VICENTE CALABUIG BENEYTO

Licenciado en Ciencias Matemáticas
Catedrático de Instituto
Profesor de Matemáticas en el C. Universitario-Farmacia
del CEU San Pablo de Valencia

ELEMENTOS DE CRIPTOLOGÍA

Lección magistral leída en la apertura del curso 1999-2000

FUNDACIÓN UNIVERSITARIA SAN PABLO CEU
VALENCIA
1999

De esta edición
se han impreso
600 ejemplares
numerados del 1 al 600

EJEMPLAR 572

Edita:
SERVICIO DE PUBLICACIONES
Fundación Universitaria San Pablo CEU Valencia

Ilustración cubierta:
Juan García González

Diseño y maquetación:
Cristina Ríos / Ana Isabel Molins
Servicio de Publicaciones
CEU San Pablo de Valencia

ISBN: 84-95219-12-3
Depósito Legal: V. 2.954 - 1999

Printed in Spain - Impreso en España

Artes Gráficas Soler, S.L. - La Olivereta, 28 - 46018 Valencia - 1999

ÍNDICE

Firma Digital

Timestamping

Secreto compartido

Pruebas "interactivas" y "de conocimiento cero"

Dinero Digital

Necesidad de la criptografía

Estructura básica de Internet

Para entender mejor la necesidad de la criptografía moderna, es necesario entender la **estructura básica de Internet**.

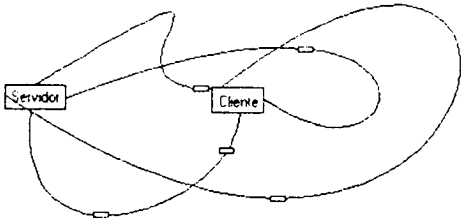
Durante la década de los 60 el Departamento de Defensa de los EEUU temía que un enemigo destruyera su red de comunicaciones atacando al **nodo principal**.

RAND, una famosa organización de investigación, tuvo la idea de eliminar el concepto de **autoridad centralizada**. Para lo cual ideó una estructura de una agudeza asombrosa:

- Los mensajes serían fragmentados en paquetes.
- Cada uno de ellos sería encaminado por una ruta adecuada para el momento. Si el paso por un nodo no fuese posible, se desviaría a otro.
- Finalmente, en el nodo de destino, se iría recomponiendo el mensaje a medida que se fueran recibiendo los paquetes integrantes.

Este esquema tiene unas propiedades importantes:

- Todos los nodos son iguales.
- Resulta sencillo añadir otros nuevos, incluso nuevas redes a la red principal.
- Los mensajes no tienen un camino prefijado, pudiendo seguir la ruta más inimaginable y pasar por nodos completamente desconocidos tanto para el emisor como para el receptor.



Tal es la estructura básica de Internet (literalmente "entre redes") y el sistema de convenios (**protocolos**) que realiza la fragmentación y encaminamiento de la información se llama **TCP/IP** (*Transport Control Protocol / Internet Protocol*, es decir Protocolo para Controlar el Transporte y Protocolo de Internet).

Internet es un medio inseguro

A la vista del diagrama que esquematiza la división y encaminamiento de los paquetes en que se descompone todo mensaje a transmitir por Internet, resulta fácil imaginar unos cuantos ataques elementales (ateniéndonos únicamente en lo que concierne a la transmisión):

- El servidor puede ser suplantado.
- El cliente puede ser adulterado.
- Los paquetes pueden ser retenidos, enviados a un competidor, falseados, etc.
- Los paquetes pueden ser curioseados por cualquier espía interesado.

Todos ellos hacen necesarias tres aspiraciones de cualquier usuario de la red: la *confidencialidad*, la *seguridad* y la *integridad*.

El procedimiento para lograrlas, a la vez que la piedra angular que las garantiza, es la **criptografía**.

Noción de criptografía

Antes de proseguir es necesario aclarar que, con el tiempo, las líneas divisorias entre lo que es y lo que no es criptografía han ido desdibujándose.

Aunque muchas personas siguen pensando que, debido a la contracción de las palabras griegas -κρυπτός oculto, γραφω escribir-, la criptografía se ocupa de la privacidad en las comunicaciones. (Y, realmente, tal

ha sido su tarea a lo largo de una gran parte de su historia). Hoy, tal actividad, sólo es una parte de la criptografía llamada **encriptación**[1].

La criptografía se ocupa, además, de problemas tan fundamentales como:

i. La **autentificación**.
ii. La **firma digital**.
iii. El **timestamping**, necesario para controlar el acceso a un disco compartido, a una instalación de alta seguridad, a un canal de TV de pago, etc.
iv. El **dinero electrónico**.
v. La demostración de que se posee cierta información sin desvelarla, es decir, **problemas de conocimiento cero**.
vi. El **secreto compartido**, a saber, el modo de cómo compartir fragmentos de un secreto de manera que con un subconjunto de tales porciones se pueda reconstruir el secreto completo.

Dada la multitud de sus tareas, en adelante procuraremos matizar las acciones criptográficas mediante tres conceptos mucho menos difusos: **criptosistema**, **criptoanálisis** y **criptología**.

Según se deduce de una etimología elemental:

- **Criptosistema** o **sistema criptográfico**, será cada una de las técnicas ideadas para el enmascaramiento de información.
- Llamaremos **criptoanálisis** es el estudio de cómo derrotar (o romper) los criptosistemas.
- Por último, la **criptología** será la disciplina que se ocupa de los **criptosistemas** y del **criptoanálisis**, combinados.

El mensaje original se llama **texto plano**[2] o **texto original** (si bien algunos autores utilizan el término **texto en claro**) y el texto encriptado recibe el nombre de **texto cifrado**[3] o, también, **criptograma**.

---

[1] Curiosamente en algunas culturas esta palabra (que tampoco recoge el diccionario de la Real Academia) constituye un vocablo de mal gusto, debido a que el término "criptos" evoca en ellas reminiscencias a las criptas de los enterramientos.
[2] *Plaintext* en inglés.
[3] *Ciphertext*, en inglés.

La **desencriptación**[4], es la acción inversa de la **encriptación**, consistente en recuperar los datos originales a partir del **texto cifrado**.

Aunque en la vida social se suele contar un viejo chiste diciendo de las personas algo torponas que son de ideas "pocas pero fijas". Aquí consideraremos necesario tener **conceptos básicos** "pocos" (siempre lo son) pero bien asentados.

Antes de ocuparnos de las tareas criptográficas actuales (recordemos: *autentificación, firma digital, timestamping, secreto compartido, conocimiento cero y dinero digital*) digamos que:

- Para su realización se emplean **protocolos**.
- Para la implementación de los protocolos son necesarios **algoritmos**.
- Para entender los algoritmos son imprescindibles algunos conocimientos matemáticos, lógicos e, incluso, lingüísticos.

Conceptos previos

La operación binaria XOR

Su comportamiento es muy fácil de describir:

Cuando opera sobre bits proporciona un resultado igual a 1 si los operandos son diferentes y un 0 si son iguales. Así:

$$1 \oplus 1 = 0 \oplus 0 = 0$$
$$1 \oplus 0 = 0 \oplus 1 = 1$$

Dicho de otra manera:

$$E_K(M) = C$$

[4] Palabra que tampoco recoge el diccionario de la Real Academia.

Para comprender mejor el papel que juega la operación XOR en la criptografía vamos a considerar un sencillo ejemplo:

Supongamos que queremos encriptar el acrónimo CEU.

Los ordenadores no entienden los abecedarios, por eso, utilizan los códigos ASCII de los caracteres "C", "E" y "U", o sea 65, 69 y 85. Pero tampoco emplean el sistema decimal, por lo que ellos leen 0100 0011, 0100 0101, 0101 0101.

Supongamos que la clave es 1011 1110, que denotaremos por K.

Un modo muy sencillo de encriptar el texto CEU sería calcular la operación:

$$CEU \oplus K$$

CEU =	0100	0011	0100	0101	0101	0101
K =	1011	1110	1011	1110	1011	1110
C =	1111	1101	1111	1011	1110	1011

Si llamamos C al texto cifrado, tenemos C = 1111 1101 1111 1011 1110 1011, o también:

$$CEU \oplus K = C$$

Para desencriptarlo basta volver a utilizar la misma clave:

$$C \oplus K = CEU$$

C =	1111	1101	1111	1011	1110	1011
K =	1011	1110	1011	1110	1011	1110
CEU =	0100	0011	0100	0101	0101	0101

Esto es debido a que cualquier texto XOReado consigo mismo da 0. Luego la clave verifica que:

$$K \oplus K = 0$$

Y, en consecuencia:

$$C \oplus K = (CEU \oplus K) \oplus K = CEU \oplus (K \oplus K) = CEU$$

Teoría de la Complejidad

Proporciona una metodología para analizar la **complejidad computacional**. Nos viene a decir si un algoritmo tiene posibilidad de ser roto en un tiempo razonable.

La complejidad de un algoritmo está determinada por la potencia de cómputo necesaria para ejecutarlo. Para medirla se utilizan dos magnitudes:

T = Complejidad Temporal.
S = Complejidad Espacial.

Tanto una como otra se expresan en función del tamaño del texto de entrada, que denotaremos por "n".

El orden de magnitud de la complejidad computacional se escribe utilizando una O mayúscula. Se define como el término de la función de complejidad que crece con mayor rapidez, a medida que la n se hace cada vez mayor.

Por ejemplo, si la complejidad temporal de un determinado algoritmo es $4n^2 + 7n + 12$, diremos que su complejidad computacional es del orden de n^2, y escribiremos $O(n^2)$

Esta notación nos permite saber cómo afecta el tamaño de la información que se introduce tanto al tiempo de ejecución como al espacio necesario para ella.

Un algoritmo es **constante** si su complejidad es independiente de n. En cuyo caso se escribe O(1).

Un algoritmo es **lineal** si su complejidad es O(n) (Léase "del orden de n"). Del mismo modo se pueden definir los algoritmos **cuadráticos**, **cúbicos**, etc. Todos estos algoritmos son **polinómicos**.

Los algoritmos cuyas complejidades son $O(t^{f(n)})$, siendo t una constante mayor que la unidad y f(n) una función polinómica de n, se llaman **exponenciales**. El subconjunto de los algoritmos exponenciales cuyas complejidades son $O(c^{f(n)})$, donde c es una constante y f(n) es más que una constante pero menos que lineal, se llaman **superpolinómicos**.

A medida que crece n, la complejidad temporal de un algoritmo puede determinar si el algoritmo es practicable. La siguiente tabla muestra los tiempos de ejecución para diferentes clases de algoritmos para "n" igual a un millón:

Clase	Complejidad	Tiempo
Constante	O(1)	1 μsg.
Lineal	O(n)	1 sg.
Cuadrática	$O(n^2)$	11.6 días
Cúbica	$O(n^3)$	32.000 años
Exponencial	$O(2^n)$	$10^{301.006}$ veces la edad del universo

La ejecución de algoritmos exponenciales es fútil, no importa lo bien que extrapolemos la potencia de cálculo, procesamiento en paralelo o contacto con alienígenas inteligentes.

Complejidad de problemas

La teoría de la complejidad también se ocupa de la complejidad de los problemas, no sólo de los algoritmos utilizados para resolverlos.

La teoría estudia el espacio y tiempo mínimos requeridos para resolver la instancia más dura de un problema, sobre un ordenador teórico llamado **máquina de Turing**. Se demuestra que ésta es un modelo realista de computación.

Los problemas que pueden resolverse con algoritmos de tiempo polinómico se llaman **tratables** debido a que usualmente se resuelven en una cantidad de tiempo racional (siempre que la información introducida sea razonable[5]). Los problemas que no pueden ser resueltos en tiempo polinómico se llaman **duros** o **intratables**.

Y lo peor de todo, Alan Turing demostró que algunos problemas son **indecidibles**. Es imposible diseñar un algoritmo para resolverlos, independientemente de la complejidad temporal del algoritmo.

Los problemas pueden dividirse en clases según la complejidad de sus soluciones. Las que interesan a los criptógrafos son:

[5] La definición exacta de "razonable" depende de la circunstancia.

- La clase **P** contiene de todos los problemas que pueden ser resueltos en tiempo polinómico.
- La clase **NP** incluye de todos los problemas que pueden ser resueltos en tiempo polinómico pero sobre máquinas de Turing no deterministas[6].

La importancia de la clase NP para los criptógrafos es la siguiente: Muchos algoritmos simétricos y todos los de clave pública pueden ser rotos en tiempo polinómico no determinista. Dado un texto cifrado C, el criptoanalista simplemente conjetura un texto plano X, y una clave k, y en un tiempo polinómico ejecuta el algoritmo de encriptación sobre X, con la clave k, y examina si el resultado es igual a C. Esto es teóricamente importante, porque acota superiormente la complejidad del criptoanálisis para ese algoritmo. En la práctica, evidentemente, es un algoritmo en tiempo polinómico determinístico lo que el criptoanalista busca.

Es evidente que la clase NP contiene a la P, porque cualquier problema resoluble en tiempo polinómico sobre una máquina de Turing determinista, también lo es sobre una no determinista (la etapa de la conjetura puede ser, simplemente, omitida).

La cuestión de si es verdad o no que P = NP es uno de los problemas sin resolver más importantes de las matemáticas relacionadas con la ciencia del cálculo. Hasta ahora se ha progresado muy poco en su solución.

Otro concepto importante es el de problema **NP-completo**. Ciertos problemas NP se dice que son NP-completos si pueden *reducirse* (esto es, *transformarse*) en cualquier otro problema, también NP, en tiempo polinómico.

Si pudiera resolverse algún problema NP-completo en tiempo polinómico, entonces todos los problemas NP podrían ser resueltos en tiempo polinómico y P = NP, en cuyo caso la mayor parte de lo que vamos a decir sería irrelevante.

En efecto, muchas clases de cifradores son trivialmente rompibles en un tiempo polinómico no determinista, pero si P = NP, entonces serían rompibles por algoritmos determinísticos.

---

[6] Variedad de una máquina de Turing normal pero que puede hacer conjeturas. La máquina supone la solución del problema y examina tal hipótesis en tiempo polinómico.

Ejemplos de problemas NP-completos:

1. El problema del viajante.- Un viajante tiene que visitar n ciudades, ¿hay alguna ruta que le permita visitar todas las ciudades una sola vez?

2. El problema de los matrimonios.- En una habitación hay n mujeres, n hombres y n clérigos (sacerdotes, rabinos, pastores, etc.). También hay una lista de matrimonios aceptables, la cual consta de un hombre, una mujer y un clérigo dispuesto a oficiar la ceremonia. ¿Es posible disponer n matrimonios tales que cada persona esté o bien casándose con una persona u oficiando en un matrimonio?

3. Satisfacibilidad de tres variables.- Hay una lista de n sentencias lógicas con tres variables. Por ejemplo:

$$(x \wedge y) \Rightarrow z), (x \wedge w) \vee \neg z \quad ,$$
$$((\neg u \wedge \neg x) \vee (u \vee \neg x)) \Rightarrow ((\neg z \wedge u) \vee x)$$

¿Hay alguna asignación de verdad para todas las variables que satisfaga todas las sentencias?

Aritmética modular

Recordemos que una congruencia entre números enteros del tipo

$$a \equiv b \pmod{n}$$

significa que a es igual a b más un múltiplo de n:

$$a = b + k \cdot n$$

Si a no es negativo y $0 < b < n$, podemos pensar que b es el resto de la división entera de a por n.

Las congruencias dan origen a la operación "**a mód n**", que llamaremos **reducción modular**, la cual denota el resto que da "*a*" al ser dividido por "*n*".

La aritmética modular se comporta como la normal: Es *conmutativa*, *asociativa* y *distributiva*.

Los criptólogos usan mucho los cálculos módulo n porque calcular raíces cuadradas o logaritmos pueden ser problemas duros. Además, la aritmética modular resulta más fácil en ordenadores porque restringe el rango de todos los valores intermedios.

Factorización

El problema de la factorización es uno de los más antiguos de la teoría de números y es considerado como uno de los problemas *duros* que antes mencionábamos.

Factorizar un número significa encontrar sus factores primos. Por ejemplo:

$$2^{113} - 1 = 3391 \cdot 23279 \cdot 65993 \cdot 1868569 \cdot 1066818132868207$$

Dado que muchos algoritmos criptográficos necesitan números primos, es fácil que se susciten algunas dudas:

1. Si hay tanta necesidad de ellos, ¿no se agotarán? La respuesta es decididamente negativa. Hay aproximadamente 10^{151} números primos de 512 bits o menos. Pensemos que 10^{84} es el volumen del universo en cm^3 (excluyendo la materia negra).

2. ¿Qué pasaría si dos personas accidentalmente escogieran el mismo número primo? No sucedería nada. Con 10^{151} números primos para escoger, las probabilidades de que esto suceda son significativamente menores de que su ordenador ardiera espontáneamente en el mismo momento en que le tocase la lotería.

3. Si alguno crease una base de datos de números primos, ¿no podría emplearla para romper algoritmos? Si, pero es irrealizable. Si pudiéramos almacenar un gigabyte de información sobre un gramo de disco duro, entonces la lista de los primos de 512 bytes pesaría tanto que excedería el límite de Chandrasekar y el disco duro se convertiría en un agujero negro.

Pero si factorizar números es muy dificultoso, ¿cómo podemos generar números con facilidad? En la práctica la respuesta es muy simple: Se generan números aleatorios y, luego, se examina si son primos.

Hay varios tests de primalidad probabilísticos, que determinan si un número determinado es primo con cierto grado de confianza. Mencionaremos los tests de **Salovay-Strassen**, **Lehmann** y **Rabin-Miller**.

En el mundo real la generación de números primos sigue el siguiente proceso:

1. Se genera un número aleatorio, **p**, de n bits.
2. Se establecen a 1 los bits de mayor y menor significación. (El bit más significativo puesto a 1 garantiza que el número tendrá los bits requeridos, y activar el bit de menor significación asegura que es impar).
3. Se examina la divisibilidad para todos los números primos menores que 2000.
4. Se utiliza algún test de primalidad apropiado. (Generalmente se ejecuta el test de **Rabin-Miller** cinco veces como mínimo).

Logaritmos discretos en un campo finito

Es fácil evaluar la expresión

$$a^x \ mód\ n$$

Pero el problema inverso: Calcular el exponente x tal que:

$$a^x \equiv b \quad (mód\ n)$$

es un problema duro. Incluso puede no tener solución. Por ejemplo, es fácil ver que $3^x \equiv 7 \ (mód\ 13)$ no tiene solución entera.

La seguridad de muchos algoritmos de clave pública se basan en el problema de encontrar logaritmos discretos.

Puede demostrarse que *la complejidad de encontrar logaritmos discretos es esencialmente el mismo que el de factorizar un entero del mismo tamaño.*

Funciones hash

Una función hash (H) es una transformación que toma un mensaje de entrada (m), o ***preimagen****, y devuelve una cadena de tamaño fijo (h), llamada* ***valor hash*** *o* ***digest****.*

H: m α h ⇒ h = H(m)

Podemos imaginar que un *digest* es la *huella dactilar* que deja un mensaje.

Si dos mensajes diferentes tienen un mismo digest (esto es, dejan la misma "huella dactilar") se produce lo que se llama una **colisión**.

Ejemplo: Una función hash muy simple podría ser una que tome una pre imagen y devuelva el XOReado de todos sus bytes.

Ya que las funciones hash son "muchos a uno", no podemos utilizarlas para saber con certeza si las dos pre imágenes son iguales, pero nos proporcionan confianza razonable de veracidad.

Las funciones hash han sido usadas por los científicos durante mucho tiempo, pero cuando se emplean en criptografía, se les exige que sean **unidireccionales**. Este concepto es central en la criptografía de clave pública, constituyendo un bloque de construcción fundamental para muchos de los protocolos que discutiremos después.

Recibe el nombre de función ***unidireccional*** *aquella que es relativamente fácil de computar, pero significativamente más dura de invertir.*

Esto es, dado x es fácil de calcular f(x), pero dado f(x) es muy difícil de hallar x. Recordemos que en este contexto *duro* significa algo así como: "*Se necesitarían millones de años para calcular x a partir de f(x), incluso si todos los ordenadores del mundo estuviesen dedicados al problema*".

La función del ejemplo anterior no es unidireccional: Dado el valor de un byte particular, es muy fácil obtener muchas cadenas que se XOReen en dicho byte.

Una buena función hash unidireccional también tiene la propiedad de estar **libre de colisiones**, es decir, es duro generar dos pre imágenes con el mismo valor hash.

Las funciones hash son públicas, su proceso no se mantiene en secreto. Su seguridad descansa en su unidireccionalidad.

*Una **función "trampa" unidireccional** es un tipo especial de función unidireccional que tiene una "puerta trasera" (trapdoor) secreta de modo que, si se conoce, la función resulta fácil de invertir.*

Un reloj es una buena prueba de ello: Es muy fácil de desmontar, pero difícil de reconstruir para alguien que no sepa cómo hacerlo. Sin embargo, si se conoce el secreto de su montaje, resulta fácil realizarlo.

Redundancia

Una limitación criptológica importante de los lenguajes naturales es la **redundancia**, es decir, las limitaciones que la gramática impone para obtener palabras significativas.

Por poner un ejemplo evidente, es manifiesto que en castellano, después de una "q" no puede escribirse una "m".

Shanon sugirió la siguiente fórmula para medirla:

$$D_n = \log_2 \frac{V'^{\,n}_{m}}{V'^{\,n}_{m,H}}$$

Siendo:

m	Número de letras que tiene el alfabeto. (En el caso del castellano, 27).
n	Número de letras que tienen las agrupaciones que estamos considerando.
$V'^{\,n}_{m}$	(Leer "variaciones con repetición de *m* letras tomadas de *n* en *n*"). Es el número de palabras posibles de "*n*" letras..
$V'^{\,n}_{m,H}$	Es el número de palabras del conjunto anterior que tienen significado.

Así pues, la fórmula de Shanon nos dice que la redundancia refleja la proporción entre el número total de palabras y el de aquellas que sean significativas.

Se demuestra que, cuando hay redundancia, el cociente:

$$\frac{V'^{n}_{m,H}}{V'^{n}_{m}}$$

decrece a medida que se incrementa "n". Por lo tanto, para algún valor de n este expresión valdrá 1. Dicho valor se llama distancia de unicidad y en muchos casos, es una cantidad muy importante para el criptoanalista, pues representa *la cantidad de texto cifrado que debe poseer para tener suficiente confianza en realizar su criptoanálisis.*

En inglés la **distancia de unicidad** es **K/6.8**, siendo K la longitud de la clave y el 6.8 una medida de la redundancia del inglés.

Para DES (uno de los algoritmos más conocidos) la distancia de unicidad es 56/6.8 = 8.2, esto significa que si intentamos descifrar, mediante la fuerza bruta (esto es, probando todas las claves posibles hasta encontrar la adecuada) un texto cifrado con DES, necesitaremos dos bloques de texto cifrado. Descifraremos el primer bloque con una clave tras otra. Si encontramos un texto descifrado que se parezca al inglés, entonces descifraremos el segundo bloque con la misma clave. Si también se parece al inglés, podemos decir que hemos localizado la clave correcta.

Estructuras básicas

Con las herramientas que acabamos de ver ya podemos examinar algunos algoritmos. Comenzaremos por los más elementales haciendo un poco de historia (pero muy poca).

La encriptación es muy antigua, pues fue impulsada por cuestiones militares, religiosas y comerciales.

- Los primitivos egipcios la practicaron desde el momento en que el pueblo utilizaba la lengua **demótica** mientras que los sacerdotes empleaban la **jeroglífica** (o **hierática**) incomprensible para el

resto. Mas recientemente los americanos realizaron algo parecido utilizando la difícil lengua de los indios navajos, si bien esto no es criptografía en sentido estricto.

- Los escritores judíos ocultaban sus textos haciendo sustituciones simétricas de las letras de su *alifato*, esto es, reemplazando la primera letra por la última (y viceversa), la segunda con la penúltima, y así sucesivamente. Este criptosistema lo denominaron **atbash**.

- Los **éforos** de Esparta (5 magistrados que, teóricamente, controlaban el poder real) se comunicaban con sus generales mediante mensajes escritos en bandas de cuero que se enrollaban formando una espiral sobre un bastón llamado **escítalo**. Naturalmente, una vez desenrollada la tira de cuero, sólo se podía leer la información si se volvía a enroscar sobre un escítalo del mismo diámetro.

Podríamos extendernos durante varias horas (incluso escribir todo un libro) mencionando curiosos y asombrosos criptosistemas -así, la Edad Media y el Renacimiento son abundantes en ellos -, pero hoy el universitario encontrará mucho más interesante trascender las anécdotas y estudiar la base y funcionamiento de todas ellas. De paso, algunos ejemplos, nos ayudarán a manejar el vocabulario recién introducido.

En 1949 **Shanon** en su *Teoría de las comunicaciones secretas* (publicada en la *Revista Técnica de los Sistemas Bell*, pág. 656 - 715), para evitar la redundancia de las fuentes, sugirió dos métodos básicos que frustrasen un criptoanálisis estadístico:

- La **confusión**, que trata de ocultar la relación entre el texto plano y el criptograma (o texto cifrado).
  Su recurso más simple es la **sustitución**, consistente en cambiar un símbolo por otro.

- La difusión, encargada de esparcir la redundancia difundiéndola por todo el texto.
  Su táctica mas elemental es la **transposición**, consistente en desordenar las unidades que constituyen el texto plano.

Quizá algunos ejemplos aclaren mejor su significado.

Sustituciones

Como su nombre indica, una **sustitución** reemplaza los caracteres del texto plano[7] por otros.

Existen dos tipos de sustitución la **monoalfabética** y la **polialfabética**.

La primera es aquella que establece una correspondencia entre caracteres que se conserva a lo largo de todo el mensaje.

El ejemplo más conocido de cifrado monoalfabético es el **Algoritmo César**.

Sabido es que Cayo Julio César mandaba mensajes a sus generales cambiando las letras del texto por las situadas n lugares más a su derecha en el alfabeto.

Históricamente César utilizaba n = 3, es decir cada letra era sustituida por otra que ocupaba (alfabéticamente) tres lugares más avanzados a su derecha.

Pero el sistema César permite tomar n = 5, o cualquier otro número entero incluyendo valores negativos.

Como más adelante veremos, esta peculiaridad se expresa en el argot criptográfico diciendo que la clave de este criptosistema es n.

El criptoanálisis de los textos (suficientemente largos) cifrados con sustituciones monoalfabéticas es muy simple: *Basta realizar un análisis de frecuencias y utilizar la permanencia estadística de los símbolos.*

Una explicación curiosa de este procedimiento aparece en la novela *El escarabajo de oro* de Edgar Allan Poe (gran aficionado a la criptografía). Para aquellos que no la hayan leído, su explicación es sencilla y breve.

Si sabemos que en nuestro idioma la frecuencia de las letras es:

E = 17%, A = 12%, O = 9%, L = 8%, S = 8%, N = 7%, D = 7%, ...

Al considerar un texto cifrado mediante una sustitución monoalfabética, lo más probable es que la letra de mayor frecuencia sea una E o una A, el siguiente carácter más repetido

[7] Esto es, del **mensaje inicial**.

corresponderá, seguramente, a una A o una O, y así sucesivamente.

NOTA. Algunos criptoanálisis hacen intervenir, además, la frecuencia con la que se presentan las letras iniciales o finales de cada palabra, o la reiteración de los grupos de dos caracteres, etc.

Las **sustituciones polialfabéticas** son aquellas en las que la asignación de caracteres varía en función de la posición de éste en el texto plano. Se realiza tomando varias sustituciones monoalfabéticas y aplicándolas cíclicamente.

El caso más paradigmático de este procedimiento es el **Cifrado Vigenère** que comprenderemos mejor con un ejemplo.

Tomemos varios criptosistemas monoalfabéticos, que caracterizaremos por sus claves. Digamos {3,1}. Pues bien, el primer carácter del texto plano será sustituido por aquel que esté situado tres posiciones a su derecha en el alfabeto, el siguiente por su inmediato, el siguiente volverá a ser encriptado por una sustitución César y así sucesivamente hasta completar todo el mensaje.

Así CEU se transformará en FFX

Como muestra este ejemplo, la *sustitución polialfabética* dificulta el análisis estadístico. No obstante su criptoanálisis resulta bastante sencillo. El número de elementos de la clave puede estimarse a partir de la periodicidad de patrones comunes que pueden aparecer en el texto cifrado. Una vez conocido éste, basta con efectuar tantos análisis estadísticos independientes como indique la cantidad estimada.

Transposiciones

Como resulta fácil de deducir, una transposición no sustituye unos símbolos por otros, sino que reordena los caracteres del texto plano para incrementar su ilegibilidad.

Pueden citarse numerosos ejemplos, desde el simple procedimiento de escribir el mensaje al revés, hasta el uso de complicados esquemas matriciales.

Ejemplos:

1. "EIDUTSELEETNERFOSUR" puede leerse al revés para entresacar las palabras RUSO, FRENTE, EL y ESTUDIE. Ordenadas de nuevo en sentido inverso, se obtiene ESTUDIE EL FRENTE RUSO.

2. Teniendo en cuenta el *cuadrado mágico de Durero* (una disposición numérica en filas y columnas con la propiedad de que tanto ellas como las diagonales suman 34), a saber:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

El criptograma:

.	C	L	C
U	L	E	V
A	D	E	N
E	A	I	E

Se convierte en "EL CEU DE VALENCIA.".

3. Una mezcla de scitalo y transposición puede ser el criptografiado del siguiente aforismo: "DEBEMOS VIAJAR EN LA DIRECCIÓN DE NUESTRO MIEDO."

   Dado que tenemos 48 caracteres, los escribimos en una tabla de 7 filas y 7 columnas:

1	2	3	4	5	6	7
D	E	B	E	M	O	S
	V	I	A	J	A	R
	E	N		L	A	
D	I	R	E	C	C	I
O	N		D	E		N
U	E	S	T	R	O	
M	I	E	D	O	.	

Si, como clave, escogemos una permutación cualquiera de las columnas, como (2, 4, 6, 1, 3, 5, 7), el criptograma sería:

EVEINEIEA EDTOOAAC O. D DOUMBINR SEMJLCEROSR IN

El final de lo que pudiéramos llamar la "**Era de los criptosistemas clásicos**" se produce con el criptoanálisis de la máquina cifradora ENIGMA utilizada por el ejército alemán durante la II Guerra Mundial.

En 1923 el ingeniero alemán **Arthur Scherbius** patentó una máquina de rotores por la que pronto se interesó el ejército alemán. Una vez mejorada se la llamó ENIGMA y fue utilizada de inmediato e intensivamente.

En 1928 Polonia recibió por accidente un ejemplar por correo ordinario, que fue enfáticamente reclamado por el Gobierno alemán despertando, por ello, las sospechas de los servicios secretos polacos, los cuales desmontaron y volvieron a ensamblar la máquina antes de devolverla.

Gracias al conocimiento de sus mecanismos internos, tres matemáticos: **Marian Rejewski, Jerzy Rozycki** y **Henry Zygalski** lograron descubrir su funcionamiento, construyendo otra máquina -a la que llamaron *Bomba*- que permitía el descifrado automático.

Una mejora alemana posterior, consistente en añadir dos rotores más a los tres ya existentes, obligaba a emplear 60 "*Bombas*" para la desencriptación, número imposible de costear por las autoridades polacas.

Dada la inminencia de la invasión de Polonia, el equipo de Rejewski salvó todo lo pudo en varios camiones forzados en su huida a cruzar

Rumania e Italia. Las adversas circunstancias de su viaje, les obligaron a tener que quemar los camiones uno a uno hasta que el último consiguió llegar a París, ciudad en la que consiguieron colaborar con siete españoles expertos en criptografía bajo las órdenes de un tal **Camazón**.

Cuando Alemania invadió Francia el equipo de Rejewski tuvo que escapar a España y posteriormente a Inglaterra, pero allí ya no se les consideró seguros al haber estado en contacto con el enemigo, confiándoles trabajos de menor importancia.

Entre tanto **Alan Turing**, director del proyecto ULTRA en el que participaba también **Von Neuman**, impulsaba la creación de una nueva máquina más evolucionada y rápida basada en los estudios del equipo polaco.

Esta larga historia tiene connotaciones desagradables para España. Es poco conocido el echo de que Alemania regaló al régimen de Franco varias máquinas ENIGMA que fueron utilizadas para comunicaciones secretas, es de suponer que ante la sonrisa de los equipos de contraespionaje extranjeros.

Finalizaremos diciendo que este cifrador es el que emplea el programa **crypt** de **Unix**, luego no debería ser utilizado por los usuarios de este sistema operativo.

La criptología actual

Hoy la criptología se realiza con ordenadores. Los científicos del ya mencionado proyecto ULTRA emplearon lo que podríamos considerar el primer ordenador, aunque este hecho permaneció en secreto hasta los 70.

El trabajo con computadoras requiere un vocabulario especificado que, en lo referente a la criptología, recordaremos brevemente.

Un **algoritmo criptográfico**, también llamado **cifrador**, es la función matemática usada para encriptar y desencriptar. (Generalmente son dos funciones relacionadas).

Si la seguridad de un algoritmo se basa en mantener secreto el modo en el que trabaja, entonces se llama un algoritmo **restringido** (o, también, **limitado**).

Los algoritmos **restringidos** tienen un interés histórico pero, lamentablemente, son inadecuados para los estándares de hoy:

- Un grupo grande o cambiante de usuarios no puede usarlos porque cada vez que un usuario abandona el grupo todos los demás deben cambiarse a un algoritmo nuevo.
- Si, accidentalmente, alguien revela el secreto todos deberán volver a cambiar de algoritmo.
- Incluso peor. Los algoritmos restringidos no permiten controlar la calidad o la estandarización. Cada grupo de usuarios debe tener su propio y único algoritmo, de lo contrario cualquier entrometido puede comprar el mismo producto y aprender el algoritmo. Además, tienen que escribir sus propios algoritmos e implementaciones. Si nadie en el grupo es un buen criptógrafo, no sabrán si su algoritmo es seguro.

A pesar de estos importantes defectos, los algoritmos restringidos son enormemente populares para las aplicaciones de baja seguridad (los usuarios o no se dan cuenta o no se preocupan de los problemas de seguridad inherentes a su sistema.

La criptografía moderna resuelve este problema mediante **claves**, que denotaremos por $\mathbf{K_i}$.

Las claves son números binarios pertenecientes a un gran conjunto de valores llamado **espacio de claves**.

Los algoritmos que emplean una misma clave para el cifrado y el descifrado se dice que son **simétricos**.

Es costumbre utilizar las letras E para simbolizar una función genérica de encriptación, D para denotar otra de desencriptación, y escribir la clave K como subíndice. De acuerdo con este convenio escribiremos:

$$E_K(M) = C \qquad D_K(C) = M$$

Estas funciones tienen, evidentemente, la propiedad:

$$D_K(E_K(M)) = M$$

Aquellos algoritmos que emplean una clave para encriptar y otra distinta para desencriptar, reciben el nombre de **asimétricos**, o de **clave pública**, porque una de ellas (a la que llamaremos **clave pública**) puede ponerse en conocimiento de cualquier persona.

Ahora deberemos escribir:

$$E_{K_1}(M) = C \quad D_{K_2}(C) = M \quad D_{K_2}\left(E_{K_1}(M)\right) = M$$

Ataques criptográficos

Un criptoanálisis intentado se llama un ***ataque.***

Una suposición fundamental en criptoanálisis -enunciada en primer lugar por **Dutchman A. Kerckhoffs** en el s. XIX-, es que **el secreto debe residir enteramente en la clave.**

Por lo tanto Kerckhoffs supone que el **criptoanalista tiene detalles completos del algoritmo criptográfico y así como de su implementación.**

Esto significa que:

- El algoritmo puede publicarse y analizarse.
- Los productos que usan el algoritmo pueden ser producidos en masa.
- No importa si un espía conoce nuestro algoritmo, mientras no conozca nuestra clave particular, no podrá leer nuestros mensajes.

Si bien esta afirmación puede parecer muy fuerte y, evidentemente, es de suponer que la CIA no tiene la costumbre de comentar sus algoritmos criptográficos con el Mossad. Sin embargo no está demás suponer que el Mossad, probablemente, los habrá encontrado por algún procedimiento.

Aunque los criptoanalistas del mundo real no tienen siempre una información tan detallada, es bueno presumirlo. Si otros no pueden rom-

per un algoritmo, incluso conociendo cómo trabaja, entonces es seguro que no podrán romperlo cuando lo desconozcan.

Tipos de ataques

Hay cuatro tipos generales de ataques criptoanalíticos. (recordemos que el criptoanalista tiene siempre un conocimiento completo del algoritmo de encriptación utilizado).

1. **Ataque sólo con texto cifrado**. El criptoanalista tiene el texto cifrado de varios mensajes, los cuales han sido cifrados todos usando el mismo algoritmo de encriptación. La tarea del criptoanalista es recuperar el texto plano de tantos mensajes como sean posibles o, mejor todavía, deducir la clave (o claves) usadas para encriptar los mensajes, en orden a desencriptar otros mensajes encriptados con las mismas claves

Dados: $C_1 = E_K(P_1)$, $C_2 = E_K(P_2)$, ... , $C_i = E_K(P_i)$

Deducir: O bien los textos originales P_1, P_2,... P_i ;
o bien la clave K ;
o bien un algoritmo para inferir P_{i+1} de $C_{i+1} = E_K(P_{i+1})$

2. **Ataque conociendo textos originales**. El criptoanalista tiene acceso no sólo a los textos cifrados de varios mensajes, sino también a los textos planos de los mismos. Su tarea consiste en deducir la clave (o claves) usadas para encriptar los mensajes, o inferir un algoritmo que permita desencriptar cualquier nuevo mensaje encriptado con la misma clave (o claves).

Dados: P_1, $C_1 = E_K(P_1)$, P_2, $C_2 = E_K(P_2)$,... P_i, $C_i = E_K(P_i)$

Deducir: O bien la clave K ;
o bien un algoritmo para inferir P_{i+1} de $C_{i+1} = E_K(P_{i+1})$

3. **Ataque con texto original escogido**.- El criptoanalista no sólo tiene acceso al texto cifrado y al texto plano asociado de varios mensajes, sino que puede escoger el texto plano para obtener el encriptado corres-

pondiente. Este ataque es más potente que el ataque anterior (con textos planos conocidos) porque el criptoanalista puede escoger bloques de textos planos específicos para encriptarlos, lo cual puede facilitar más información acerca de la clave. Su tarea consiste en deducir la clave (o claves) usada para encriptar los mensajes o inferir un algoritmo para desencriptar cualesquiera nuevos mensajes encriptados con la misma clave (o claves).

Dados: $P_1, C_1 = E_K(P_1)$, $P_2, C_2 = E_K(P_2)$,... $P_i, C_i = E_K(P_i)$
donde el criptoanalista puede escoger $P_1, P_2, \ldots P_i$.

Deducir: O bien la clave K ;
o bien un algoritmo para inferir P_{i+1} de $C_{i+1} = E_K(P_{i+1})$

4. **Ataque con texto original adaptativo escogido**. Es este un caso especial de ataque con texto plano escogido. El criptoanalista no sólo puede escoger el texto plano que está encriptado sino que también puede modificar su elección basándose en los resultados de encriptaciones previas. En un ataque con texto plano escogido, el criptoanalista puede escoger un gran bloque de texto plano para encriptarlo; en un ataque adaptativo puede escoger un bloque más pequeño de texto plano y después otro basándose en los resultados del primero, y así sucesivamente.

Hay al menos otros tres tipos de ataque criptoanalista:

5. **Ataque con texto cifrado escogido**. El criptoanalista puede escoger diferentes textos cifrados para desencriptarlos y tiene acceso a los textos planos desencriptados. Por ejemplo, el criptoanalista tiene acceso a una caja "tamper proof" (resistente al saboteo) que realiza automáticamente la desencriptación. Su tarea consiste en deducir la clave.

Dados: $C_1, P_1 = D_K(C_1)$, $C_2, P_2 = D_K(C_2)$,... $C_i, P_i = D_K(C_i)$

Deducir: La clave K.

Este ataque es aplicable en primer lugar a algoritmos de clave pública y será discutido en la Sección 19.3. Un ataque con texto cifrado escogido es a veces efectivo contra un algoritmo simétrico.

En algunas ocasiones se juntan un ataque con texto original escogido y otro con texto cifrado escogido, esta unión se conoce como **ataque con texto escogido**.

6. **Ataque con clave escogida**. Este ataque no significa que el criptoanalista pueda escoger la clave, sino que tiene algún conocimiento sobre cómo se relacionan las diferentes claves. Es un ataque extraño y oscuro, que no suele resultar muy práctico.

7. **Criptoanális de "manguera"**. El criptoanalista amenaza, extorsiona o tortura a alguien hasta que le da la clave. El soborno es a veces referido como un ataque de **compra de clave**. Todos ellos son ataques muy potentes y frecuentemente constituyen el mejor modo de romper un algoritmo.

Los *ataques con texto original conocido* y los *ataques con texto original escogido* son más comunes de lo que podemos pensar. No es un caso sin precedente para un criptoanalista obtener el texto original de un mensaje que ha sido encriptado o sobornar a alguien para que encripte un mensaje escogido. Muchos mensajes tienen comienzos y finales estándares que pueden ser conocidos por el criptoanalista. Para los programadores, el código fuente encriptado es especialmente vulnerable debido a la aparición regular de palabras clave como **#define**, **structure**, **else**, **return**. El código ejecutable encriptado posee la misma clase de problemas: funciones, bucles, estructuras, etc. Los ataques con texto original (e incluso los ataques con texto original escogido) fueron utilizados con éxito tanto contra alemanes como contra japoneses durante la Segunda Guerra Mundial.

Algoritmos simétricos

Hay dos tipos básicos de algoritmos simétricos:

- **Cifradores por bloques**
- **Cifradores por flujo**

Con un cifrador por bloque, un mismo texto plano siempre se encripta en el mismo texto cifrado (si se usa la misma clave). Con un cifrador por flujo un mismo bit se encriptará en un bit distinto cada vez que se realice la encriptación.

*Un **block cipher** (cifrador por bloque) es un tipo de algoritmo de encriptación de clave simétrica que transforma un bloque de datos de longitud fija de **texto plano**[8] (texto sin encriptar) en un bloque de **texto cifrado**[9] (texto encriptado) de la misma longitud.*

La desencriptación se realiza aplicando la transformación inversa al bloque de texto cifrado usando la misma clave secreta. La longitud prefijada se llama el **tamaño del block**.

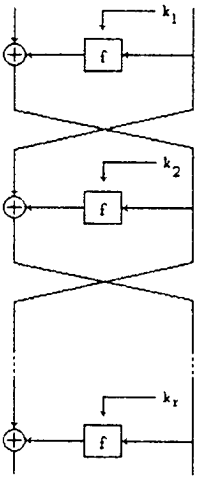
```
Un tamaño frecuente es 64 bits, pero en los próximos años
se incrementará a 128 bits a medida que los procesadores sean
más sofisticados.
```

Cifradores producto

El concepto de **cifradores producto** se debe, una vez más a Shanon, el cual eligió este nombre para designar aquellos *cifradores que encriptan un bloque de texto plano mediante un proceso en varias etapas*. En cada una de ellas, se aplica la misma transformación (conocida como **round function**) a los datos usando una subclave. El conjunto de subclaves se deduce normalmente de la clave secreta proporcionada por el usuario mediante una función especial. Este conjunto de subclaves recibe el nombre de **key schedule**.

---

[8] En inglés *plaintext*.
[9] En inglés *ciphertext*.



El número de etapas de un cifrado iterativo depende del nivel de seguridad deseada y la consecuente renuncia al rendimiento. En muchos casos, un incremento del número de pasos mejorará la seguridad ofrecida por el *cifrador por bloques*, pero para algunos *ciphers* el número de etapas requerido para conseguir una seguridad adecuada no será demasiado grande para que el cifrado sea práctico o deseable.

Los **cifradores Feistel** son una clase especial de cifrador por bloques en los que el texto cifrado se calcula del texto plano mediante una aplicación repetida de la misma transformación o *round function*.

En un *cifrador Feistel*, el texto que está siendo encriptado se divide en dos mitades. La *round function*, *f*, es aplicada a una de las mitades usando una subclave y la salida es XOReada con la otra mitad. A continuación las dos mitades son intercambiadas. Cada etapa sigue el mismo patrón, excepto la última en la que no tiene lugar el intercambio.

Un *cifrador Feistel* es un cifrador *simétrico*, luego la encriptación y desencriptación son estructuralmente idénticas, aunque las subclaves utilizadas durante la encriptación de cada etapa se toman en orden inverso.

Es posible diseñar cifrados iterativos que no sean Feistel, sin embargo la encriptación y desencriptación son estructuralmente las mismas. Un ejemplo es IDEA.

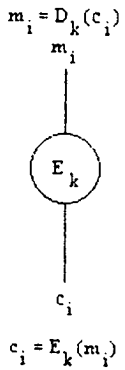
Modos de funcionamiento

Combinan:

- El cifrador básico.
- Alguna clase de retroalimentación.
- Algunos operadores simples.

Modo ECB (Electronic Codebook)

Es el modo mas obvio de usar un cifrador por bloques: Cada bloque de texto es encriptado independientemente. No es necesario encriptar un fichero linealmente, puede comenzarse con los bloques centrales, luego los finales y, por último, los iniciales. Esto es importante cuando se encriptan ficheros a los que se accede aleatoriamente como bases de datos, pues entonces cualquier registro puede ser añadido, borrado, encriptado o desencriptado independiente de cualquier otro. Además el procesamiento es paralelizable: Si tenemos varios procesadores de encriptación, pueden encriptar o desencriptar bloques distintos sin preocuparse de los demás.



Es teóricamente posible crear un diccionario de textos planos y sus correspondientes textos cifrados. No obstante, si el tamaño del bloque es de 64 bits, el diccionario tendría 2^{64} entradas (demasiado grande para almacenar).

El problema con el modo ECB es que si un criptoanalista tiene el texto plano y el correspondiente texto cifrado de varios mensajes (encriptados con la misma clave), puede comenzar tal diccionario aunque no sepa la clave.

En muchas situaciones del mundo real, hay fragmentos de mensajes que tienden a repetirse. Por ejemplo, los mensajes generados por el ordenador (como los e-mails) tienen estructuras muy regulares. Si los mensajes encriptados tienen muchas redundancias y éstas tienden a ocupar los mismos lugares en los distintos mensajes, un criptoanalista puede obtener mucha información, pudiendo montar algunos ataques estadísticos sobre el texto subyacente, sin tomar en cuenta la fortaleza del cifrador por bloques.

Esta vulnerabilidad es mayor al comienzo y final de los mensajes, en los que cabeceras y pies contienen información sobre el emisario, receptor, fecha, etc. Este problema es conocido como **comienzos y finales estereotipados**.

En el lado positivo, los errores cuando se desencripta no afectan a la totalidad del texto plano. Sin embargo, si se pierde o se agrega un bit accidentalmente, la desalineación provocará que todo el texto cifrado subsiguiente se desencripte incorrectamente.

Rellenado

Muchos mensajes no son múltiplos de 64 bits (o cualquiera que sea el tamaño del bloque). El modo ECB requiere que se rellene el último bloque con algún patrón regular de ceros y unos. Si necesitamos borrar el relleno después de la desencriptación, se agrega el número de bytes de relleno como el último byte del bloque.

Por ejemplo, supongamos que los bloques tienen 8 bytes de tamaño (64 bits) y que el último bloque consta de 3 bytes. Se requieren 5 bytes de relleno para que el último bloque tenga la longitud requerida. Entonces se añaden 4 bytes de ceros y el byte final con el número 5.

El problema más serio con el modo ECB es que un adversario podría modificar los mensajes encriptados sin conocer la clave, ni incluso el algoritmo, de tal modo que engañe al receptor pretendido.

Para ilustrar el problema, consideremos un sistema de transferencia de dinero que lo envía de unas cuentas a otras en bancos distintos. Para hacer la vida más fácil a los ordenadores de los bancos, éstos han acordado un formato estandarizado para los mensajes de transferencia de dinero que se parece al siguiente:

Banco emisario	1.5 bloques
Banco receptor	1.5 bloques
Nombre del depositario	6 bloques
Cuenta del depositario	2 bloques
Cantidad de depósito	1 bloque

Supongamos que los mensajes se encriptan empleando algún algoritmo por bloques en modo ECB y que hay un enemigo, M, oyendo las comunicaciones entre ambos bancos, que denotaremos por A y B. M puede hacerse rico del siguiente modo:

1. Prepara su ordenador para que registre todos los mensajes encriptados de A a B.
2. Abre dos cuentas a su nombre en ambos bancos y realiza dos transferencias, digamos que de 10.000 ptas. entre ellas. Como su ordenador registra todas las transferencias, busca dos mensajes que sean iguales, los cuales corresponderán a las transferencias que ha realizado.
3. Ahora, puede insertar el mensaje en las comunicaciones encadenándolo a voluntad.

Los bancos podrían fácilmente prevenir esto fechando sus mensajes:

Fecha	1 bloque
Banco emisario	1.5 bloques
Banco receptor	1.5 bloques
Nombre del depositario	6 bloques
Cuenta del depositario	2 bloques
Cantidad de depósito	1 bloque

Dos mensajes idénticos podrían detectarse fácilmente usando esta técnica.

Sin embargo, M todavía puede hacerse rico. En efecto, fijémonos en la estructura de los mensajes:

1	2	3	4	5	6	7	8	9	10	11	12	13
Fecha	Banco emisor Banco receptor			Nombre del depositario						Cuenta del depositario		Cantidad

Sólo tiene que interceptar mensajes aleatorios del banco A al B y reemplazar los bloques 5 al 12 con los bytes que corresponden a su nombre y número de cuenta. No tiene ni que saber de quién procede el dinero.

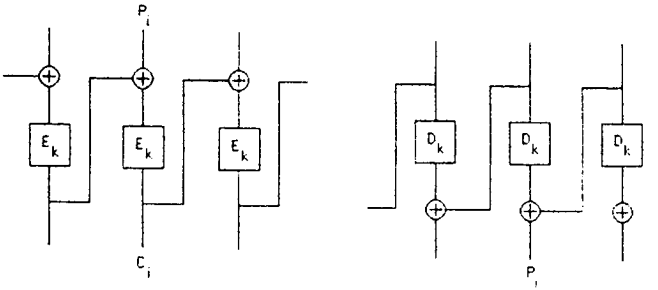
Los bancos pueden minimizar el problema cambiando las claves frecuentemente, pero esto sólo requiere que M realice sus acciones con mayor rapidez.

La solución es una técnica llamada **encadenamiento**.

Modo CBC (Cipher Block Chaining)

El encadenamiento añade un mecanismo de retroalimentación al cifrador de bloques. Así, el resultado de la encriptación de los bloques previos influye en la encriptación del bloque actual.

En el modo CBC el texto plano es XOReado con el bloque de texto cifrado previo antes de ser encriptado. La siguiente figura lo refleja:



Vector de inicialización

El modo CBC fuerza a que bloques de texto plano idénticos se encripten en bloques de texto cifrado distintos sólo si algún bloque de texto plano previo es diferente.

Dos mensajes idénticos aún se encriptan en el mismo texto cifrado. Y, aún peor, dos mensajes que comienzan del mismo modo se encriptan de la misma manera hasta la primera diferencia. Y esto puede proporcionar a cualquier criptoanalista información útil.

Esto se puede prevenir encriptando datos aleatorios en el primer bloque, el cual recibe el nombre de **vector de inicialización** (VI). El VI no tiene significado, sólo sirve para hacer que el texto cifrado sea único. Estampillar la hora y la fecha puede ser un buen vector de inicialización y, también, puede serlo cualquier número aleatorio.

Error de propagación

Brevemente, el modo CBC puede describirse como una **retroalimentación** del texto cifrado cuando se encripta, y una **prealimentación** del texto cifrado en la desencriptación.

Esta forma de comportamiento tiene implicaciones cuando existen errores. Un único Bit erróneo en un bloque de texto plano, afectará al bloque correspondiente de texto cifrado y a todos los bloques subsiguientes de texto cifrado. Esto no tiene importancia porque la desencriptación invertirá este efecto y el texto plano recuperado tendrá el mismo y único error.

Los errores en el texto cifrado son más comunes, pues pueden producirse por el ruido en las comunicaciones o un mal funcionamiento en el medio destinado al almacenamiento.

En el modo CBC, un solo bit de error en el texto cifrado afecta a un bloque y a un bit del texto plano recuperado. El bloque subsiguiente tiene un bit erróneo en la misma posición que la del error.

Esta propiedad se llama **extensión del error**. Es una molestia importante. Los bloques posteriores al segundo no se ven afectados por el error, luego el modo CBC es **auto-recuperativo** Dos bloques se ven afectados por un error, pero el sistema se recupera y continua su trabajo correctamente en todos los bloques siguientes. CBC es un ejemplo de cifrador por bloques, utilizado de una manera autosincronizada, pero sólo a nivel de bloque.

Aunque el modo CBC se recupera rápidamente de los bits erróneos, no le ocurre lo mismo con los errores de sincronía. Si desaparece o se agrega un bit al flujo de texto cifrado, entonces todos los bloques subsiguientes son movidos un bit en su posición y la desencriptación generará basura indefinidamente.

Cualquier criptosistema que use modo CBC debe asegurarse de que estructura del bloque permanece intacta, bien mediante marcos (framing) o bien almacenando los datos en trozos múltiples del mismo tamaño que el bloque.

Problemas de seguridad

La estructura de CBC provoca algunos problemas potenciales:

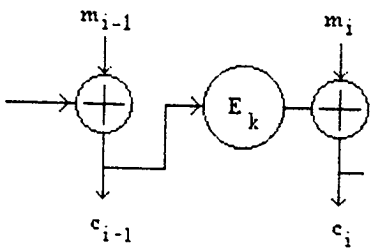
- Como un bloque de texto cifrado alfecta al siguiente, el enemigo M puede añadir bloques al final de un mensaje sin que pueda ser detectado. Seguro que al desencriptar obtendrá un galimatías, pero en algunas situaciones esta posibilidad es indeseable.
- Si se utiliza el modo CBC, se debe estructurar el texto plano de manera que se pueda saber dónde finaliza el mensaje y poder detectar la adición de bloques extra.
- M puede también alterar un bloque de texto cifrado e introducir cambios controlados en el bloque siguiente de texto plano desencriptado.
- Por ejemplo, si M cambia un solo bit en el texto cifrado, el bloque completo se desencriptará incorrectamente, pero el que le sigue tendrá un bit erróneo en la posición correspondiente.
- Hay situaciones en las que esto es deseable. El texto plano del mensaje completo puede incluir alguna clase de redundancia o autentificación controlada.
- Finalmente, aunque los patrones de texto plano son ocultados por el encadenamiento, los mensajes muy largos aún pueden conservar algunos.

  NOTA.- La paradoja del cumpleaños predice que el periodo de identidad es de $2^{m/2}$ (siendo m el tamaño del bloque). Si m = 64, esto proporciona 34 GB. Un mensaje tiene que ser verdaderamente largo antes de que le afecte este problema.

Modo CFB (Cipher FeedBack)

En el modo Cipher FeedBack (CFB) el bloque previo de texto cifrado es encriptado y la salida producida combinada con el bloque de texto plano usando XOR para producir el bloque de texto cifrado actual. Es posible definir el modo CFB para que use un feedback que sea menos que un bloque de datos completo. Se usa un vector de inicialización c_0 como "semilla" para el proceso.

$$m_i = c_i \oplus E_k(c_{i-1})$$

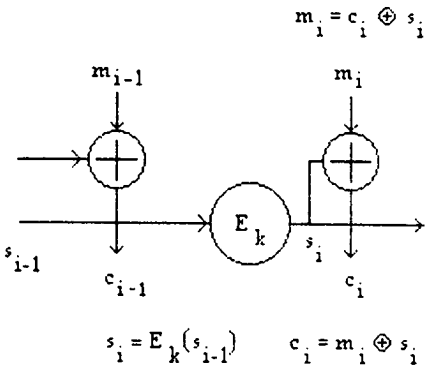


$$c_i = m_i + E_k(c_{i-1})$$

El modo CFB es tan seguro como el cifrado subyacente y los patrones de texto plano son ocultados en el texto cifrado por la operación XOR. El texto plano no puede ser manipulado directamente excepto por el removimiento de bloques del comienzo o del final del texto cifrado. Con el modo CFB y total retroalimentación (full feedback), cuando dos bloques de texto cifrado son idénticos, las salidas de la operación de cifrado en el paso siguiente también son idénticas. Esto hace que haya un escape de información. Cuando se usa total retroalimentación (full feedback), la velocidad de encriptación es idéntica a la del cifrado del bloque, pero el proceso de encriptación no puede ser fácilmente paralelizado.

Modo OFB (Output FeedBack)

El modo OFB (Output FeedBack) es similar al modo CFB excepto en la cantidad de XOR con cada bloque de texto plano es generada independientemente tanto del texto plano como del cifrado. También se utiliza un vector de inicialización s_0 como semilla para una secuencia de bloques de datos s_i, y cada uno de ellos se deriva del anterior s_{i-1}. La encriptación de un bloque de texto plano se realiza XOReando el bloque de texto plano con el bloque de datos relevante.



Las retroalimentaciones de anchura inferior a un bloque completo no son recomendables para la seguridad, el modo OFB tiene la ventaja sobre el modo CFB de que cualesquiera errores de bits que puedan ocurrir durante la transmisión no son propagados y no afectan la desencriptación de los bloques siguientes. Sin embargo, debido al encadenamiento del texto cifrado, el texto plano puede ser fácilmente manipulado. La velocidad de encriptación es idéntica a la de un cifrador por bloques. Aunque el proceso no puede ser paralelizado fácilmente, puede ahorrarse tiempo generando el flujo clave antes de que los datos estén disponibles para la encriptación.

Debido a las deficiencias en el modo OFB, Diffie ha propuesto un modo de operación adicional, llamado el modo **counter**. Difiere de OFB en la manera en que son generados los bloques sucesivos de datos para las encriptaciones posteriores. En lugar de deducir un bloque de datos de la encriptación de los previos, Diffie ha propuesto encriptar la cantidad

$$i + V.I. \quad (\text{mód } 2^{64})$$

para el bloque de datos i-ésimo.

Modo PCBC (Propagating Cipher Block Chaining)

El modo PCBC (Propagating Cipher Block Chaining) es el utilizado en protocolos como Kerberos v.4. No ha sido formalmente publicado como un estándar nacional o federal, y no tiene soporte generalmente difundido. Es una variación de CBC y está diseñado para extender o pro-

pagar un único bit de error en el texto cifrado. Esto permite que los errores en la transmisión sean capturados y que el texto plano resultante sea rechazado.

La expresión de encriptación puede escribirse:

$$c_i = E_K(m_i \oplus m_{i-1} \oplus c_{i-1})$$

Y la desencriptación se realiza calculando:

$$m_i = D_K(c_i) \oplus c_{i-1} \oplus m_{i-1}$$

siendo $m_0 \oplus c_0$ el vector de inicialización.

DES

El cifrador Feistel mas famoso de todos los tiempos ha sido (y todavía lo es) DES (Data Encryption Standard). En realidad, debíamos hablar de DEA (Data Encryption Algorithm) pero las siglas DES son tan populares, que se tienden a utilizar como sinónimos.

La interesante génesis de DES no se entiende sin saber antes qué papel juegan la NSA (Agencia Nacional de Seguridad) y el NIST (Instituto Nacional de Estándares y Tecnología).

NSA

La **NSA** es una institución altamente secreta creada por el Presidente Harry Truman en1952. Su existencia fue ocultada durante muchos años, pues su principal ocupación es la de escuchar y decodificar todas las comunicaciones extranjeras de interés para la seguridad de los EEUU. Ha utilizado su poder de varios modos para impedir o retardar la difusión de la criptografía disponible para impedir que posibles enemigos empleen métodos de encriptación demasiado fuertes para que la NSA pueda romperlos.

Como primera agencia criptográfica gubernamental, la NSA dispone de enormes recursos financieros y computacionales empleando un gran cantidad de criptógrafos. Naturalmente, sus consecuciones criptográficas

no se publican, lo cual ha desatado muchos rumores sobre su capacidad para romper populares criptosistemas (como DES), así como su influencia para obligar a los fabricantes de software a incluir "trap doors" en sus productos de manera que la agencia pueda romper fácilmente los mensajes que se encripten con ellos.

Por ejemplo, en una noticia aparecida en El Mundo, el domingo 5 de septiembre de 1999 podíamos leer que:

***Andrew Fernandes**, un cifrador y jefe de seguridad de **Cryptonym Corporation** (una empresa de seguridad cibernética canadiense) ha acusado a la compañía de Bill Gates de permitir que la NSA descifre fácilmente los mensajes secretos de los usuarios de sus ordenadores.*

El técnico ha encontrado una opción en el interior de Windows con el nombre de NSAkey.

Mientras, Microsoft negaba cualquier responsabilidad y la NSA decía que no sabía nada del asunto, miles de internautas se preguntan si toda su documentación está expuesta a los ojos de espías gubernamentales[10].

---

[10] Entre las compañías de productos criptográficos la frase "*¿Te has reunido con Lew Giles?*", es un eufemismo de "*¿Te ha pedido la NSA en secreto que debilites tus productos?*".

Se sabe que el tal *Giles* ha visitado varias compañías para pedirles que añadiesen puertas traseras a sus productos, de manera que la NSA pudiera romper fácilmente los datos cifrados por ellos.

El trato funcionaba más o menos así: Giles ofrecía un tratamiento preferente para la exportación si se añadía la puerta trasera, la cual podría ser tan sutil que no se viera en el diseño. Algo que pudiera confundirse con un error en el software. Quizá podría debilitar el generador de números aleatorios, revelar unos pocos bits de la clave, etc.

Nada de esto debería sorprendernos. Parece que la NSA ha hecho todo lo posible para añadir puertas traseras a los productos criptográficos. Destruyeron completamente a la compañía suiza **Crypto AG**, y durante al menos medio siglo, han estado interceptando y descifrando todos los documentos de alto secreto de la mayoría de los gobiernos mundiales.

En Canadá ocurre algo muy parecido, salvo que el personaje visitador se llama aquí *Norm Weijer*.

El URL de estas fascinantes historias es:
http://www.caq.com/CAQ/caq63/caq63madsen.html

La Organización de Investigación y Desarrollo para la Defensa (DRDO) de la India publicó una "alerta roja" contra todo el software de seguridad estadounidense. La institución advierte que todo el software americano podría tener puertas traseras.

Las actuaciones de la NSA han pasado frecuentemente de lo sublime a lo ridículo. Por ejemplo, para finalizar citaremos que los Furbys —unos muñecos de peluche que incluyen un procesador que les permite relacionarse e interactuar con los niños— han sido prohibidos por la NSA, debido a que pueden repetir lo que oyen y se temen que "la gente se los lleve a casa y empiece a hablar de información clasificada".

NIST (National Institute of Standards and Technology)

El Instituto Nacional de Estándares y Tecnología es una división del Departamento de Comercio estadounidense, que previamente se denominó Oficina Nacional de Estándares. Mediante su Laboratorio de Sistemas de Ordenadores aspira a auspiciar sistemas abiertos e interoperables que estimulen el desarrollo de la actividad económica basada en los ordenadores. NIST publica estándares y líneas maestras que sean adoptadas en todos los sistemas de ordenadores de EEUU. Los estándares oficiales se publican como FIPS (Federal Information Processing Standards).

En 1987 el Congreso aprobó el Acta de Seguridad en Ordenadores, la cual autorizó a NIST para desarrollar estándares que garantizasen la seguridad de información sensitiva aunque no clasificada en sistemas gubernamentales. Esto animó a NIST a trabajar con otras agencias gubernamentales así como con la industria privada para evaluar estándares propuestos sobre seguridad.

NIST emite estándares para algoritmos criptográficos que las agencias gubernamentales de Estados Unidos deben usar. Un gran porcentaje del sector privado a menudo también las adopta. En enero de 1977 NIST declaró a DES el estándar oficial de encriptación de EEUU y lo publicó como FIPS (Publicación nº 46). También se ocupa del AES (Advanced Encryption Standard), que será el sustituto de DES.

NIST ha sido criticado por permitir que la NSA interviniera demasiado en el establecimiento de sus estándares criptográficos. Podemos contar al respecto la siguiente anécdota:

El 30 de agosto de 1991 el NIST publicó una nota en el Registro Federal anunciando un estándar para las firmas digitales, al que se refería como DSS (*Digital Signature Standard*).

En su nota el NITS afirmaba que seleccionó el DSS después de evaluar diferentes alternativas y que había seguido las instrucciones gubernamentales del *Acta de Seguridad Informática* de 1987.

La referencia a este documento es importante porque, al promulgarlo, el Congreso buscaba dar autoridad al NIST sobre la seguridad informática civil y limitar el papel de la Agencia de Seguridad Nacional.

La nota hacía referencia explícita a la NSA y dejaba claramente implícito que el NIST había desarrollado el estándar.

En un esfuerzo para analizar el proceso federal de establecimiento de estándares, la **Computer Professionals for Social Responsability** (*Profesionales de la Informática para la Responsabilidad Social*) envió una petición al NIST solicitando los registros relacionados con el DSS.

El NIST respondió asegurando que todos los materiales relacionados con el estándar de firma digital estaban exentos de ser liberados.

Después de que la asociación presentara una demanda en el juzgado federal (para lograr la liberalización del material relacionado con el DSS), el NIST reconoció por primera vez que la mayor parte de los documentos relevantes que poseía en realidad provenían de la NSA, no del NIST. De hecho, el NIST creó sólo 142 páginas, mientras que la NSA escribió las 1138 páginas restantes.

Como respuesta a la investigación de los medios de comunicación, la NSA reconoció el papel principal que había desarrollado en la propuesta del DSS. El jefe de Política de Información de la NSA reconoció que habían evaluado y proporcionado los algoritmos candidatos, incluido el seleccionado por el NITS.

No hace falta decir que el papel de la NSA en el desarrollo del DSS levantó más de una suspicacia en los sectores comerciales y tecnológicos. La mayoría de las implementaciones actuales de firmas digitales se basan en el algoritmo Diffie-Hellman, no en el DSS.

DES (continuación)

Y hemos dicho que DES es un algoritmo simétrico, luego tanto él como la clave sirven para la encriptación y la desencriptación (salvo diferencias menores en la agenda de claves[11]).

La clave tiene una longitud de 56 bits, si bien ocupa 64, siendo el último de cada 8 un **bit de paridad**, esto es, un bit para asegurar que los 7 anteriores no contengan errores.

La clave puede ser cualquier número de 56 bits y puede cambiarse en todo momento, aunque existen unos cuantos de ellos se consideran como claves débiles, si bien pueden ser evitados fácilmente. Toda la seguridad se apoya en la clave.

En su nivel más simple, el algoritmo no es más que una combinación de las dos técnicas básicas de la encriptación: La confusión y la difusión. El elemento fundamental de DES es una combinación de estas dos técnicas (una sustitución seguida de una permutación) sobre el texto, basada en la clave. Este proceso se llama una **ronda**. DES tiene 16 rondas y aplica la misma combinación de técnicas sobre el bloque de texto plano 16 veces.

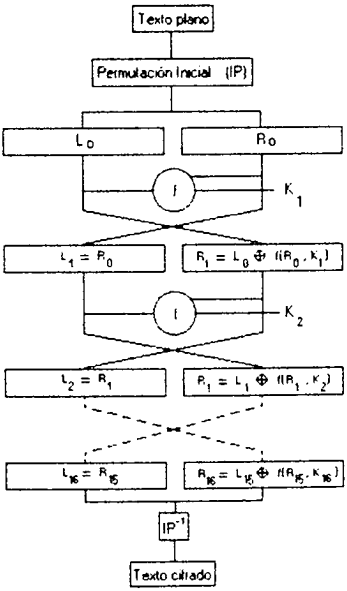
El algoritmo sólo usa operaciones lógicas y aritméticas estándares sobre números de 64 bits como máximo, luego pudo ser implementada fácilmente con la tecnología existente a finales de los 70. La naturaleza repetitiva del algoritmo lo hace ideal para ser utilizado sobre un chip de propósito especial. Las implementaciones iniciales en software fueron torpes, pero las actuales son mucho mejores.

Sipnopsis del algoritmo

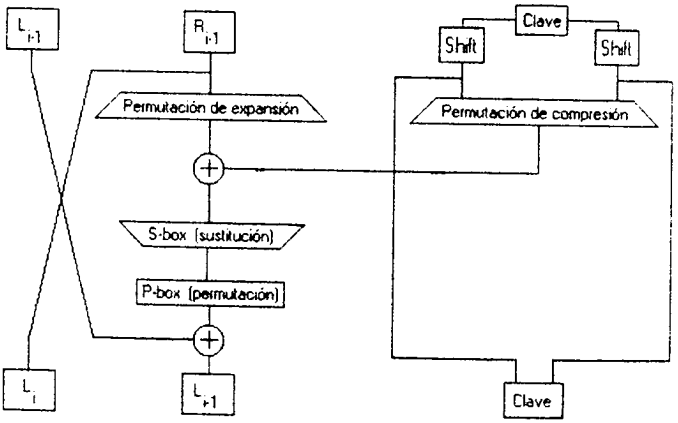
DES opera sobre un bloque de texto plano de 64 bits (8 bytes). Después de una permutación inicial, el bloque se rompe en una mitad derecha y otra izquierda. A continuación se efectúan 16 rondas de operaciones idénticas, llamadas "función f", en las que los datos se combinan con la clave. Después de la decimosexta ronda, las mitades derecha e izquierda se juntan y actúa una permutación final (inversa de la inicial) que pone fin al algoritmo.

---

[11] En inglés **Key Schedule**.



En cada ronda (v. Figura), los bits de la clave son cambiados, después de lo cual, se escogen 48 bits de los 56 bits que tiene la clave. Los 32 bits de la mitad derecha de los datos se expande hasta 48 bits mediante una permutación de expansión y son XOReados con los 48 bits permutados y expandidos de la clave, enviados a 8 S-cajas que producen 32 bits nuevos, los cuales se permutan de nuevo. Estas 4 funciones constituyen la "función f". La salida de la misma se XORea con la mitad izquierda de los datos; la antigua mitad derecha se convierte en la nueva mitad izquierda. Estas operaciones se repiten 16 veces, constituyendo las 16 rondas de DES.



Si B_i es el resultado de la i-ésima operación, L_i y R_i son las mitades derecha e izquierda de B_i, K_i es la clave de 48 bits para la ronda i, y f es la función que realiza todas las sustituciones, permutaciones y XOReaciones con la clave, entonces una ronda puede escribirse así:

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

La permutación inicial

Se realiza antes de la primera ronda. Transpone el bloque de entrada como se describe en la tabla siguiente:

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Esta tabla (como las siguientes que veremos) debe leerse de izquierda a derecha y de arriba hacia abajo. Su significado se explica mejor con un ejemplo. Así, la permutación mueve el bit 58 del texto plano a la posición 1, el bit 50 a la posición 2, y así sucesivamente.

La permutación inicial así como la correspondiente permutación final, no afectan a la seguridad de DES. (Su principal propósito es facilitar la carga del texto plano y datos del texto cifrado en un chip DES en piezas de tamaño de 1 byte. Recordemos que DES antecede a buses de 32 o 16 bits. Puesto que esta permutación de bits es difícil de realizar con software (pero trivial en hardware), muchas implementaciones de DES en software no las realizan. Aunque el algoritmo resultante no es menos seguro que DES, no debería llamarse DES porque no procede como el estándar.

La transformación clave

Inicialmente los 64 bits de la clave se reducen a 56 ignorando los últimos bits de cada 8, los cuales se pueden tomar como bit de paridad para asegurar que la clave está libre de error. Después de que los 56 hayan sido extraídos, se genera una **subclave** distinta de 48 bits para cada una de las 16 rondas de DES. Estas subclaves, K_i, se determinan de la manera siguiente:

Primeramente, la clave de 56 bits se divide en dos mitades de 28. Después estas mitades son cambiadas circularmente uno o dos bits a la izquierda, según la ronda de acuerdo con la tabla:

Ronda	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Número	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Después del cambio, se seleccionan 48 de los 56 bits. Puesto que esta operación permuta el orden de los bits y luego selecciona un subconjunto, se llama **permutación de compresión**, definida por la siguiente tabla:

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Por ejemplo, el bit en la posición 33 se mueve a la posición 35 de la salida, y el bit en la posición 18 es ignorado.

Debido a estos cambios, en cada subclave se emplea un subconjunto diferente de bits de la clave. Cada bit se usa aproximadamente en 14 de las 16 subclaves, aunque no todos los bits se usan exactamente el mismo número de veces.

La permutación de expansión

Expande la mitad derecha, R_i, del dato desde 32 a 48 bits. Como esta operación cambia el orden de los bits además de repetir ciertos bits, se la llama **permutación de expansión**. Tiene dos propósitos: hacer que la mitad derecha tenga el mismo tamaño que la clave para la operación XOR y proporcionar un resultado más largo que pueda ser comprimido durante la operación de sustitución. Sin embargo, ninguno de ellos el propósito criptográfico principal. Permitiendo que un bit afecte a dos sustituciones, la dependencia de los bits que salen de los bits que entran se extiende con mayor rapidez. A esto se le llama **efecto de avalancha**. DES está diseñado para alcanzar la condición de tener cada bit del texto cifrado dependiente de cada bit del texto plano y de cada bit de la clave tan rápidamente como sea posible.

La siguiente figura define la permutación de expansión. Esta es llamada a veces la E-box. Por cada 4 bits del bloque de entrada, el primero y cuarto representan dos bits del bloque de salida, mientras que el segundo y tercero sólo representan uno. La siguiente tabla muestra qué posiciones de salida corresponden a qué posiciones de entrada:

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Por ejemplo, el bit en la posición 3 del bloque de entrada se mueve a la posición 4 del bloque de salida, y el bit en la posición 21 se mueve a la 30 en el bloque de salida.

Aunque el bloque de salida en mayor que el de entrada, cada bloque de entrada genera un único bloque de salida.

Las S-boxes

S-box significa caja de sustitución.

Después de que la clave comprimida es XOReada con el bloque expandido, los 48 bits resultantes se someten a un proceso de sustitución. Este se realiza mediante 8 S-cajas, cada una de las cuales tiene una entrada de 6 bits y una salida de 4.

Así pues, los 48 bits han de ser divididos en 8 bloques de 6 bits. Cada subbloque es transformado por una S-caja distinta.

- Cada S-caja es una tabla de 4 filas y 16 columnas
- Cada elemento de la S-caja es un número de 4 bits (del 0 al 15).
- Los 6 bits que acceden a una S-box especifican la fila y la columna del elemento que debe escogerse para la salida.

Esta especificación se realiza de una manera muy particular: El primero y el sexto determinan la fila. Los 4 bits centrales definen la columna.

```
Evidentemente en software es más difícil implementar
las S-cajas en arrays de 64 elementos. Esto requiere alguna
ordenación de los mismos, pero no es difícil.
```

Por consiguiente, cada S-caja no es sino una función de sustitución de los 4 bits b_2 a b_6 que entran, en los 4 bits que salen. Los bits b_1 y b_6 seleccionan una de las cuatro funciones de sustitución disponibles en cada S-caja.

Las sustituciones que realiza cada S-caja constituyen el paso verdaderamente crítico de DES, las demás operaciones del algoritmo son lineales y fáciles de analizar. Las S-cajas, sin embargo, no son lineales y, más que ninguna otra cosa, proporcionan a DES su seguridad.

Los 8 bloques de 4 bits resultantes son recombinados en único bloque de 32 bits que se somete al proceso siguiente: la caja P-box de permutación.

La permutación P-box

La P-caja es la matriz:

16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25

la cual aplica cada bit de entrada (recordemos que hay 32) en una posición de salida. Ningún bit se usa dos veces y tampoco se ignora ninguno.

Por ejemplo, el bit 21 se mueve a la posición 4, mientras que el bit 4 se mueve a la 31, etc.

Finalmente, el resultado de la permutación P-box es XOReado con la mitad izquierda del bloque inicial de 64 bits. Después de lo cual, las mitades derecha e izquierda son intercambiadas y comienza otra de ronda.

La permutación final

Es la inversa de la permutación inicial y se describe en la siguiente tabla:

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

Notemos que las mitades derecha e izquierda no son cambiadas después de la última ronda, sino que la concatenación $R_{16}L_{16}$ forma un bloque que constituye la entrada de la permutación final. No hay nada que comentar aquí, salvo decir que es así para que el algoritmo pueda ser utilizada para encriptar y desencriptar.

Desencriptación con DES

Después de todas las sustituciones, permutaciones, XORes y rotaciones de bits podemos pensar que el algoritmo de desencriptación es completamente diferente y tan confuso como el algoritmo de encriptación.

Todo lo contrario, las diversas operaciones fueron escogidas para que el algoritmo sirva tanto para encriptar como desencriptar.

La única diferencia es que las claves deben emplearse en orden inverso. El algoritmo que genera las claves utilizadas en cada ronda también es circular. La rotación de bits se hace ahora a la derecha y el número de posiciones desplazadas es ahora 0, 1, 2, 2, 2, 2, 2, 2,1, 2, 2, 2, 2, 2, 2, 1.

Modos de DES

FIPS PUB 81 especifica cuatro modos de operación: ECB, CBC, OFB y CFB. El banco de estándares ANSI especifica ECB y CBC para la encriptación y CBC y CFB de n bits para la autentificación.

Debido a su simplicidad, ECB se usa más a menudo en productos comerciales de software, aunque es el más vulnerable a los ataques. CBC se emplea ocasionalmente aunque sólo es ligeramente más complicada que ECB y proporciona mucha más seguridad.

Implementaciones de DES en hardware y software

Se ha escrito mucho sobre el tema. Por ahora el actual poseedor del récord para el chip DES más rápido es un prototipo desarrollado en Digital Equipment Corporation. Soporta los modos ECB y CBC y se basa en una array GaAs de 50.000 transistores. Los datos pueden ser encriptados y desencriptados a razón de un gigabit por segundo, lo cual se traduce en 16.8 bloques por segundo.

El chip DES más impresionante es el 6868 de VLSI (previamente llamado Gatekeeper (portero)). No sólo puede realizar encriptación DES en tan sólo 8 ciclos de reloj (hay prototipos en el laboratorio que pueden hacerlo en 4), sino que puede realizar triple-DES ECB en 25 ciclos de reloj, y triple-DES OFB o CBC en 35 ciclos de reloj.

Una implementación software de DES sobre un mainframe IBM 3090 puede ejecutar 32.000 encriptaciones DES por segundo. Muchos microcomputadores son más lentos, pero aún así impresionan.

Seguridad de DES

Hace mucho tiempo que la gente ha cuestionado la seguridad de DES. Ha habido mucha especulación sobre la longitud de la clave, número de iteraciones y diseño de las S-cajas. Estas últimas son particularmente misteriosas (todos esos números sin ninguna razón aparente de su presencia). Aunque IBM declaró que el funcionamiento interno es el resultado de 17 años de criptoanálisis intensivo, algunos temen que la NSA incrustó una puerta trasera (trapdoor) en el algoritmo para poder desencriptar los mensajes fácilmente.

El Senate Select Committee sobre Intelligence investigó el asunto en 1978. Las conclusiones del comité están clasificadas, pero un sumario no clasificado de tales conclusiones exoneraba a la NSA de cualquier implicación inadecuada en el diseño del algoritmo. Sin embargo, puesto que el Gobierno nunca ha publicado los detalles de la investigación, mucha gente no está convencida.

Tuchman y Meyer, dos de los criptógrafos de IBM que diseñaron DES, han indicado que la NSA no hizo nada para alterar el diseño:

> Su estrategia básica fue examinar la fortaleza de las funciones de sustitución, permutación y listas de claves... IBM ha clasificado las notas que contenían los criterios de selección en la solicitud de la NSA... "La NSA nos informó que nosotros habíamos reinventado inadvertidamente algunos de los profundos secretos que ella utiliza para realizar sus propios algoritmos", explica Tuchman.

Más tarde en el artículo Tuchman apostilla "Nosotros desarrollamos el algoritmo DES enteramente dentro de IBM. La NSA no fijó ni un solo alambre". Tuchman lo volvió a reafirmar cuando habló sobre la historia de DES en la National Computer Security Conference en 1992.

Por otra parte Coppersmith escribió: "La National Security Agency (NSA) también proporcionó consejo técnico a IBM", y Konheim ha dicho "Enviamos las S-boxes fuera de Washington. Cuando las volvimos a recibir eran completamente distintas. Realizamos nuestros tests y los pasaron".

La NSA, cuando se le interrogó respecto a cualquier debilidad impuesta en DES, dijo:

> Respecto al Data Encryption Standard (DES), creemos que el registro público de la investigación del Comité Senatorial para la Inteligencia de 1978 respecto al papel de la NSA en el

```
desarrollo de DES es suficiente para su pregunta. El informe
del comité indicaba que la NSA no intentó forzar el diseño del
algoritmo de ninguna manera y que la seguridad ofrecida por
DES fue mas que adecuada durante al menos 5 - 10 años, inter-
valo de tiempo para el que fue pensado. Resumiendo, la NSA no
impuso ni intentó imponer ningún tipo de debilidad sobre DES.
```

Entonces, ¿por qué hicieron modificar las S-cajas? Quizá fue para asegurar que IBM no pusiera una puerta trasera a DES. La NSA no tenía ninguna razón para confiar en los investigadores de IBM, y hubieran sido negligentes en sus obligaciones si no hubieran estado absolutamente seguros de que DES estaba completamente libre de trampillas. Imponer las S-cajas es un medio para ello.

Claves débiles

Debido al modo en que la clave inicial se modifica para obtener las subclaves de cada ronda del algoritmo, ciertas claves iniciales son **débiles**. Recordemos que el valor inicial es dividido en dos mitades y cada una de ellas rota sus bits de manera independiente. Si todos los bits de cada mitad son 0s o 1s, entonces las subclaves usadas en cualquier ciclo del algoritmo es la misma. Esto también puede ocurrir si una mitad esta compuesta por entero de 1s y la otra de 0s o viceversa. Además, de las claves débiles tienen otras propiedades que las hacen menos seguras.

Adicionalmente, algunos pares de claves encriptan texto plano en texto cifrado idéntico. En otras palabras, una clave del par puede desencriptar mensajes encriptados con la otra de cualquier otro par. Esto es debido al modo en que DES genera las subclaves; en lugar de generar 16 claves distintas, genera sólo dos cada una de las cuales se utiliza 8 veces en el algoritmo. Estas claves se llaman **semidébiles**.

Algunas claves sólo producen 4 subclaves, cada una de las cuales se emplea 4 veces en el algoritmo. Se llaman **posiblemente débiles**.

Antes de condenar a DES por presentar estas debilidades, consideremos que estas 64 claves es minúscula comparado con el conjunto total de $72.057_{5}94.037_{1}927.936$ claves posibles. Si seleccionamos una clave al azar, las posibilidades de que sea una de las claves débiles es insignificante.

Claves complementarias

Tomemos el complemento binario de una clave (es decir, reemplacemos los 0s con 1s y viceversa). Ahora, si la clave original encripta un bloque de texto plano, la complementaria encriptará el complemento del bloque de texto plano en el complemento del bloque de texto cifrado.

Si x' es el complemento de x, podemos escribirlo así:

$$E_K(P) = C$$
$$E_{K'}(P') = C'$$

No tiene ningún misterio. Las subclaves son XOReadas con la mitad derecha después de la permutación de expansión en cada ronda. Esta **propiedad de complementación** es un resultado directo de este hecho.

Esto significa que un ataque con texto plano escogido contra DES sólo tiene que testear la mitad de las claves posibles; 2^{55} en lugar de 2^{56}. Eli Biham y Adi Shamir probaron que hay un ataque con texto plano conocido de la misma complejidad, que requiere al menos 2^{33} textos planos conocidos.

Es cuestionable que esto sea una debilidad, ya que muchos mensajes no tienen bloques complementarios de texto plano (en un texto plano aleatorio, las probabilidades contra ello son extremadamente altas) y los usuarios pueden ser advertidos de no usar claves complementarias.

Criptoanálisis de DES

Criptoanálisis Diferencial

En 1990, Eli Biham y Adi Shamir introdujeron el **análisis diferencial** contra DES, que dio mejores resultados que el ataque por la fuerza bruta.

Es este un ataque de texto plano escogido en el que el criptoanalista examina pares de textos cifrados, cuyos textos planos correspondientes tienen diferencias particulares especialmente escogidas. Analizando la evolución de cómo se propagan tales diferencias a través de las 16 rondas, se obtienen claves más probables que otras. A medida que se analizan

más y más pares de textos cifrados, una clave emerge como la más probable. Ésta es la clave correcta.

El ataque funciona también contra otros algoritmos similares que también tengan S-boxes fijas y contra cualquiera de sus modos de operar (ECB, CBC, etc.). El mejor ataque diferencial contra la variante DES de 16 rondas requiere 2^{47} textos planos escogidos.

Veamos algunos puntos importantes:

- El ataque es muy teórico. Las enormes demandas de tiempo y datos para montar un ataque diferencial están más allá del alcance de casi cualquiera. Es necesario encriptar 1.5 megabits/sg. durante casi tres años.
- Puede convertirse en un ataque de texto plano conocido, pero tenemos que tamizar todos los pares (texto plano, texto cifrado) para escoger los que sean útiles, lo cual lo hace *menos eficiente* que el ataque por la fuerza bruta.

Se conviene que DES todavía se puede considerar como seguro contra el criptoanálisis diferencial.

Criptoanálisis Lineal

Inventado por Mitsuru Matsui, usa aproximaciones lineales para describir la acción de un cifrador por bloques.

Esto significa que si XOReamos los bits de un texto plano, XOReamos los bits del texto cifrado correspondiente y, por último, XOReamos ambos resultados, obtendremos un bit que es el XOR de algunos de los bits de la clave. Esto es lo que se llama una aproximación lineal, la cual da una probabilidad p. Si $p \neq \frac{1}{2}$, esta discriminación puede ser explotada.

Contra del DES completo de 16 rondas este ataque puede recuperar la clave en un promedio de 2^{43} textos planos conocidos. Implementaciones en software recuperan una clave en 56 días usando 12 estaciones de trabajo HP9000/735.

El criptoanálisis lineal es mucho más nuevo que el diferencial y puede dar mejor rendimiento en los próximos años, pero no está claro que pueda ser usado efectivamente contra DES, aunque funciona muy bien con versiones reducidas (con menos rondas).

"Caída" de DES

En julio de 1998 la Fundación de Fronteras Electrónicas (EFF) construyó la máquina **Deep Crack**, un descifrador hardware que puede romper DES en un tiempo en un tiempo promedio de 4.5 días, cuyo coste fue de 220.000 dólares

La noticia no fue la inseguridad de DES, ni que pudiera construirse hardware crackeador de algoritmos, ni que una clave de 56 bits sea demasiado corta -- los criptógrafos ya lo venían diciendo durante años --, sino que el gobierno las hubiera negado muy poco antes. ha negado que tales máquinas fueran posibles. El mismo 8 de junio de 1998, Robert Litt, Procurador General del Departament de Justicia, negó que fuera posible para el FBI crackear DES. "[Es un mito que] tengamos supercomputadores que puedan crackear nada de lo que poseemos", dijo Litt. "Situemos el problema técnico en su contesto: Se necesitan 14.000 ordenadores Pentium trabajando durante 4 meses para desencriptar un solo mensaje... No estamos hablando del FBI y NSA [que necesitan potencia de cálculo masiva], estamos hablando de todos los departamentos policiales" [12].

Después de esto lo primero que se le ocurre a cualquiera es que el FBI o era incompetente, o mentía, o ambas cosas. Nadie usa Pentiums para romper DES, excepto como una demostración. Cualquiera se lo podría haber dicho al Sr. Litt.

La máquina de EFF no posee una ingeniería muy innovadora. No es el no va mas de la criptografía. Utiliza chips viejos y tecnologías aburridas, diseño hardware simple, software no muy interesante y nada de criptografía. No es una maravilla de la ingeniería; la única cosa interesante es cómo es de sencillo su diseño.

EFF se ha gastado 220.000$ en su primera máquina. Se pueden gastar otros tantos en una máquina de doble tamaño que rodará con el doble de rapidez. Ahora que el diseño básico del trabajo está realizado, las mejoras en la implementación y rendimiento pueden incrementar las prestaciones (o decrementar el precio) al menos en un factor de 5. Y la "Ley de Moore" predice que la máquina será bien dos veces más rápida o dos veces más barata en otro 18 meses.

---

[12] Ver la historia completa en www . wired . com / news / politics / story / 12830.html.

La máquina EFF rompe DES, pero podría haber sido diseñada fácilmente para romper cualquier otro algoritmo de encriptación., El ataque fue contra la longitud de la clave, no contra el diseño del algoritmo [13]. Además. Un diseño ligeramente más caro podría haber usado FGPAs, permitiendo al sistema trabajar contra varios algoritmos y sus variantes.

El EFF es un grupo civil libre y su trabajo ha sido un proyecto demostrativo. Las agencias gubernamentales como el FBI y la NSA, presumiblemente, gastarían mucho más tiempo ingeniando una solución más eficiente. Es razonable suponer que cualquier país con un presupuesto para inteligencia haya construido esta clase de máquina, probablemente uno o dos órdenes de magnitud más rápida.

Indudablemente hay muchas, muchas mejoras técnicas que pueden realizarse al diseño de EFF para hacer la búsqueda por la fuerza bruta más barata y rápida. Pero el hecho de que un grupo civil y libre pueda usar vieja tecnología para construir algo que la administración ha negado, es la verdadera noticia.

El lunes 18 de enero de 1999, la empresa **RSA Data Security Inc** anunció el "Desafío DES III" El ordenador DES Cracker de la EFF y 100.000 PCs coordinados por **Distributed Net** en sus tiempos muertos, descifraron el mensaje del concurso en tan sólo 22h y 15m.

A pesar de su caída, DES sigue siendo ampliamente utilizado en multitud de aplicaciones, por ejemplo en las transacciones de los cajeros automáticos. Mucha gente se resiste a abandonar este algoritmo porque no se sabe de ningún otro que haya resistido durante casi 20 años tantos ataques y de tantas personas significadas dentro de la criptografía.

Dado que el verdadero problema de DES no es su arquitectura, sino la cortedad de su clave, mucha gente prefiere emplear algunas variantes que se han propuesto.

La que ha obtenido más aceptación ha sido Triple-DES, la cual obedece a un esquema del tipo:

$$C = E_{K_1}(E^{-1}_{K_2}(E_{K_1}(M)))$$

[13] Ver www . counterpane . com / keylength.html.

De esta manera, se consigue una clave de 112 bits. No hay suficiente silicio en la galaxia, ni suficiente tiempo hasta que el Sol se oscurezca para romper Triple-DES por la fuerza bruta.

DES-X, con XOR adicional con bloques clave antes de la primera ronda y después de la última, añade considerable seguridad a DES y es considerablemente más barato que Triple-DES.

Otros cifradores por bloques

IDEA (International Data Encryption Algorithm)

Es la segunda versión de un cifrador por bloques diseñado y presentado por Lai y Massey. Los bloques son de 64 bits y la clave de 128. El proceso de encriptación requiere 8 etapas pero muy complejas.

Aunque no tiene una estructura Feistel, la encriptación y desencriptación se realizan de la misma manera (una vez calculadas las subclaves). Su estructura fue diseñada para facilitar la implementación (tanto en software como en hardware).

Su seguridad depende del uso de tres tipos incompatibles de operaciones aritméticas con palabras de 16 bits. Sin embargo algunas de ellas no son demasiado rápidas en software. Por ello resulta que la velocidad de IDEA en software es similar a la de DES.

Una de las preocupaciones durante su diseño fue hacerlo fuerte contra el criptoanálisis diferencial. Hoy se considera que IDEA es inmune a él. Por otra parte no hay criptoanálisis lineales en su contra ni tampoco presenta debilidad algebraica.

El resultado criptoanalítico más significativo es el debido a **Daemen** quien descubrió 251 claves débiles, sin embargo este hecho no tiene impacto sobre la seguridad del cifrador porque pueden ser detectadas fácilmente.

IDEA es considerado generalmente como un cifrador muy seguro y tanto su desarrollo como su base teórica han sido amplia y extensamente discutidos. Sin embargo su futuro no está muy claro. No ha podido ser adoptado como sustituto de DES, en parte porque está patentado y hay

que costear su licencia en aplicaciones comerciales y en parte porque todavía se está esperando ver cómo evoluciona su precio a medida que, durante los próximos años, vayan apareciendo resultados criptoanalíticos. Su fama actual se debe a que forma parte de PGP.

Skipjack

Es un algoritmo de encriptación por bloques de 64 bits, con una clave de 80 bits y 32 vueltas. Ha sido diseñado por la NSA, por lo que algunos sospechan que ésta habrá introducido deliberadamente puertas traseras.

En junio de 1998, por primera vez en la historia, la NSA lo desclasificó y lo puso a disposición del público. La difusión de Skipjack ha marcado un hito en el criptoanálisis pues, al igual que DES, va ser tomado como punto de referencia.

Los investigadores lo analizarán buscando en él las claves sobre cómo diseñar cifradores por bloques.

En agosto de 1998 un grupo de criptógrafos israelitas presentó un ataque matemático y esotérico llamado **criptoanálisis diferencial imposible**, que funciona contra una variante de Skipjack de 31 rondas.

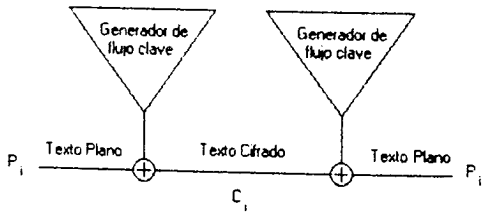
Las consecuencias de este ataque son importantes. Hay dos posibles explicaciones:

- La NSA no conocía este ataque, lo cual representa una gran victoria de los académicos contra los militares.
- La NSA si lo sabía, lo cual significa que, por alguna razón, están difundiendo algoritmos que están a punto de ser rotos.

Ambas explicaciones son fascinantes y apuntan a algún descubrimiento por venir.

Cifradores por flujo

Son aquellos que convierten texto plano en texto cifrado bit a bit. El modo más sencillo de conseguirlo se diagramatiza en la siguiente figura:



La seguridad del sistema depende por completo de los entresijos internos del **generador del flujo clave**.

- Si éste sólo generase ceros, el texto cifrado t el texto plano serían iguales, luego el cifrador sería inútil.
- Si produjese patrones iguales, digamos que de 16 bits, el algoritmo se convertiría en una operación XOR simple con una seguridad insignificante.
- Si originase un flujo interminable perfectamente aleatorio, tendríamos un criptosistema totalmente seguro.

Actualmente los generadores producen un flujo de bits que parece aleatorio pero que, en realidad, es determinístico y puede ser reproducido sin errores en la desencriptación.

Por otra parte, si el generador produjese el mismo flujo de bits cada vez que se activa, el criptosistema resultante sería trivial de romper.

En efecto, Si E tiene el texto plano a la vez que su texto cifrado correspondiente, puede XORearlos y recuperar el flujo clave.

O si tiene 2 textos cifrados distintos encriptados con el mismo flujo clave, puede XORearlos y obtener dos textos planos XOReados cada uno

con el otro. Esto es muy fácil de romper y, después, E puede XORear uno de los textos planos con el texto cifrado para obtener el flujo clave.

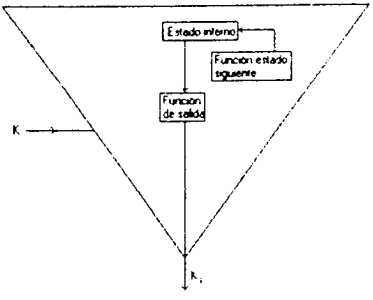
Ahora, si intercepta otro texto cifrado, ya tiene el flujo clave para desencriptarlo.

Además, puede desencriptar y leer cualquier texto cifrado antiguo que haya podido interceptar con anterioridad.

En resumen, si E tiene un par (texto plano, texto cifrado), podrá leer cualquier mensaje. Por este motivo, todos los cifradores de flujo también usan claves. Entonces, la salida del generador es una función de la clave. Así, si E obtiene un par (texto plano, texto cifrado) podrá leer todos los mensajes encriptados con una clave determinada pero sólo con ella.

Los cifradores de flujo son especialmente útiles para encriptar información sin fin, como sería el caso de dos ordenadores del CEU conectados entre sí.

Un **generador de flujo clave** tiene tres partes básicas:



- El **estado interno** describe el estado actual del generador. Dos generadores con la misma clave e idéntico estado interno producirán el mismo flujo clave.
- La **función de salida** recoge el estado interno y genera un bit del flujo clave.
- La **función "estado siguiente"** recoge el estado interno y genera otro estado interno nuevo.

La generación del *flujo clave* puede ser:

- Independiente del texto plano y del texto cifrado. Lo cual produce el llamado cifrado de flujo **síncrono**.
- Dependiente de los datos y de su encriptación. En cuyo caso el cifrado de flujo se dice que es **auto sincronizado.**

NOTA.- La encriptación de un determinado texto plano en un cifrador de bloque produce siempre el mismo resultado (si se usa la misma clave). pero con un cifrado de flujo, las transformaciones de estas unidades más pequeñas de texto plano variará según vayan siendo procesadas.

One-Time Pad

El actual interés por el cifrado de flujo se centra en las interesantes propiedades teóricas de la "one time pad", a veces llamada **cifrado Vernan**. Las características principales de la clave son:

- Es una cadena de bits generada por un proceso totalmente aleatorio.
- Ha de tener la misma longitud que el texto plano.
- Se combina con éste mediante un XOR entre bits para producir el texto cifrado.

Ya que el flujo clave es aleatorio, incluso un oponente con infinitos recursos computacionales sólo puede "adivinarlo" a la vista del texto cifrado.

Un cifrado así se dice que ofrece un **secreto perfecto**, y el análisis del *one time pad* es considerado como una de las piezas angulares de la criptografía moderna. Aunque el *one time pad* se ha usado durante la guerra en canales diplomáticos de alta seguridad, el hecho de que la clave secreta (que sólo puede usarse una vez) sea tan larga como el mensaje, introduce varios problemas de administración de claves.

Aunque perfectamente seguro, el One-Time Pad es, en general, impracticable.

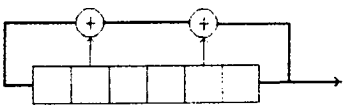
Los *cifrados de flujo* fueron desarrollados como una "aproximación" a los *one time pad*. Aunque los *cifradores de flujo* son incapaces de proporcionar la seguridad teórica satisfactoria del one time pad, por lo menos son prácticos.

Como hasta ahora ningún *cifrado de flujo* se ha convertido en un estándar de facto, el más utilizado es RC4 (v. 3.6.3).

Curiosamente, ciertos modos de operación de un *cifrado de bloque* se transforman, efectivamente, en generadores de *flujos de clave* y, así, cualquier *cifrado de bloque* puede ser usado como *cifrado de flujo*. (p. ej. DES, CFB o OFB). Sin embargo, los *cifrados de flujo*, realizados con un diseño específico propio, son mucho más rápidos.

LFSR (Linear Feedback Shift Register)

Un LFSR (Linear Feedback Shift Register) es un mecanismo que permite generar una secuencia de bits. El "registro" (abreviatura coloquial de LFSR) consta de una serie de celdas cuyo valor inicial se establece mediante un **vector de inicialización** (que en muchos casos suele ser la clave). Su comportamiento está regulado por un reloj de modo que en cada instante, el contenido de las celdas se desplaza una posición a



la derecha, dando origen a un bit de salida.

El bit "libre" de la izquierda es sustituido por el XOR de un subconjunto de celdas previamente seleccionadas, llamadas **feedback taps**.

Los LFSRs son rápidos y fáciles de implementar, tanto en hardware como en software. Con una elección juiciosa de los **feedback taps** (en la figura son los bits 2° y 5°), las secuencias generadas tienen un buen aspecto estadístico. Sin embargo, no son seguras porque ha sido desarrollado un potente marco matemático que permite su análisis simple.

Un **shift register cascade** es un conjunto de LFSRs en cascada de tal modo que el comportamiento de uno depende del de su precedente en la cascada. Este comportamiento consiste (usualmente) en emplear un LFSR para controlar el reloj del siguiente.

Por ejemplo, un "registro" puede ser adelantado un paso si la salida del precedente es un 1 y dos en caso contrario.

El **shrinkin generator** fue desarrollado por Coppersmith, Krawczyk y Mansour. Es un *cifrado de flujo* basado en la simple interacción entre las salidas de dos LFSRs. Los bits de uno se usan para determinar si los bits de salida del otro se emplearán en el flujo clave.

El *shrinkin generator* es simple y tiene buenas propiedades de seguridad. Un inconveniente es que el porcentaje de salida del flujo clave no es constante, a menos que se tomen ciertas precauciones.

Una variante del *shrinkin generator* es el **self shrinkin generator** en el que, en lugar de utilizar la salida de un LFSR para "shrink" (encoger) la salida del otro (como en el *shrinkin generator*), la salida del primer LFSR se usa para extraer bits de la salida del otro.

Todavía no hay resultados sobre el análisis criptográfico de ninguna de ambas técnicas.

RC4

Es un cifrador por flujo diseñado en 1987 por Ron Rivest para RSA Data Security Inc. Las claves son de tamaño variable y sus operaciones se realizan sobre bytes. Durante 7 años fue su propietario y los detalles del algoritmo sólo estaban disponibles para aquellos que firmaran un contrato de no exposición. RC puede significar "Ron's Code", "Rivest's Cipher", etc.

En septiembre de 1994 alguien (anónimamente) puso el código fuente en el correo electrónico de Cypherpunks. Rápidamente se esparció por el grupo de noticias Usenet y, vía Internet, a sitios ftp de todo el mundo. Los lectores con copias legales de RC4 confirmaron la compatibilidad. RSA Data Security intentó devolver el genio a su botella, asegurando que toda-

vía era un secreto comercial antes de que fuera hecho público. Pero era demasiado tarde. Ya había sido discutido y diseccionado en Usenet, distribuido en conferencias y enseñado en cursos de criptografía.

RC4 es simple de describir. El algoritmo trabaja en modo OFB (Output Feed Back):

El flujo clave es independiente del texto plano. Tiene 256 S-boxes 8×8: S_0, ... , S_{255} que son permutaciones de los números 0, 1, ... , 255 y cada permutación es función de la longitud de la clave. Tiene dos contadores i, j inicializados a cero.

Para generar un byte aleatorio, se hace lo siguiente:

```
i = (i+1)  mód 256
j = (j+1)  mód 256
se intercambian Si y Sj
t = (Si + Sj)  mód 256
K = St
```

El byte K es XOReado con el texto plano para producir el texto cifrado. La encriptación es muy rápida (casi 10 veces más que DES).

La inicialización de las S-boxes también es fácil. Primero se rellenan linealmente: $S_0 = 0$, $S_1 = 1$, ... , $S_{255} = 255$. Después se rellena otra matriz de 256 bytes con la clave, repitiendo la clave tantas veces como sea necesario hasta rellenar la matriz completamente: K_0, K_1, ... , K_{255}. Se pone a cero el índice j. Después:

```
for i =0 to 255
        j = (j + Si + Ki)  mód 256
        Swap Si y Sj
```

Y esto es todo. RSA Data Security Inc asegura que el algoritmo es inmune a criptoanálisis diferencial y lineal, no parece tener ciclos pequeños y el altamente no lineal. No hay resultados criptoanalíticos puros. RC4 puede estar en 2^{1700} estados posibles (un número enorme). Las S-boxes se desarrollan lentamente con su uso: i asegura que cada elemento cambia, y j que los elementos cambian aleatoriamente.

Sería posible generalizar esta idea para S-boxes mayores y tamaños de palabra. La versión previa es de 8 bits, pero no hay razón para que no se pueda definir un RC4 de 16 bits con S-boxes 16×16 (100K de memoria) y una palabra de 16-bits. Será necesario iterar el setup inicial 65.536 veces, pero el algoritmo resultante será más rápido.

RC4 tiene un estado legal de exportación si la longitud de su clave es de 40 bits o menor. RC4 está en docenas de productos comerciales criptográficos como Lotus Notes, AOCE de Apple Computers, Oracle Secure SQL y forma parte de la especificación Cellular Digital Packet Data.

Algoritmos de clave pública

El concepto de criptografía de clave pública fue inventado por Whitfield Diffie y Martin Hellman, e independientemente por Ralph Merkle. Su contribución a la criptografía fue el concepto de **pares de claves** -- una para la encriptación y otra para la desencriptación -- y que debe ser imposible generar una de la otra.

Diffie y Hellman presentaron primero este concepto en la National Computer Conference. Unos cuantos meses después, publicaron su trabajo "Nuevas Direcciones en Criptografía". Debido a la lentitud del proceso de publicación, la primera contribución de Merkle en este campo no apareció hasta 1978.

Desde 1976, han sido propuestos numerosos algoritmos criptográficos de clave pública. Muchos de ellos son inseguros. De los que se consideran seguros, muchos son poco prácticos, bien porque tienen una clave demasiado larga o bien porque el texto cifrado es mucho mas grande el texto plano.

Sólo algunos son a la vez seguros y prácticos. En general se basan en algunos de los "problemas duros" que hemos mencionado (factorización, logaritmo discreto, etc.). Algunos sólo son aplicables a la distribución de claves, otros a la encriptación, otros a las firmas digitales, etc.

Sólo hay tres algoritmos que funcionen bien en la encriptación y en las firmas digitales: RSA, ElGamal Y Rabin, pero son muy lentos, mucho más que los algoritmos simétricos. Usualmente son demasiado lentos para soportar la encriptación de datos voluminosos.

Seguridad de los algoritmos de clave pública

Desde el momento en que un criptoanalista tenga acceso a la clave pública, puede escoger cualquier mensaje para encriptarlo. Esto significa que un criptoanalista, dado $C = E_K(p)$, puede conjeturar el valor de P y comprobar fácilmente su suposición. Esto es un problema serio si el número de posibles textos planos es suficientemente pequeño para permitir la búsqueda exhaustiva, pero puede resolverse rellenando los mensajes con una cadena de bits aleatorios.

Es especialmente importante el caso en que un algoritmo de clave pública se use para encriptar una clave de sesión. E puede generar una BD de todas las claves de sesión posibles encriptadas con la clave pública de B. Seguramente esto requerirá una gran cantidad de tiempo y de memoria, pero para una clave exportable de 40-bits o una clave DES de 56 bits, exige mucho menos tiempo y memoria que romper la clave pública de B. Una vez E haya generado la BD, tendrá su clave y podrá leer su correo a voluntad.

Los algoritmos de clave pública están diseñador para resistir ataques de texto escogido. Su seguridad se basa tanto en la dificultad de deducir la clave privada a partir de la pública como en la de deducir el texto plano a partir del texto cifrado. Sin embargo, muchos algoritmos de clave pública son particularmente susceptibles a un ataque de texto cifrado escogido.

Un sistema en el que la firma digital es la inversa de la encriptación, este ataque es imposible de prevenir a menos que se usen claves distintas para la encriptación y las firmas.

Consecuentemente, es importante fijarse en la totalidad del sistema y no en las partes individuales. Los buenos protocolos de clave pública están diseñados para que sus diversas partes no puedan desencriptar mensajes arbitrarios generados por otras partes (los protocolos de prueba de identidad son un buen ejemplo).

RSA

De todos los algoritmos de clave pública propuestos en los últimos años, RSA es con mucho el más fácil de comprender y de implementar. (Martin Gardner publicó una antigua descripción del algoritmo en su columna de "Juegos Matemáticos" en la revista Scientific American). También es el más popular. Es el acrónimo de sus tres inventores - **Ron Rivest**, **Adi Shamir** y **Leonard Adleman** - ha resistido muchos años de extensivos criptoanálisis. Aunque los criptoanálisis nunca probaron ni desaprobaron la seguridad de RSA, sugieren un gran nivel de confianza en el algoritmo.

RSA basa su seguridad en la dificultad de factorizar números grandes. Las claves pública y privada son función de un par de números primos muy grandes (de 100 a 200 dígitos e incluso más). La recuperación del texto plano a partir de la clave pública y del texto cifrado se conjetura que es equivalente a factorizar el producto de los dos números primos.

Para generar las dos claves, se escogen dos números primos aleatorios muy grandes, p y q. Para mayor seguridad se escogen de la misma longitud. El ordenador calcula su producto:

$$n = p \cdot q$$

Después, aleatoriamente escoge la clave de encriptación, e, tal que:

$$\text{m.c.d.}(e, (p-1)(q-1)) = 1$$

Finalmente, utiliza el algoritmo extendido de Euclides para calcular la clave de desencriptación, d, de manera que:

$$e \cdot d \equiv 1 \mod ((p-1)(q-1))$$

Dicho de otra manera:

$$d = e^{-1} \mod ((p-1)(q-1))$$

Notemos que d y n también son primos relativos. Los números (e, n) constituyen la clave pública y el número d es la clave privada. Los dos primos p y q ya no son necesarios, por lo que pueden ser descartados, pero nunca revelados.

Para encriptar un mensaje, m, primero lo dividimos en bloques numéricos menores que n (con datos binarios, se escoge la mayor potencia de 2 menor que n). Es decir, si p y q son primos de 100 dígitos, entonces n estará por debajo de los 200 dígitos y cada bloque del mensaje, m, debe tener menos de 200 dígitos. (Si se necesita encriptar un número fijado de bloques, puede rellenarlos con ceros a la izquierda para asegurar que siempre serán menores que n).

El mensaje encriptado, c, tendrá bloques de tamaño similar (c_i). La fórmula de encriptación es muy simple:

$$c_i = m_i^e \mod n$$

Para desencriptar el mensaje, tomemos cada bloque encriptado c_i y calculemos:

$$m_i = c_i^d \mod n$$

Puesto que:

$$c_i^d = [m_i^e]^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i \cdot m_i^{k(p-1)(q-1)} = m_i \cdot 1 = m_i \mod n \quad \text{todo}$$

la fórmula recupera el mensaje.

El mensaje también podría haber sido encriptado con d y desencriptado con e, la elección es arbitraria.

Por ejemplo, si p = 47 y q = 71, entonces n = p·q = 3337.

La clave de encriptación, e, no debe tener factores en común con (p-1)(q-1) = 46·70 = 3220.

Supongamos que e (escogido aleatoriamente) sea 79. En tal caso:

$$d = 79^{-1} = 1019 \mod 3220$$

Este número se calcula usando el algoritmo extendido de Euclides. Se publican (e, n), se mantiene d en secreto y se descartan p y q.

Para encriptar el mensaje m = 688232687966668003, primero se divide en bloques más pequeños. Los bloques de 3 dígitos en este ejemplo son muy adecuados. Así:

$m_1 = 688$, $m_2 = 232$, $m_3 = 687$, $m_4 = 966$, $m_5 = 668$, $m_6 = 003$

El primer bloque se encripta como

$688^{79} \mod 3337 = 1570 = c_1$

Realizando las mismas operaciones con los siguientes bloques, se genera el mensaje encriptado:

c = 1570 2756 2091 2276 2423 158

La desencriptación del mensaje requiere realizar las mismas exponenciaciones pero con la clave 1019, luego:

$1570^{1019} \mod 3337 = 688 = m_1$

El resto del mensaje puede recuperarse de la misma manera.

Rapidez de RSA

En hardware, RSA es cerca de 1000 veces más lento que DES. La implementación más rápida en hardware VLSI con un módulo de 512 bits tiene un rendimiento de 64 kilobits/seg. También hay chips que ejecutan encriptación RSA de 1024 bits. Se están planeando chips que se aproximarán a 1 megabit/seg usando un módulo de 512 bits. Hay fabricantes que han implementado RSA en tarjetas inteligentes, pero tales implementaciones son más lentas.

En software, DES es unas 100 veces más rápido que RSA. Estas cantidades pueden variar ligeramente a medida que cambie la tecnología, pero RSA nunca se aproximará a la velocidad de los algoritmos simétricos.

La encriptación de RSA es mucho más rápida si se escoge un valor adecuado de e. Las tres elecciones más comunes son 3, 17 y 65537 ($2^{16}+1$). (La representación binaria de 65537 sólo tiene dos unos, luego sólo necesita 17 multiplicaciones para exponenciar). No hay problemas de seguridad si se usan cualesquiera de estos tres valores para e, incluso si un grupo de usuarios utiliza el mismo valor para e.

Las operaciones con clave privada pueden ser aceleradas con el teorema chino del resto si salvamos los valores de p y q, y valores adicionales tales como d mód (p-1), d mód (q-1) y q^{-1} mód p. Estos números adicionales pueden ser calculados fácilmente a partir de las claves pública y privada.

Seguridad de RSA

La seguridad de RSA depende totalmente del problema de la factorización de números grandes. Técnicamente esto no es verdad. Se ha *conjeturado* que la seguridad de RSA depende de tal problema. No ha sido demostrado matemáticamente que necesitemos factorizar n para calcular m de c y e. Es concebible que pueda descubrirse un modo completamente diferente para criptoanalizar RSA. Sin embargo, si tal método permite el criptoanálisis para deducir d, también podría usarse como un modo nuevo de factorizar grandes números.

Para los ultraescépticos, ha sido probado que algunas variantes de RSA son tan difíciles como la factorización. También se ha descubierto que recuperar incluso ciertos bits de información del texto cifrado en una encriptación RSA es tan duro como desencriptar el mensaje entero.

Factorizar n es el medio más obvio de ataque. Cualquier adversario dispone de la clave pública e y del módulo n. Para encontrar la clave de desencriptación, d, hay que factorizar n. Actualmente un módulo de 129 dígitos decimales es el límite de la tecnología de la factorización. Luego n debe ser más largo aún.

Es ciertamente posible para un criptoanalista intentar cada posible "d" hasta encontrar el valor correcto. Este ataque por la fuerza bruta aún es menos eficiente que el intento de factorizar n.

De vez en cuando salen personas que afirman haber encontrado modos fáciles de romper RSA, pero hasta la fecha ninguna de tales afirmaciones ha podido sostenerse. Por ejemplo, en 1993 un anteproyecto de William Payne propuso un método basado en el pequeño teorema de Fermat. Desgraciadamente este método es más lento que factorizar el módulo.

Hay otro problema. Muchos algoritmos comunes para calcular p y q son probabilísticos. ¿Qué sucede si p y q son compuestos? Bien, en primer lugar podemos hacer que la probabilidad de que esto suceda sea tan pequeña como queramos. Y si sucede, la encriptación y desencriptación no funcionarán adecuadamente. Hay unos cuantos números, llamados números de Carmichael, para los cuales fallarán ciertos algoritmos de primalidad. Estos son muy raros. No hay que preocuparse por ellos.

Ataques de texto cifrado escogido contra RSA

Algunos ataques funcionan contra las implementaciones de RSA. No son ataques contra el algoritmo básico, sino contra el protocolo. Es importante darse cuenta de que no es suficiente para dejar de usar RSA.

E escuchando a escondidas las comunicaciones de A, se las ingenia para conseguir un texto cifrado, c, encriptado con RSA mediante su clave pública. E quiere leer el mensaje.

Matemáticamente, lo que quiere es conseguir m, tal que $m = c^d$.

Para ello, escoge primero un número aleatorio, $r < n$. Como sabe la clave pública de A (e), puede calcular:

$$x = r^e \mod n$$
$$y = x \cdot c \mod n$$
$$t = r^{-1} \mod n$$

Ahora, E consigue de A que le firma "y" (con su clave privada), así desencripta y. (A tiene que firmar el mensaje, no el hash del mensaje). Recordemos que nunca ha visto "y". Así pues A envía a E:

$$u = y^d \mod n$$

Ahora E calcula:

$$t \cdot u = r^{-1} \cdot y^d = r^{-1} \cdot x^d \cdot c^d = r^{-1} \cdot r \cdot c^d = c^d = m \pmod n$$

T es un notario público. Si A desea un documento notarial, lo envía a T, éste lo rubrica con una firma digital RSA y se lo devuelve. (Ninguna función hash unidireccional ha intervenido, T encripta el mensaje entero con sus clave privada).

M quiere que T le firme un mensaje que de otra manera no podría. Posiblemente tiene un timestamp falso, o se propone ser otra persona. Cualquiera que sea la razón, T se negaría a firmarlo si lo supiese. Llamemos a este mensaje m'.

En primer lugar, M escoge un valor arbitrario x y calcula $y = x^e$ mód n. Puede obtener fácilmente "e" porque es la clave pública de T y debe estar a disposición de todos para verificar sus firmas.

A continuación calcula m = y·m' (mód n) y se lo envía a T para que lo firme. T devuelve m'^{d} (mód n) que es la firma de m'.

La debilidad que M está explotando es que la exponenciación conserva la estructura multiplicativa de la entrada:

$$(x \cdot m)^d = x^d \cdot m^d \quad (\text{mód } n)$$

E quiere que A firme m_3. Entonces genera dos mensajes m_1 y m_2 tales que:

$$m_3 = m_1 \cdot m_2 \quad (\text{mód } n)$$

Si E puede conseguir que A firme m_1 y m_2, puede calcular:

$$m_3^d = m_1^d \cdot m_2^d \quad (\text{mód } n)$$

Moraleja.- Nunca debe emplearse RSA para firmar un documento aleatorio que nos presente un desconocido. Siempre hay que utilizar una función hash primero. El formato de bloque ISO 9796 previene este ataque.

Ataque al módulo con RSA

Una posible implementación de RSA puede dar a cada uno el mismo n, pero diferentes valores para los exponentes e y d. Desgraciadamente esto no funciona. El problema mas obvio es que si un mismo mensaje es encriptado con dos exponentes distintos (teniendo ambos el mismo módulo), y estos dos exponentes son relativamente primos (que es el caso general), entonces el texto plano puede ser recuperado sin ningún exponente de desencriptación.

Sea m el mensaje y sean e_1 y e_2 las dos claves de encriptación. Los dos textos cifrados son:

$$c_1 = m^{e_1} \quad (mód\ n)$$
$$c_2 = m^{e_2} \quad (mód\ n)$$

El criptoanalista conoce n, e_1, e_2, c_1, c_2 veamos cómo puede recuperar m.

Puesto que e_1, e_2 son relativamente primos, el algoritmo extendido de Euclides puede encontrar dos números r y s tales que:

$$r \cdot e_1 + s \cdot e_2 = 1$$

Suponiendo que r sea negativo (uno de los dos tiene que serlo), el mismo algoritmo extendido de Euclides puede emplearse nuevamente para calcular c_1^{-1} tal que:

$$(c_1^{-1})^{-r} \cdot c_2^{s} = m \pmod{n}$$

Moraleja.- No hay que compartir un módulo común entre un grupo de usuarios.

Ataque contra un exponente de encriptación bajo

La encriptación y verificación de firma, en RSA, son más rápidas si se usa un valor bajo para e. Pero también puede ser inseguro. Si encriptamos e(e+1)/2 mensajes linealmente dependientes con diferentes claves públicas pero el mismo valor de e, existe un ataque contra el sistema. Si hay muy pocos mensajes, o si éstos no están relacionados, no hay problema. Si los mensajes son idénticos, entonces e mensajes bastan. La solución más sencilla consiste en rellenar los mensajes con valores aleatorios independientes. Esto asegura también que $m^e \pmod{n} \neq m^e$. Muchas implementaciones RSA (como PGP y PEM) lo hacen.

Ataque contra exponentes de desencriptación bajos

Otro ataque, debido a Michael Wiener, recupera d si es superior a un cuarto del tamaño de n y e es menor que n. Esto raramente sucede si e y d se escogen aleatoriamente, y no puede ocurrir si e tiene un valor pequeño.

Moraleja.- Hay que escoger valores grandes para d.

Recordemos que no basta tener un algoritmo criptográfico seguro. Debe serlo el sistema criptográfico completo, además del protocolo criptográfico. Un fallo en cualquiera de estas tres áreas hace inseguro todo el sistema.

Ataque contra la encriptación y firma en RSA

Tiene sentido firmar los mensajes antes de encriptarlos, pero no todo el mundo sigue esta práctica. En RSA hay un ataque contra los protocolos que encriptan antes de firmar.

A quiere enviar un mensaje a B. Primero lo encripta con la clave pública de B, después firma con su clave privada. Su mensaje encriptado y firmado puede denotarse así:

$$\left(m^{e_B} \ mód\ n_B\right)^{d_A} \ mód\ n_A$$

Veamos cómo B puede afirmar que A le ha enviado m' y no m. Démonos cuenta que, puesto que B sabe la factorización de n_B (es su módulo), puede calcular el logaritmos discretos respecto a n_B.. En consecuencia todo lo que tiene que hacer es encontrar un x tal que:

$m'^{x} = m \pmod{n_B}$

Entonces, si puede publicar xe_B como su nuevo exponente público y mantener n_B como su módulo, puede afirmar que A le ha enviado el mensaje m' encriptado con este nuevo exponente.

Es este un ataque particularmente repugnante en algunas circunstancias. Notemos que las funciones hash no pueden resolver este problema. Sin embargo, forzar un exponente de encriptación fijado para cada usuario, sí lo hace.

Estándares

RSA es un estándar *de facto* en muchas partes del mundo. ISO casi (pero no del todo) creó un estándar RSA de firma digital; RSA está en una información anexa a ISO 9796. La comunidad bancaria francesa está estandarizada sobre RSA, así como la australiana. Los EEUU actualmente no tienen estándar para la encriptación de clave pública debido a las presiones de la NSA y patentes publicadas. Muchas compañías estadounidenses usan PKCS, escrito por RSA Data Security Inc. Un anteproyecto ANSI para estandarizar la banca especifica RSA.

Patentes

El algoritmo RSA está patentado en los EEUU, pero no en otros países. PKP autoriza la patente, junto con otras patentes criptográficas de clave pública. La patente EEUU expirará el 20 de septiembre del 2000.

NOTA.- El 13 de enero de est mismo año apareció la noticia de que la estudiante irlandesa Sarah Flannery, de 16 años, ha conseguido desarrollar un algoritmo de clave publica basado en matrices 2x2, con el que se consigue un cifrado fuerte comparable a RSA y en un tiempo de proceso muy inferior (30 veces más rápido).

Su sistema puede ser aplicado de manera global a todo tipo de cifrados y, en particular, puede acarrear grandes beneficios en el terreno del comercio electrónico, donde su velocidad de proceso lo convierten en un algoritmo muy recomendado.

Sarah ha denominado a su sistema "Cayley-Purser" en honor a Arthur Cayley, un experto en matrices de Cambridge del siglo XIX, y a Michael Purser, un criptógrafo del Trinity College, en Dublín, quienes le inspiraron en su desarrollo.

Diffie-Hellman

El protocolo Diffie-Hellman o **de acuerdo de clave** (también llamado "*exponencial*") fue desarrollado por Diffie y Hellman en 1976 y publicado en *New Directions in Cryptografy*. El protocolo permite que dos usuarios canjeen una clave secreta incluso en un medio inseguro.

El sistema tiene dos parámetros **p** y **g**. Ambos son públicos y pueden ser utilizados por todos los usuarios del sistema. El parámetro **p** es un número primo y **g** (usualmente llamado **generador**) es un entero menor que p con la siguiente propiedad:

$$\forall n \ni 1 \leq n \leq p-1 \quad \exists k \ni g^k = n \ (\text{mód } p)$$

Supongamos que A y B quieren canjear una clave secreta compartida usando el protocolo de Diffie-Hellman. Procederían como sigue:

- A genera un valor aleatorio **privado** "*a*" y B genera otro "*b*" de las mismas características.
- Ambos has sido extraídos del conjunto de enteros {1, 2, ... , p-2}.
- Después deducen sus valores **públicos** a partir de sus *valores privados* y de los parámetros *p* y *g* de la siguiente manera:
- El valor público de A es g^a (mód p)
- El valor público de B es g^b (mód p)

- A continuación canjean sus valores públicos.
- Finalmente, A calcula $(g^b)^a$ (mód p) y B calcula $(g^a)^b$ (mód p).
- Puesto que $g^{ba} = g^{ab}$, ambos obtendrán el mismo resultado. Éste es la **clave** que deseaban compartir.

La seguridad del protocolo depende del *Problema del Logaritmo Discreto*, esto es, de la hipótesis de que es computacionalmente infactible calcular la clave secreta compartida $k = g^{ab}$ (mód p) dando los valores públicos g^a (mód p) y g^b (mód p) cuando el número primo p es suficientemente grande. Maurer ha probado que romper el protocolo Diffie-Hellman es equivalente a computar logaritmos discretos bajo ciertas hipótesis.

El canje de claves Diffie-Hellman es vulnerable al ataque "*Hombre-en-el-medio*". En este ataque un oponente, C, intercepta el valor público de A y envía su propio valor público a B. Cuando B transmite el suyo, C lo sustituye por el suyo y lo envía a A. Así pues, A y C han acordado una clave compartida y B y C otra. A partir de ahora C puede desencriptar cualquier mensaje enviado por A o por B y después de leerlos y, posiblemente, modificarlos antes de re-encriptarlos con la clave apropiada, puede transmitirlos a la otra parte.

Tal vulnerabilidad está presente porque en el canje de clave Diffie-Hellman no se pide la autentificación a los participantes. Posibles soluciones incluyen el uso de firmas digitales y otras variantes.

El *Protocolo Diffie-Hellman Autentificado* (o protocolo STS = *Station To Station*) fue desarrollado por Diffie, van Oorschot y Wiener en 1992 para derrotar al ataque "*Hombre-en-medio*". La inmunidad se alcanza permitiendo a las dos partes que se autentifiquen cada una a la otra usando *firmas digitales* (v. 2.2.2) y *certificados de clave pública*.

La idea es la siguiente: Antes de la ejecución del protocolo, cada una de las dos partes, A y B, obtiene un par (clave pública, clave privada) y un certificado para la clave pública. Durante el protocolo, A computa una firma sobre ciertos mensajes que abarcan el valor público g^a (mod p). B procede de un modo similar. Aunque C sea capaz de interceptar mensajes entre A y B, no puede falsear firmas sin la clave privada de A y de B.

En los últimos años el protocolo Diffie-Hellman ha sido tomado como ejemplo de una técnica criptográfica mucho más general. El elemento común es la derivación de un valor secreto compartido (la clave) desde una clave pública por una parte y una clave privada por otra. Los pares de claves pueden generarse de nuevo en cada ejecución del protocolo. Las claves públicas pueden ser certificadas para que las partes puedan ser autentificadas y pueda haber combinaciones de estos atributos. El borrador ANSI X9.42 ilustra algunas de estas combinaciones, y una reciente colaboración de Blake-Wilson, Johson y Menezes proporciona algunas pruebas relevantes de seguridad.

Algunas tareas criptográficas

Autentificación

En una comunicación por Internet no podemos estar seguros ni de la autenticidad de los participantes ni de la información que se transmite.

Supongamos que A quiere darse a conocer a un ordenador anfitrión (host):

- A le envía su contraseña.
- El anfitrión ejecuta una función hash sobre la contraseña recibida.
- Compara el valor hash con el que tiene almacenado.

Un fichero de contraseñas encriptado con una función hash todavía es vulnerable:

M compila una lista de un millón o mas de contraseñas. (Si cada contraseña tiene 8 bytes, el fichero sólo tendrá 8 MB). Sólo tiene que comparar su fichero con el del host y ver qué contraseñas coinciden.

Este ataque se llama **ataque del diccionario** y es sorprendentemente exitoso.

Salt es un modo de dificultar este ataque. Consiste en concatenar una cadena aleatoria con la contraseña antes de aplicarles la función unidireccional. Si el número de *valores salt* es suficientemente grande, el *ataque del diccionario* queda prácticamente anulado, pues M tiene que generar el hash de cada valor salt posible.

SKEY

Es un programa de autentificación que confía en una función hash.

A introduce en número aleatorio, R. El ordenador calcula

$$f(r),\ f(f(R)),\ f(f(f(R))),\ \ldots$$

Y así hasta un centenar de valores que podemos llamar x_1, x_2, ... , x_{100}. A debe guardar una lista y ponerla a salvo. El ordenador también almacena x_{101} en una BD de acceso junto al nombre de A.

La primera vez que A quiera acceder, teclea su nombre y x_{100}. El ordenador calcula $f(x_{100})$ y lo compara con x_{101} que ha almacenado, si coinciden A esta autentificado. Seguidamente, el ordenador reemplaza x_{101} con x_{100} en la BD. A elimina x_{100} de su lista.

Cada vez que A accede, introduce el último número no eliminado x_i. El ordenador calcula $f(x_i)$ y lo compara con x_{i+1} almacenado en su BD. El enemigo, E, no puede obtener ninguna información útil porque cada número sólo se usa una vez y la función es unidireccional.

Autentificación usando criptografía de clave pública

Los protocolos de prueba de identidad segura funcionan de la siguiente forma:

1. A ejecuta algunos cálculos con números aleatorios y su clave privada. El resultado lo envía al host.
2. El host devuelve un número aleatorio diferente.

3. A vuelve a realizar algunos cálculos con el número recibido y algunos otros que genera por su cuenta, además de su clave privada. El resultado lo vuelve a enviar al host.
4. El host realiza algunas operaciones con los diversos números recibidos de A y su clave pública para verificar que A conoce su clave privada.
5. Si es así, la identidad de A ha sido verificada.

Firma Digital

Las firmas manuscritas han sido utilizadas durante mucho tiempo como prueba de autoría, o al menos como consentimiento con el contenido de un documento. ¿Qué es lo que caracteriza a una firma?

1. La firma ha de ser auténtica. La firma convence al receptor del documento de que el firmante lo ha firmado intencionalmente.
2. La firma debe ser infalsificable. La firma es prueba de que el firmante, y ningún otro, ha firmado a propósito el documento.
3. La firma no es reutilizable. La firma es parte del documento. Una persona sin escrúpulos no puede mover la firma a un documento distinto.
4. El documento firmado es inalterable. Después de que un documento esté firmado, no puede ser alterado.
5. La firma no puede ser repudiada. La firma y el documento son cosas físicas. El signatario no puede negar posteriormente que el no ha realizado la firma.

En realidad, ninguna de estas afirmaciones sobre las firmas es completamente cierta. Las firmas pueden falsificarse, pueden recortarse de un papel y ser trasladadas a otro, un documento puede ser alterado después de su firma. Sin embargo, estamos viviendo con tales problemas debido a la dificultad de la falsificación y al riesgo de la detección.

Nos gustaría hacer estas mismas cosas con los ordenadores, pero surgen problemas:

- Primero, los ficheros de ordenador son triviales de copiar. Incluso si la firma de una persona fuera difícil de falsificar, puede ser

fácil de cortar y pegar una firma válida de un documento a otro documento. La presencia de una firma no significa nada.

- Segundo, los ficheros de ordenador son fáciles de modificar después de firmados, sin dejar ningún rastro de tal modificación.

Firma de documentos con criptosistemas simétricos. Terciador.

A quiere firmar un mensaje digital y enviarlo a B. Con la ayuda de un terciador de confianza, T, y un criptosistema simétrico puede llevarlo a cabo.

T es un terciador que puede comunicarse tanto con A como con B. Comparte una clave secreta, K_A, con A y otra, K_B, con B. Estas claves han sido establecidas mucho antes de que el protocolo comience y pueden ser reutilizadas muchas veces para otras firmas.

1. A encripta su mensaje para B con K_A y lo envía a T.
2. T desencripta el mensaje con K_A, toma el mensaje desencriptado y lo encripta (con K_B) junto con una declaración de que este mensaje lo ha recibido de A.
3. T lo envía, todo junto, a B.
4. B desencripta el mensaje recibido con K_B y lee tanto el mensaje de A como la certificación de T de que ha sido A quien lo ha enviado.

¿Cómo sabe T que el mensaje se lo ha enviado A y no cualquier impostor? Lo infiere de la encriptación del mensaje, ya que sólo él y A comparten la clave secreta (sólo A ha podido encriptar el mensaje con tal clave).

¿Es esto tan bueno como una forma sobre papel? Examinemos las características que deseamos:

1. Esta signatura es auténtica. T es un mediador fiable y sabe que el mensaje proviene de A. La certificación de T le sirve a B como prueba.
2. La firma es infalsificable. Sólo A (y T, pero todos confían en él) sabe K_A, luego sólo A puede haber enviado a T un mensaje en-

criptado con K_A. Si alguien intentara hacerse pasar por A, T hubiera caído en la cuenta en el paso 2 y no certificaría su autenticidad.

3. La firma no es reutilizable. Si B intenta tomar la certificación de T y adherirla a otro mensaje,
4. El documento firmado es inalterable. En cuanto B intentara alterar el documento después de recibirlo, T podría probar la mala jugada exactamente de la misma manera que hemos descrito.
5. La firma no puede ser repudiada. Incluso si A posteriormente proclama que nunca ha enviado el mensaje, la certificación de T dice lo contrario. Recordemos que T tiene la confianza de ambos y lo que él dice es la verdad.

Si B quiere mostrar a C un documento firmado por A, no puede revelarle la clave secreta de A. Lo tiene que hacer a través de T nuevamente;

a. B toma el mensaje y la afirmación de T de que el mensaje procede de A, lo encripta con K_B y lo reenvía a T.
b. T desencripta el conjunto con K_B.
c. T consulta su base de datos y confirma que el mensaje original procedía de A.
d. T vuelve a encriptar el conjunto con la clave secreta que comparte con C, K_C, y se lo envía a C.
e. C desencripta el conjunto con K_C y puede leer tanto el mensaje como el certificado de T de que es A quien lo envía.

Estos protocolos funcionan, pero le hacen consumir a T mucho tiempo. Pasará su tiempo encriptando y desencriptando mensajes, actuando como intermediario entre cada par de personas que quieran enviar documentos firmados entre ellas. Debe mantener una base de datos con todos los mensajes (aunque esto puede evitarse enviando al receptor una copia del mensaje encriptado del emisor). Será un cuello de botella en cualquier sistema de comunicaciones, incluso con un programa que lo haga automáticamente.

Todavía es más duro crear y mantener a alguien como T, alguien en quien confíen todos los usuarios de la red. T tiene que ser infalible, si comete un error nadie confiará en él. T ha de ser completamente seguro.

Si su base de datos de claves secretas se perdiera o si alguien modificara su programa, todas las firmas serían completamente inútiles.

Árboles de firmas digitales

Ralph Merkle propuso un esquema de firma digital basado en criptografía de clave secreta, que produce un número infinito de firmas one-time utilizando una estructura de árbol.

La idea básica de este esquema es situar la raíz de este árbol en algún fichero público, autentificándolo consecuentemente. La raíz firma un mensaje y autentifica sus subnodos en el árbol. Cada uno de tales nodos firma un mensaje y autentifica sus propios subnodos, etc.

Firma de documentos con criptografía de clave pública

Hay algoritmos de clave pública que pueden ser utilizados para firmas digitales. En algunos algoritmos (p. ej. RSA) tanto la clave pública como la privada pueden ser utilizadas para la encriptación. Al encriptar un documento con nuestra clave privada obtendremos una firma digital segura. En otros casos (como DSA) hay un algoritmo separado para firmas digitales que no pueden ser utilizado para la encriptación).

La idea fue inventada por Diffie y Hellman y otros la expandieron y refinaron. El protocolo básico es simple:

1. A encripta el documento con si clave privada, por ello firma el documento.
2. A envía el documento firmado a B
3. B desencripta el documento con la clave pública de A, por lo tanto verifica la firma.

Este protocolo es mucho mejor que el anterior. T no necesita firmar ni verificar firmas. Las partes no necesitan a T no siquiera para resolver disputas (si B no puede ejecutar el paso 3, sabe que la firma no es válida).

Además el protocolo satisface todas las características deseables:

1. La firma es auténtica; cuando B verifica el mensaje con la clave pública de A, sabe que A lo ha firmado.
2. La firma es infalsificable; sólo A conoce su clave privada.
3. La firma no es reutilizable; la firma es función del documento y no puede ser transferida a ningún otro.
4. El documento firmado es inalterable; si se realiza cualquier alteración, la firma no puede ser verificada con la clave pública de A.
5. La firma no puede ser repudiada. B no necesita la ayuda de A para verificar su firma.

Firma de documentos y estampillado temporal

A puede ser engañada por B en algunas circunstancias. B puede reutilizar el documento y la firma juntos. Esto no es ningún problema si A ha firmado un contrato, pero puede ser excitante si A ha firmado un cheque digital.

Digamos que A envía a B un cheque digital firmado por 100.000 ptas. B lo envía al banco, el cual verifica la firma y transfiere el dinero de una cuenta a otra.

B, que es una persona poco escrupulosa, salva una copia del cheque digital. A la semana siguiente, envía de nuevo el cheque al banco (puede ser a un banco distinto). El banco vuelve a verificar la firma y vuelve a transferir el dinero de una cuenta a otra. Si A no cuadra su libreta de cheques, B puede hacer lo mismo durante años.

Por eso, las firmas digitales incluyen frecuentemente "timestamps". La fecha y hora de la firma son agregados al mensaje y firmados junto con el resto del mensaje. El banco almacena el timestamp en una base de datos, Ahora, cuando B intente hacer efectivo el dinero de A por segunda vez, el banco contrastará el timestamp en su base de datos y, como lo encontrará almacenado, avisará inmediatamente a la policía. B pasará los próximos años leyendo protocolos criptográficos.

La firma de documentos con criptografía de clave pública y las funciones hash unidireccionales

En las implementaciones prácticas, los algoritmos de clave pública son demasiado ineficaces para firmar documentos largos. Para ahorrar tiempo, se implementan a menudo protocolos de firma digital mediante funciones hash unidireccionales. En lugar de firmar un documento, A firma el hash de dicho documento. En este protocolo, tanto la función hash unidireccional como el algoritmo de firma digital deben ser acordados previamente.

1. A calcula el valor hash de un documento.
2. A encripta dicho valor con su clave privada, por lo tanto firma el documento.
3. A envía el documento y el valor hash firmado a B.
4. B vuelve a calcular el valor hash del documento que A le ha enviado. Después, con el algoritmo de firma digital, desencripta (con la clave pública de A) el valor hash que le ha sido remitido. Si ambos valores coinciden la firma es válida.

La velocidad se incrementa drásticamente y, puesto que la probabilidad de que dos documentos distintos tengan el mismo valor hash de 160 bits es de $1/2^{160}$, cualquiera puede estar seguro de que el hash es la firma del documento.

Si se utilizase una función hash que no fuera unidireccional, podría ser una cuestión más fácil crear múltiples documentos que dieran un mismo valor hash, luego cualquiera que firmara un documento particular podría ser engañado firmando varios documentos.

El protocolo tiene otras ventajas. Primero, la firma puede ser mantenida por separado del documento. Segundo, los requerimientos de almacenaje por parte del destinatario para el documento y firma son mucho menores. Un sistema de archivos puede usar este tipo de protocolo para verificar la existencia de documentos sin almacenar sus contenidos. La base de datos central podría almacenar los valores hash de los ficheros. No necesita ver los ficheros. Los usuarios someten sus hashes a la base de datos, y la base de datos estampilla la fecha y hora de los envíos y los almacena. Si hay un desacuerdo en el futuro sobre quién creó un documento y cuándo, la base de datos puede resolverlo buscando el hash en sus ficheros. Este sistema tiene grandes implicaciones en lo que concierne

a la privacidad: A puede tener el copyright de un documento pero mantenerlo todavía en secreto. Sólo si desea demostrar su copyright tendría que hacer el documento público.

Algoritmos y terminología

Hay muchos algoritmos de firma digital. Todos son de clave pública con información secreta para firmar documentos e información pública para verificar firmas. A veces el proceso de firmado se llama **encriptado con clave privada** y el proceso de verificación se llama **desencriptado con clave pública**. Esta terminología es engañosa y sólo es verdadera para RSA. Diferentes algoritmos tienen implementaciones distintas.

Por ejemplo, las funciones hash unidireccionales y timestamps a veces agregan pasos extra para los procesos de firma y verificación. Hay muchos algoritmos que pueden usarse para firmas digitales, pero no para encriptación.

En general, nos referiremos a los procesos de firma y verificación sin detallar los algoritmos que envuelven. Firmar un mensaje con la clave privada K es:

$S_K(M)$

y verificar la firma con la clave pública correspondiente es:

$V_K(M)$

La cadena de bits adjuntada al documento cuando se firma (en el ejemplo previo el hash unidireccional del documento encriptado con la clave privada) se llamará **firma digital**, o simplemente, **firma**. El protocolo entero, mediante el cual el receptor de un mensaje es convencido de la identidad del emisor y de la integridad del mensaje, se llama autentificación.

Firmas múltiples

¿Cómo podrían A y B firmar el mismo documento? Sin funciones hash unidireccionales, hay dos opciones. Una es que A y B firmen copias separadas del documento. El mensaje resultante sería dos veces mayor

que el tamaño del documento original. La segunda es que A firme el documento en primer lugar y después B firme la firma de A. Esto funciona, pero es imposible verificar la firma de A sin verificar también la de B.

Con funciones hash unidireccionales, las firmas múltiples son fáciles:

1. A firma el hash del documento.
2. B firma el hash del documento.
3. B envía su firma a A.
4. A envía el documento, su firma y la firma de B a C.
5. C verifica tanto la firma de A como la de B.

No repudiación y firmas digitales

A puede engañar con firmas digitales sin que se pueda hacer nada. Puede firmar un documento y mas tarde, si no le conviene, negar que lo hizo.

Primero firma el documento normalmente. Después, anónimamente, publica su clave privada, convenientemente la pierde (o pretende que la ha perdido) y asegura que la firma ha sido comprometida y que los demás la han utilizado. Desautoriza la firma del documento y de cualesquiera otros que haya firmado con dicha clave privada. A esto se le llama **repudiación**.

Los timestamps pueden limitar los efectos de esta clase de engaño, pero A puede siempre afirmar que su clave fue comprometida con anterioridad. Por eso hay mucho que decir sobre las claves privadas enterradas en módulos resistentes al sabotaje.

Aunque no se puede hacer nada sobre este posible abuso, se pueden realizar algunos pasos para garantizar que las viejas firmas no están invalidadas por acciones tomadas en disputas posteriores. Por ejemplo, A podría "perder" su clave para no pagar a B el coche que vendió y, en el proceso, invalidar su cuenta en el banco. La solución para el receptor de un documento firmado es tenerlo estampillado.

El protocolo general es:

1. A firma el mensaje.

2. A genera una cabecera conteniendo alguna información identificativa. La concatena al mensaje firmado, lo vuelve a firmar y lo envía a T.
3. T verifica la firma externa y confirma la identificación. Añade un timestamp al mensaje firmado de A y la identificación. Después firma todo y lo envía a ambos (A y B).
4. B verifica la firma de T, la identificación y la firma de A.
5. A verifica el mensaje que T envía a B. Si no ha originado el mensaje deberá decirlo inmediatamente.

Aplicaciones de firmas digitales

Una de las aplicaciones mas antiguas propuestas para las firmas digitales fue para facilitar la verificación de las pruebas nucleares abolidas en tratados. Los EEUU y la Unión Soviética, permitieron poner sismómetros cada una en el suelo de la otra para monitorizar las pruebas nucleares. El problema fue que cada país necesitaba asegurarse de que la nación anfitriona no saboteaba los datos de los sismómetros, Simultáneamente, la nación anfitriona necesitaba asegurarse de que el monitor enviaba sólo la información específica necesaria para la monitorización.

Técnicas convencionales de autentificación pueden resolver el primer problema, pero sólo las firmas digitales pueden resolver ambos. La nación anfitriona puede leer, pero no alterar, los datos del sismómetro, y la nación que monitoriza sabe que los datos no han sido saboteados.

Firmas digitales con encriptación

Combinando firmas digitales con criptografía de clave pública, desarrollamos un protocolo que combina la seguridad de la encriptación con la autentificación de las firmas digitales. Piense en una carta de su madre. La firma proporciona un prueba de autoría y el sobre proporciona privacidad.

1. A firma el mensaje con su clave privada $S_A(M)$.
2. A encripta el mensaje firmado con la clave pública de B y se lo envía: $E_B(S_A(M))$.

3. B desencripta el mensaje con su clave privada: $D_B(E_B(S_A(M))) = S_A(M)$.
4. B lo verifica, con la clave pública de A, y recupera el mensaje: $V_A(S_A(M)) = M$.

Firmar antes de la encriptación parece natural. Cuando A escribe una carta, la firma y después la pone en un sobre. Si la mete en el sobre sin haberla firmado y firma el sobre, entonces B puede preocuparse si la carta no ha sido secretamente reemplazada. Si B muestra a C la carta de A, C puede acusar a B de mentir sobre la carta que ha llegado en el sobre.

En la correspondencia electrónica también, firmar antes de encriptar es una práctica prudente. No sólo es más seguro (un adversario no puede remover una firma de un mensaje encriptado y añadir una propia) sino que hay consideraciones legales: Si el texto que ha de ser firmado no es visible para el firmante cuando anexa su firma, entonces la firma puede tener una fuerza legal muy pequeña, Y hay algunos ataques criptoanalíticos contra esta técnica con firmas RSA.

No hay motivo para que A use el mismo par de claves (clave pública, clave privada) para encriptar y firmar. Puede tener dos pares de claves: una para la encriptación y otra para las firmas. La separación tiene sus ventajas: puede rendir su clave de encriptación a la policía sin comprometer su firma, una clave puede ser escrowed sin afectar a la otra, y las claves pueden tener tamaños distintos y expirar en tiempos diferentes.

Evidentemente, podría utilizarse el estampillado en este protocolo para prevenir la reutilización del mensaje. Los timestamps pueden ser también protección contra otros potenciales asechanzas, tales como las que describimos a continuación.

Reenvio del mensaje como un recibo

Consideremos una implementación de este protocolo, con la característica adicional de confirmar los mensajes. Si B recibe un mensaje, lo ha de devolver como una confirmación de su recepción.

1. A, firma un mensaje con su clave privada, lo encripta con la clave pública de B y lo envía a B: $E_B(S_A(M))$.

2. B desencripta el mensaje con su clave privada y verifica la firma con la clave pública de A, consecuentemente verifica que A ha firmado el mensaje y lo recupera: $V_A(D_B(E_B(S_A(M)))) = M$
3. B, firma el mensaje con su clave privada, lo encripta con la clave pública de A, y lo reenvía a A: $E_A(S_B(M))$
4. A, desencripta el mensaje con su clave privada y verifica la firma con la clave pública de B. Si el mensaje resultante es el mismo que el que ha enviado a B, sabe que B ha recibido el mensaje correctamente.

Si el mismo algoritmo es utilizado para la encriptación y para la verificación de la firma digital hay un ataque posible. Es estos casos, la operación de firma digital es la inversa de la operación de encriptación: $V_X = E_X$ y $S_X = D_X$.

Supongamos que M es un usuario legítimo del sistema con sus propias claves pública y privada. Ahora vigilemos cómo lee el correo de B. Primero registra el mensaje que A envía a B en el paso (1). Después, un poco más tarde, envía el mensaje a B afirmando que el mensaje procede de él (M). B piensa que es un mensaje legítimo de M, luego desencripta el mensaje con su clave privada y después intenta verificar la firma de M desencriptándola con la clave pública de M. El mensaje resultante, que es un puro galimatías, es

$$E_M(D_B(E_B(D_A(M)))) = E_M(D_A(M))$$

Aún así, B prosigue con el protocolo y envía a M un recibo:

$$E_M(D_B(E_M(D_A(M))))$$

Ahora, todo lo que M tiene que hacer es desencriptar el mensaje con su clave privada, encriptarlo con la clave pública de B, desencriptarlo con su clave privada, y encriptarlo con la clave pública de A. ¡Voilá! M tiene el mensaje.

No es razonable imaginar que B envíe automáticamente a M un recibo. Este protocolo puede estar incrustado en su software de comunicaciones, por ejemplo, y envía recibos automáticamente. Es la voluntad de admitir el recibo del galimatías lo que crea la inseguridad. Si B examina la comprensibilidad del mensaje antes de enviar un recibo, podría evitar este problema.

Existen mejoras de este ataque que permiten a M enviar a B un mensaje distinto del que ha escuchado. Nunca debemos firmar mensajes arbitrarios de otras personas o desencriptar mensajes arbitrarios y dar el resultado a otros.

Frustrar el ataque de reexpedición al destinatario

El ataque descrito funciona porque la operación de encriptación es la misma que la de verificación de firma y la desencriptación es la misma que la operación de firma. Un protocolo seguro podría usar una operación ligeramente distinta para la encriptación y para la firmas digitales. El uso de claves diferentes para cada operación resuelve el problema, así como emplear algoritmos distintos para cada operación; así como los estampillados que hacen el mensaje de entrada y el de salido diferentes; así como las firmas digitales con funciones hash unidireccionales.

Entonces, en general, el siguiente protocolo es seguro cuando se usa un algoritmo de clave pública:

1. A firma un mensaje.
2. A lo encripta y lo firma con la clave pública de B (utilizando un algoritmo de encriptación distinto que el de la firma) y lo envía a B.
3. B desencripta el mensaje con su clave privada.
4. B verifica la firma de A.

Ataques contra la criptografía de clave pública

El modo más fácil de obtener la clave pública de alguien es de alguna base de datos segura. La base de datos tiene que ser pública, para que cualquiera pueda obtener la clave pública de otro. La BD también tiene que estar protegida contra escritura de cualquiera excepto de T, de lo contrario M podría sustituir cualquier clave pública.

Incluso si las claves públicas están almacenadas en una base de datos segura, M todavía podría sustituir una por otra durante la transmisión. Para prevenirlo, T puede firmar cada clave pública con su clave privada. Cuando se usa de esta manera se dice que T es una "**Key Certification Authority**" o "**Key Distribution Center**" (KDC). En las implementacio-

nes prácticas, la KCD firma un mensaje compuesto que consta del nombre de los usuarios, su clave pública y cualquier otra información importante sobre el usuario. Este mensaje compuesto se almacena en la base de datos de la KDC. Cuando A obtiene la clave de B, verifica la firma de la KDC para asegurarse de la validez de la clave.

En el análisis final, esto no hace las cosas imposibles para M, sólo más difíciles. A todavía tiene la clave pública de la KDC almacenada en algún sitio. M podría sustituir su propia clave pública por ésta, corromper la base de datos, y sustituir sus propias claves por las válidas (todas firmadas con su propia clave privada como si él fuera la KDC). Pero incluso las firmas sobre papel pueden ser falsificadas si M está suficientemente apurado. El intercambio de claves será tratada en la Sección 1.3.

Timestamping

En algunas situaciones es necesario certificar que un documento existía en cierta fecha.

En el mundo digital resulta muy fácil cambiar la fecha de un documento, por lo tanto los protocolos que buscamos deben tener las siguientes propiedades:

- La fecha debe ser estampillada, sin que importe el medio físico en que resida.
- Debe ser imposible cambiar un solo bit del documento sin que el cambio se haga aparente.
- Debe ser imposible estampillar un documento con fecha y hora distintas a la presente.

Una solución

1. A envía una copia del documento a T
2. T registra la hora y fecha de recepción y realiza una copia del documento para su custodia.

Este protocolo funciona pero presenta algunos problemas evidentes.

Primero, no hay privacidad. A tiene que dar una copia del documento a T. Cualquier espía del canal de comunicaciones podría leerlo.

Segundo, la BD de T debe ser descomunal, y el ancho de banda requerido para enviar documentos largos puede ser inmanejable.

El tercer problema está relacionado con los errores potenciales. Un error en la transmisión o un virus en el ordenador de T, podrían invalidar completamente las argumentaciones de A sobre una fecha.

Veamos una solución mejorada:

1. A calcula un has del documento.
2. A lo transmite a T.
3. T agrega la fecha y hora al hash recibido, lo hashea de nuevo y firma digitalmente el resultado.
4. T devuelve el has firmado con el timestamping a A.

Secreto compartido

Los **esquemas de secreto compartido** fueron descubiertos independientemente por **Blakley** y **Shamir**. El motivo para que un secreto se comparta es la seguridad en la administración de las claves. En algunas situaciones hay una clave secreta que proporciona acceso a ficheros muy importantes. Si tal clave se pierde (por ejemplo, la persona que conoce la clave sufre un accidente, o el ordenador que la almacena es destruido), entonces los ficheros serán inaccesibles. La idea básica del secreto compartido es dividir la clave en piezas para diferentes personas y que ciertos subconjuntos de las mismas puedan recuperarla.

Un esquema compartido muy simple es el llamado **esquema umbral**. Consiste en tomar un mensaje cualquiera y dividirlo en n partes, llamadas **sombras** (shadows) o **participaciones** (shares), de manera que m cualesquiera de ellas puedan emplearse para reconstruir el mensaje. Con más precisión, esto se llaman un **esquema de umbral (m, n)**.

Los **esquemas de umbral generales**, aún son mas versátiles. Cualquier posibilidad de compartición imaginable, puede ser modelizado: Podemos dividir el mensaje entre nuestros vecinos de manera que puedan reconstruirlo si hay 7 personas del primer piso y 5 del segundo, a menos

que haya alguien del tercero, en cuyo caso sólo necesitamos que haya dicha persona y tres del primer piso; pero si hay uno del cuarto, entonces...

Adi Shamir usa ecuaciones polinómicas en un campo finito para construir un esquema umbral. Escojamos un número primo, p, que sea mayor que el número de participaciones posibles y mayor que el secreto más extenso posible. Generemos un polinomio arbitrario de grado m-1.

Por ejemplo, si queremos crear un esquema umbral (3, n) (es decir, necesitamos 3 participaciones para reconstruir el secreto), generaremos un polinomio cuadrático:

$$(ax^2 + bx + M) \quad \text{mód } p$$

siendo p un número primo aleatorio mayor que los coeficientes.

Los coeficientes a y b también se escogen aleatoriamente y son mantenidos en secreto hasta que se descartan después de que las participaciones se hayan creado.

Las participaciones se obtienen evaluando el polinomio en n puntos distintos:

$k_i = F(x_i)$

En otras palabras, la primera participación podría ser el valor del polinomio para x = 1, el segundo para x = 2, etc.

Puesto que el trinomio tiene tres coeficientes desconocidos, cualquiera de las participaciones puede servirnos para plantear tres ecuaciones. (Dos serían insuficientes y 4 o más, redundantes).

Por ejemplo, sea M = 11, para construir un esquema umbral (3,5) (en el que tres personas cualesquiera de las 5 puedan reconstruir M), primero generaremos un trinomio de segundo grado:

$$F(x) = (7x^2 + 8x + 11) \quad \text{mód } 13$$

(suponemos que el 7 y el 8 han sido elegidos aleatoriamente).

Las 5 participaciones serían:

$k_1 = F(1) = 0 \quad (\text{mód } 13)$
$k_2 = f(2) = 3 \quad (\text{mód } 13)$
$k_3 = f(3) = 7 \quad (\text{mód } 13)$
$k_4 = f(4) = 12 \quad (\text{mód } 13)$
$k_5 = f(5) = 5 \quad (\text{mód } 13)$

Para reconstruir M de 3 de tales participaciones (por ejemplo k_1, k_2 y k_5) basta resolver el siguiente sistema de ecuaciones lineales:

$$a \cdot 2^2 + b \cdot 2 + M \equiv 3 \pmod{13}$$
$$a \cdot 3^2 + b \cdot 2 + M \equiv 7 \pmod{13}$$
$$a \cdot 5^2 + b \cdot 2 + M \equiv 5 \pmod{13}$$

cuya solución es $a = 7$, $b = 8$ y $M = 11$, luego M se ha recuperado.

Pruebas "interactivas" y " de conocimiento cero"

Informalmente, una **prueba interactiva** es un protocolo entre dos partes mediante el cual una de ellas (llamada el **probador**) intenta demostrar un **hecho** a la otra (llamada **verificador**).

Una **prueba interactiva** toma, usualmente, la forma de *reto-respuesta*, en la cual el probador y el verificador intercambian mensajes, siendo el del verificador una aceptación o un rechazo al final del protocolo.

Aparte de su interés teórico, las **pruebas interactivas** pueden aplicarse a la criptografía y a la seguridad en temas tales como la identificación y la autentificación. En tales situaciones el hecho que hay que probar es, generalmente, la identidad del probador, así como su clave privada.

Es conveniente que las **pruebas interactivas** tengan las siguientes propiedades (especialmente en aplicaciones criptográficas):

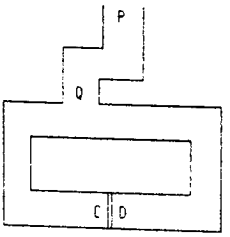
Completitud. El verificador acepta la prueba siempre que el hecho sea verdadero y tanto el probador como el verificador sigan el protocolo.

Robustez. El verificador rechaza siempre la prueba si el hecho es falso, mientras el verificador siga el protocolo.

Conocimiento cero. El verificador no sabe nada sobre el hecho que está siendo probado (excepto que sea o no correcto).

En una **prueba de conocimiento cero**, el verificador no puede probar posteriormente el hecho a nadie. (No todas las *pruebas interactivas* tienen esta propiedad).

Un *round* típico en una *prueba de conocimiento cero* consta de n mensaje de "compromiso" por parte del probador, seguida de un reto del verificador y de la respuesta del probador a dicho reto. El protocolo puede ser repetido muchas veces y, según las respuestas recibidas, el verificador decide si acepta o rechaza la prueba.



Consideremos un ejemplo intuitivo llamado "*La cueva de Alí Babá*". A quiere demostrar a B que sabe las palabras secretas para abrir la puerta que separa las dos zonas de la cueva C y D, pero no desea revelar el secreto a B. El "compromiso" de A es ir de C a D. B se sitúa en P y espera allí a que A vaya de C a D. Entonces B se desplaza a la posición Q y grita a A para que aparezca (ya sea por el lado derecho o por el izquierdo del túnel). Si A desconoce las palabras secretas (es decir "Ábrete Sésamo), sólo hay el 50% de probabilidad de que venga por el lado derecho del túnel. B repite el proceso tantas veces como necesite hasta que se convenza de que A sabe o no las palabras mágicas. Al final de la experiencia, B no aprende las palabras mágicas, pero decide si A las sabe o no.

Hay varios protocolos de *pruebas interactivas* y de *conocimiento cero* para aplicaciones criptográficas. Todas ellas se basan en la dificultad de la factorización.

La variante más común del protocolo **Fiat-Shamir** es el esquema **Feige-Fiat-Shamir**. **Guillon** y **Quisquater** mejoraron el protocolo Fiat-Shamir en términos de requerimiento de memoria e interacción (número de "rounds" (rondas) en el protocolo).

Los esquemas de identificación basados en pruebas interactivas pueden ser transformados en esquemas de firma digital.

Dinero Digital

El dinero es un problema. Es molesto llevarlo, disemina gérmenes, puede robarse. Los cheques y las tarjetas de crédito tienen un seguimiento que anulan la privacidad. La policía puede vigilar nuestras transacciones financieras, dónde y qué compramos, a quién llamamos por teléfono, etc. Necesitamos un modo de conseguir el anonimato para proteger nuestra privacidad.

Felizmente hay un complicado protocolo válido para la autentificación sin que los mensajes puedan rastrearse.

El magnate A puede transferir **dinero digital** al congresista B sin que el periodista E sepa la identidad de A. B puede entonces depositar este dinero en su cuenta bancaria, incluso aunque el banco no sepa quién es A. Pero si A intenta comprar cocaína con el dinero digital que ha utilizado para sobornar a B, será detectado por el banco. Y si B intenta depositar el dinero digital en dos cuentas distintas, también será detectado (aunque A quedará en el anonimato). Esto se llama **dinero digital anónimo** para diferenciarlo del **dinero digital** que es el que se puede rastrear.

Existe una gran demanda social para este tipo de cosas, sobre todo con el creciente uso de Internet para transacciones comerciales. (Hay muy buenas razones para que la gente sea renuente a enviar sus números de tarjeta de crédito por Internet). Por otra parte, los bancos y gobiernos no están dispuestos a prescindir del control que el seguimiento de los sistemas actuales que la banca proporciona.

Los protocolos de dinero digital son muy complejos. Veremos uno paso a paso:

Protocolo 1

Los primeros protocolos son análogos físicos de protocolos criptográficos. Este primero lo es de las órdenes de dinero anónimo:

1. A prepara 100 libranzas [14] anónimas de 1000 dólares cada una.

[14] Orden que da una persona para que, de los fondos que tiene a su disposición, le sea entregada cierta cantidad a otra. (María Moliner).

2. Alicia pone a cada uno, junto con un trozo de papel carbón, en 100 sobres diferentes y los da al banco.
3. El banco abre 99 sobres y confirma que en cada uno hay una libranza por valor de 1000 dólares.
4. El banco firma el único sobre que no ha sido abierto. La firma se copia a través del papel carbón en la libranza. El banco devuelve el sobre que no ha abierto a Alicia, y deduce 1000 dólares de su cuenta.
5. Alicia abre el sobre y gasta su libranza en un comercio.
6. El dueño del establecimiento comprueba la firma del banco para asegurarse que la libranza es legítima.
7. El comerciante acepta la libranza del banco.
8. El banco verifica su firma y abona 1000 dólares en la cuenta del comerciante.

Este protocolo funciona. El banco nunca ve la libranza que ha firmado, luego cuando el comerciante la trae, no sabe que es de Alicia. El banco está convencido de su validez, debido a su firma. El banco confía en que la libranza no abierta es de 1000 dólares (y no de 100.000 o 1000.000.000) debido al protocolo "cut and choose" (v. Secc. 5.1). Verifica los otros 99 sobres, luego Alicia sólo tiene un 1 % de probabilidad de hacer trampa. Evidentemente, el banco hará una penalización suficientemente grande para que no valga la pena correr el riesgo.

Protocolo 2

El protocolo anterior no evita que Alicia fotocopie la libranza y la gaste dos veces. A esto se le llama el **problema del doble gasto**; para resolverlo, necesitamos complicarlo un poco:

1. Alicia prepara 100 libranzas anónimas de 1000 dólares cada una. En cada una, incluye una cadena única generada aleatoriamente, suficientemente larga para que la probabilidad de que otra persona también la use sea insignificante.

2. Alicia pone cada una (junto con un trozo de papel carbón) en 100 sobres diferentes y los entrega en el banco.
3. El banco abre 99 sobres y confirma que en cada uno hay una libranza por valor de 1000 dólares.
4. El banco firma el único sobre que no ha sido abierto. La firma se copia a través del papel carbón en la libranza. El banco devuelve el sobre que no ha abierto a Alicia, y deduce 1000 dólares de su cuenta.
5. Alicia abre el sobre y gasta su libranza en un comercio.
6. El dueño del establecimiento comprueba la firma del banco para asegurarse que la libranza es legítima.
7. El comerciante acepta la libranza del banco.
8. El banco verifica su firma y consulta su base de datos para asegurarse de que existe una libranza con la misma cadena que no haya sido previamente depositada. Si no la hay, el banco abona 1000 dólares en la cuenta del comerciante. El banco registra la cadena en su base de datos.
9. Si ya ha sido previamente depositada, el banco no acepta la libranza.

Ahora, si Alicia intenta emplear una fotocopia de la libranza, o si lo intenta el comerciante, el banco lo sabrá inmediatamente.

Protocolo 3

El protocolo anterior protege al banco de estafadores, pero no los identifica. El siguiente protocolo lo corrige:

1. A prepara 100 libranzas anónimas de 1000 dólares. En cada una, incluye una cadena aleatoria distinta suficientemente larga para hacer prácticamente imposible que otra persona también la utilice.
2. A pone cada libranza en un sobre junto con un papel de calco, y los entrega en el banco.
3. El banco abre 99 sobres al azar y confirma que en cada uno hay una libranza de 1000 dólares y que todas las cadenas aleatorias son diferentes.

4. El banco firma la única que le queda por abrir. Gracias al papel carbón, la firma se copia en la libranza. El banco devuelve el sobre sin abrir a A y le descuenta 1000 dólares de su cuenta.
5. A abre el sobre y gasta la libranza en un comercio.
6. El dueño del establecimiento comprueba la firma del banco para asegurarse de que la libranza es legítima.
7. El comerciante pide a A que escriba una cadena aleatoria identificativa en la libranza.
8. A lo acepta.
9. El comerciante recibe la libranza del banco.
10. El banco verifica la firma y examina su base de datos para verificar que ninguna libranza con la misma cadena haya sido depositada. Si no la encuentra abona los 1000 dólares en la cuenta del comerciante. El banco registra la cadena de la libranza y la cadena identificativa en su base de datos.
11. Si la cadena de la libranza ya está en su base de datos, el banco no acepta la libranza. Después, compara la cadena identificativa sobre la libranza con la que tiene almacenada en su base de datos. Si es la misma, el banco sabe que el comerciante ha fotocopiado la libranza. Si es diferente, el banco sabe que ha sido A quien la ha fotocopiado.

Otros protocolos

Hay otros protocolos, algunos de los cuales necesitan matemáticas algo complicadas. Generalmente, los diversos protocolos sobre dinero digital pueden dividirse varias categorías:

Los sistemas **On line**, los cuales requieren que el comerciante se comunique con el banco en cada venta, muy parecidos a los protocolos de tarjetas de crédito. Si hay un problema, el banco no acepta el efectivo.

Los sistemas **off line** no requieren ningún tipo de comunicación entre el banco y el comerciante. No previene que A pueda hacer trampas, pero el banco las detectará. La decisión es responsabilidad suya.

Otro sistema consiste en crear tarjetas inteligentes especiales que contenga un chip llamado **observador**, el cual conserva una mini base de datos de todas las cantidades gastadas. Si su poseedor intenta copiarlas y gastar el dinero dos veces, el procesador lo detecta e impide la transac-

ción. El "observador" es resistente al sabotaje, luego A no puede editar la base de datos sin dañarlo permanentemente.

Los protocolos sobre dinero digital también pueden ser considerados desde otro punto de vista. Las **monedas electrónicas** tienen un valor fijo; las personas que las utilizan necesitan varias con diferentes denominaciones. Los **cheque electrónicos** pueden utilizarse con cualquier cantidad por debajo de un cierto límite. Existe un sistema llamado **NetCash** pero tiene una anomicidad muy débil.

Tatsuaki Okamoto y Kazuo Ohta han enunciado seis propiedades que debe tener todo sistema idóneo sobre dinero digital:

1. **Independencia**. Su seguridad no ha de depender del emplazamiento físico, debe poder transferirse a través de cualquier red de ordenadores.
2. **Seguridad**. No podrá ser copiado y reutilizado.
3. **Privacidad**. Nadie puede rastrear la relación entre el usuario y sus adquisiciones.
4. **Pago off line**. Esto es, cuando un usuario efectúe un pago, el protocolo entre él y el comerciante se ejecutará de modo que la tienda no necesite conectarse a un host para realizar el proceso.
5. **Transferibilidad**. El dinero digital debe poder ser transferido con facilidad a otros usuarios.
6. **Divisibilidad**.

Tarjetas de crédito anónimas

Este protocolo utiliza varios bancos distintos para proteger la identidad del cliente. Cada comprador posee una cuenta en dos bancos diferentes. El primero conoce su identidad y puede concederle un crédito. El segundo sólo conoce al comprador a través de un seudónimo.

El cliente puede retirar fondos del segundo banco probando que la cuenta es suya. Sin embargo como el banco no sabe su identidad, no está dispuesto a concedérselos directamente. El primer banco conoce al cliente y transfiere fondos al segundo (sin conocer el seudónimo). El cliente

puede gastar este dinero anónimamente. Al final de cada mes, el segundo banco proporciona al primero una nota de crédito, con la confianza de que éste se la pague. El primer banco presenta el recibo al cliente para que lo abone. Cuando lo hace, el primer banco lo transfiere al segundo.

Las transacciones se realizan a través de un intermediario el cual actúa como una Reserva Federal electrónica: estableciendo cuentas entre bancos, administrando los mensajes, etc.

A menos que todos se pongan en contra del cliente, su anonimato está asegurado. Sin embargo esto no es dinero digital. El protocolo permite a los clientes mantener las ventajas de la tarjeta de crédito sin perder su privacidad.

BIBLIOGRAFÍA

Pino Caballero, 1996. *Introducción a la Criptografía.* Editorial Rama.

Lars Klander, 1998.*A prueba de Hackers.* Editorial Anaya.

Antonio Zamora Gómez, 1996. *Introducción a la Teoría de la Información y de la Codificación.* Ed. Club Universitario.

José Luís Morant, Arturo Ribagorda, Justo Sancho, 1997 *Seguridad y Protección de la Información.* Ed. Centro de Estudios Ramón Areces S. A.

Javier Cilleruelo, Antonio Córdoba, 1992 *La Teoría de los Números.* Ed. Mondadori España S.A.

Bruce Schneier, 1996. *Applied Criptography (Second Edition).* John Willey & Sons.

Laboratorios RSA. *Preguntas más frecuentes de la criptografía actual.*

http://www.ssh.fi/tech/crypto/intro.htm. Introducción a la Criptografía.

http://www.kriptopolis.com. Diversas secciones y artículos.

http://www.cis.ohio-state.edu. Preguntas mas frecuentes sobre criptografía.

http://www.rsa.com. Todo lo referente a RSA.

http://bs.mit.edu:8001/ipra.html. Internet Regional Policy Authority.

http://www.quadralay.com/www/Crypt/DES/source-books.html. Código fuente de DES en Fortran, C, Ensamblador y Basic.

http://www.cygnus.com/gnu/export.htm. Casos judiciales referentes a la criptografía y al control de la exportación.

ftp://ftp.csua.berkeley.edu/pub/cypherpunks/steganography. Archivos de esteganografía.

http://www.visa.com y http://www.mastercard.com. Especificaciones sobre tarjetas inteligentes Visa y Mastrercard.

ftp://cpsr.org/cpsr/privacy/crypto/clipper. Archivos sobre Clipper.

http://www.europarl.eu.int/dg4/stoa/en/publi/166499/execsum.html. Tecnologías de control político.

http://www.freecongress/ctp/echelon.html. Informe: "Echelon: El espía de América en el cielo".

http://www.iti.upv.es. Instituto Tecnológico de Informática de la Universidad Politécnica de Valencia.

http://www-ma2.upc.es. Facultad de Informática de Barcelona.

http://www.rediris.es. Servidor de claves, referencias y enlaces.