323, 16213728552100, 94150478556282,
108907931565622, 42782583250323, 132858054847474, 0, 1, 84208423477578,
16213728552100, 121144087766341, 1, 30308012709953, 132858054847474, 0,
108907931565622, 42782583250323, 0, 54058519904057, 76520780070238,
96067072003657, 30541430930646, 1, 16213728552100]
```

c) El usuario B puede comprobar la firma del mensaje que le ha enviado el usuario A utilizando la clave pública de aquel y observando que el resultado da el mensaje inicial:

```
sage: descodifica_chain(unchiper_RSA(missatge_signat,n,e))

'Los ordenadores cuánticos pueden dejar obsoletos los métodos actuales de cifrado'
```

3. a) Escribamos la clave pública de A como $a = \alpha^U$, entonces el mensaje cifrado es $c = m \cdot (a)^k \pmod p$, o sea $57 = 30 \cdot (a)^2 \pmod{71}$ y, por lo tanto $a = \sqrt{57 \cdot 30^{-1}} \pmod{71} = \sqrt{9} \pmod{71}$.

```
sage: mod((57 * 30^(-1))^(1/2), 71)
9
```

O sea, la clave pública puede ser $\alpha^U = \pm 3 \pmod{71}$.

b) Si tomamos otro valor de k tenemos que $\alpha^k = b$ y $30 \cdot a^k = 59$, donde a es la clave pública. Entonces $b = 30 \cdot (59)^{-1} \pmod{71} = 33$.

4. Utilizaremos el software SAGE. Entramos los datos del ejercicio:

```
sage: p=1231451311
sage: alfa=21
sage: xA=113
sage: xB=97
sage: k=247
sage: m=9161302
sage: R=IntegerModRing(p)
```

Con la última instrucción indicamos que trabajaremos en el anillo $\mathbb{Z}_p$. Ahora ya podemos calcular:

a)
```
sage: YB=R(alfa^xB)
print YB
4198807
sage: YA=R(alfa^xA)
print YA
120638760
sage: K= R(alfa^k)
print K
840316018
```

El criptograma c, correspondiente a m, se encuentra haciendo:

```
sage:c=R(m*YB^k)
print c
399347538
```

Entonces, A envía c = 399347538 a B. Ahora, para reencontrar el mensaje enviado, B hará:

```
sage: beta= R(K^xB)
sage: mr=R(c/beta)
print mr
9161302
```

Efectivamente, el mensaje recibido mr coincide con el enviado m.

b) Para poder hacer la firma A, debe calcular en el anillo $\mathbb{Z}_{p-1}$, y para esto devolvemos el valor entero a K:

```
sage: R1=IntegerModRing(p-1)
sage: kinv=R1(1/k)
sage: K1=Integer(K)
sage:  s=R1((m-xA*K1)*kinv)
print s
14451644
```

Por lo tanto, el valor de la firma de m es s = 144516444

c) Para poder hacer la verificación de la firma anterior, realizamos los siguientes cálculos

```
sage: M=YA^k*K^s
print M
542102987
sage:N=R(alfa^m)
print N
542102987
```

Dado que los valores de M y N coinciden, la firma se daría por válida.

5. Utilizaremos el software SAGE. Entramos los datos del ejercicio:

```
sage: p=124540019
sage: q=17389
sage: q=10083255
sage: x=12496
sage: k=9557
sage: m=5246
```

Ahora ya podemos calcular:

a)
```
sage: n=(p-1)/q
print n
7162
sage: alfa=mod(g^n,p)
print alfa
57574454
sage: y= alfa^x
print y
33942608
```

Para la firma de m calculamos:

```
sage: r=mod(alfa^k,q)
print r
13752
sage: s=mod((m+x*r)/k,q)
print s
9137
```

Por lo tanto, la firma (r,s) = 13752,9137) coincide con la dada.

b) Para simular la verificación de la firma procedemos:

```
sage: w=mod(1/s,q)
sage: uno1=mod(m*w,q)
sage: uno2=mod(r*w,q)
sage: v=mod(alfa^uno1*y^uno2,q)
print v
13752
```

Dado que el valor de v = 13752 coincide con el valor de r, la firma se daría por válida.

6. La polarización en que A ha enviado los mencionados bits debe ser coherente con la que ha recibido B, puesto que estos bits no han sido desestimados. Por lo tanto A ha enviado:

\ / — |

La polarización que ha utilizado Ernesto en el bit 13 es $\oplus$ y en el bit 15 $\otimes$. En los bits 14 y 16 puede haber utilizado $\oplus$ o $\otimes$ indistintamente.

Bibliografía

Domingo, J.; Herrera, J.; Rifà-Pous, H. (2006). *Criptografía*. Barcelona: UOC.

Rifà, J.; Huguet, L. (1991). *Comunicación digital: Teoría matemática de la información. Codificación algebraica. Criptología*. Barcelona: Ed. Masson.

Simmons, G. J. (1992). *Contemporary cryptology: the science of information integrity*. New York: IEEE Press Inc.