Curso de Criptografía Aplicada

Avance del proyecto Libro Electrónico Multimedia de Criptografía Aplicada LEMCA
Vídeoclases en YouTube disponibles desde mediados de enero de 2020 en Class4crypt

https://www.youtube.com/user/jorgeramio

Añadido por el autor el 2 de enero de 2020

El proyecto inicial LECA, Libro Electrónico de Criptografía Aplicada, ha derivado finalmente en el proyecto LEMCA, Libro Electrónico Multimedia de Criptografía Aplicada, en el que voy a grabar una centena de vídeoclases, con una temática mucho más amplia que la de este libro, con temas nuevos, otros actualizados y añadiendo prácticas desarrolladas en las mismas. Estas vídeoclases estarán disponibles desde mediados del mes de enero de 2020 en mi canal personal de YouTube.

Canal YouTube de Class4crypt

https://www.youtube.com/user/jorgeramio

- Recomiendo que pulses en **Suscribirse**.

Para consultas, seguimiento de clases y noticias, usaremos Twitter:

Canal Twitter de Class4crypt

https://twitter.com/class4crypt

- Recomiendo que pulses en **Seguir**.

Madrid, 2 de enero de 2020

Dr. Jorge Ramió Aguirre

Añadido por el autor en mayo de 2019

Puedes utilizar esta documentación, otros libros, material multimedia y software de prácticas generados en Criptored, todos de libre distribución en Internet, para poder demostrar que entiendes y sabes cómo trabaja la criptografía, logrando la nueva certificación técnica profesional ***CriptoCert Certified Crypto Analyst***, reconocida por el Centro Criptológico Nacional CCN de España, disponible desde el mes de abril de 2019.

Encontrarás más información sobre esta certificación y correo de contacto en el sitio web: https://www.criptocert.com

Madrid, 6 de mayo de 2019

Dr. Jorge Ramió Aguirre

Prólogo

La criptografía es una de las áreas de la seguridad informática (y seguridad de la información) que más investigación está generando en los últimos años de esta segunda década del siglo XXI. Estando ya a finales del año 2018, no hace falta recordar la importancia que están adquiriendo en la sociedad temas como blockchain y las criptomonedas, el cifrado homomórfico para la seguridad en el tratamiento de la información cifrada en la nube, nuevos enfoques a protocolos de conocimiento cero y la criptografía post-cuántica, por nombrar solo algunos de los frentes de investigación y desarrollo abiertos en esta temática.

Sin embargo, para entender el porqué del auge de estas nuevas tecnologías, es menester conocer antes los conceptos básicos asociados a la criptografía actual, sus algoritmos, sus fortalezas y sus debilidades. En este curso, reflejado en el temario de este libro, se hace un repaso genérico de la criptografía, analizando el funcionamiento de los algoritmos más importantes.

Teniendo en mente un nuevo y ambicioso proyecto, consistente en la publicación en Internet de un extenso libro electrónico de criptografía, cuyo título será Criptografía Aplicada y que se entregará mensualmente por capítulos a partir de mediados del año 2019, publico hoy en Criptored este resumen de 134 páginas y que el autor ha estado utilizando en estos últimos años en sus clases de criptografía aplicada como profesor invitado, entre otros, en los siguientes posgrados:

- Máster en Seguridad Informática online, Universidad Internacional de La Rioja (Madrid, España)
- Especialización en Criptografía y Seguridad Teleinformática, Escuela Superior Técnica EST (Buenos Aires, Argentina)
- Doctorado en Informática, Escuela Politécnica Nacional (Quito, Ecuador)
- Máster en Auditoría, Seguridad, Gobierno y Derecho de las TIC, Universidad Autónoma de Madrid (Madrid, España)
- Diploma en Seguridad Informática, Facultad de Ciencias Económicas y Administrativas de la Universidad de Chile (Santiago, Chile)
- Maestría en Ciberseguridad, Panamerican Business School (Ciudad de Guatemala, Guatemala)

Al ser una versión beta de dicho libro, el lector notará que faltan varios ejemplos, referencias a pie de página, sitios de interés, prácticas de laboratorio con software de dominio público, incrustación donde corresponda de material multimedia propiedad del mismo autor, un mayor número de capítulos, un índice más detallado, etc. Todo esto se incluirá en el Libro Electrónico de Criptografía Aplicada que se publicará en 2019.

A pesar de lo anterior, este libro permite seguir o bien impartir un curso de introducción a la criptografía, desde lo más básico hasta el sistema de criptografía asimétrica de Elgamal, y cuya carga docente se encuentre

entre 12 y 40 horas, dependiendo ello de la profundización que se haga en cada temática y del tiempo que se le pueda asignar a las prácticas, un apartado vital en criptografía para un correcto aprendizaje.

Si eres profesor/a de asignaturas de seguridad en las que se imparta criptografía, puedes utilizar este material docente en tus clases o bien entregarlo a tus alumnos como bibliografía, puesto que se ha publicado bajo licencia Creative Commons CC BY-NC-ND (Reconocimiento – No Comercial – Sin Obra Derivada) que "permite que otros puedan descargar las obras y compartirlas con otras personas, siempre que se reconozca su autoría, pero no se pueden cambiar de ninguna manera ni se pueden utilizar comercialmente".

Se recomienda complementar su lectura con otras herramientas didácticas del mismo autor:

- Píldoras Formativas Thoth
  http://www.criptored.upm.es/thoth/index.php
- MOOC Crypt4you
  http://www.criptored.upm.es/crypt4you/portada.html
- Cuadernos de Laboratorio de Criptografía CLCript
  http://www.criptored.upm.es/software/sw_m001s.htm
- Enciclopedia de la Seguridad de la Información intypedia
  http://www.criptored.upm.es/intypedia/index.php?lang=es

Espero tengas una feliz y provechosa lectura.

En Madrid, a 8 de noviembre de 2018
Dr. Jorge Ramió Aguirre (Criptored)

Tema 1

Fundamentos de seguridad y criptografía

1.1. Introducción

1.2. Conceptos básicos de seguridad y criptografía

1.3. Matemáticas discretas

1.4. Uso de problemas matemáticos en la criptografía

1.5. Seguridad de los algoritmos criptográficos

1.6. Nociones de teoría de la información

1.7. Codificación de la información

1.1. Introducción

Para profundizar en este tema, lee los capítulos 3, 6 y 7 del siguiente libro.

Ramió, J. (2006). Libro electrónico de seguridad informática y criptografía, versión 4.1.

Disponible en: http://www.criptored.upm.es/guiateoria/gt_m001a.htm

También puedes lee las lecciones 1, 2, 3 y 5 del siguiente libro:

Ramió, J. (2016). Introducción a la seguridad informática y la criptología clásica.

Disponible en:

http://www.criptored.upm.es/crypt4you/temas/criptografiaclasica/leccion1.html

Se recomienda la visualización de las siguientes píldoras formativas Thoth del mismo autor:

Píldora 1: ¿Seguridad Informática o Seguridad de la Información?

Píldora 2: ¿Qué es la criptografía?

Píldora 5: ¿Qué es la tríada CIA?

Píldora 6: ¿Ciframos, codificamos o encriptamos?

Píldora 11: ¿Cifrando dentro de un cuerpo?

Píldora 12: ¿Qué son los inversos multiplicativos en un cuerpo?

Píldora 23: ¿Qué son los inversos aditivos en un cuerpo?

Píldora 24: ¿Por qué usamos el algoritmo de Euclides para calcular inversos?

Píldora 25: ¿Cómo calculamos inversos con el algoritmo extendido de Euclides?

Píldora 36: ¿Qué es el código Base64?

Píldora 37: ¿Cómo funciona el algoritmo de exponenciación rápida?

Disponibles en:

https://www.youtube.com/watch?v=7MqTpfEreJ0&list=PL8bSwVy8_IcNNS5QDLjV7gUg8dIeMFSER

En este primer capítulo introduciremos los principios matemáticos y de teoría de la información que se necesitan para comprender cómo funcionan y en qué basan su seguridad los algoritmos criptográficos. Estos principios están directamente relacionados con la matemática discreta o aritmética modular, la teoría de la información y la complejidad algorítmica**.**

De estos tres pilares, analizaremos, sin profundizar, los dos primeros. Daremos una mayor importancia a las matemáticas discretas ya que las usaremos para la resolución de ejercicios y prácticas durante el curso.

Durante el curso podrás comprobar que los conceptos y herramientas matemáticas que se necesitan para trabajar con la inmensa mayoría de los algoritmos de cifra, tanto clásica como moderna, son muy básicos.

En este capítulo conoceremos, además, los conceptos de la seguridad informática y la criptología, conoceremos y aplicaremos las herramientas de matemáticas discretas para las operaciones de cifra y analizaremos y comprobaremos en qué basan su fortaleza los algoritmos de cifra.

1.2. Conceptos básicos de seguridad y criptografía

El interés del ser humano por ocultar a terceros información sensible ha existido desde que se inventan los primeros métodos de escritura. Nace así la criptografía, teniendo sus inicios en el siglo V a. C con el artilugio conocido como escítala, usado por un pueblo griego para mantener sus secretos a buen recaudo.

Un sistema que permite en un extremo cifrar información y en el otro descifrarla, se conoce como criptosistema y su esquema es el mostrado en la Figura 1.1.

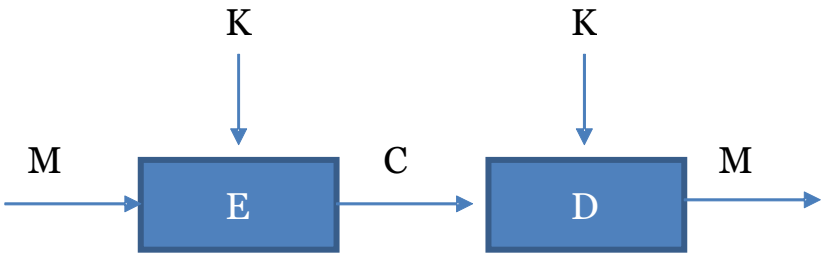


Figura 1.1. Esquema de un criptosistema clásico.

Los elementos que forman parte de un criptosistema son:

- **M**: que representa el mensaje o texto en claro, que por sí solo es legible o interpretable (caso de un programa o aplicación) por cualquier entidad.
- **C**: que representa al criptograma, esto es el texto cifrado que se trasmitirá por el canal de comunicación, el cual por definición es inseguro y por ello es necesario cifrar la información.
- **E**: que representa la función de cifrado que se aplica al texto en claro. La letra E proviene del inglés *Encrypt*.
- **D**: que representa la función de descifrado que se aplica al texto cifrado para recuperar el texto en claro. La letra D proviene del inglés *Decrypt*.
- **K**: que representa la clave empleada para cifrar el texto en claro M, o bien para descifrar el criptograma C. En los sistemas modernos de cifra asimétrica o de clave pública, estas claves K serán diferentes en ambos extremos.

Hay que destacar que, a pesar de los siglos, los sistemas de cifra clásicos y los modernos o actuales se diferencian muy poco entre sí. En el fondo, hacen las mismas operaciones, si bien los primeros estaban orientados a letras o caracteres, y los segundos lógicamente a bits y bytes. Pero los principios en los que se basan para conseguir su objetivo de enmascarar la información siguen siendo los mismos, la difusión y la confusión.

- La **difusión** pretende difundir las características del texto en claro en todo el criptograma, ocultando así la relación entre el texto en claro y el texto cifrado, y se logra mediante técnicas de cifra por permutación o transposición.

- Por su parte, la **confusión** pretende confundir al atacante, de manera que no le sea sencillo establecer una relación entre el criptograma y la clave de cifrado, y se logra mediante técnicas de cifra por sustitución.

Del esquema de la Figura 1.1, se deduce que:

$$E_k(M) = C$$

$$D_k(C) = D_k(E_k(M)) = M$$

Por lo tanto, la operación de cifrado E_k y la operación de descifrado D_k serán inversas, en el sentido de que una anula la operación de la otra y, por consiguiente, se recupera el texto en claro. Este concepto de inverso (algo más completo) será muy importante en criptografía.

En próximos capítulos veremos que en el descifrado puede usarse la operación inversa realizada en el cifrado utilizando la misma clave K, o bien utilizar en el descifrado la misma operación hecha en el cifrado, pero usando en este caso claves inversas dentro de un cuerpo finito.

Cualquier otra operación de descifrado que no sea exactamente igual a $D_k(C) = M$, no será válida. Si se diera algún caso de poder recuperar el texto en claro con una operación diferente a $D_k(C) = M$, hablaremos de claves débiles, como sucede con el algoritmo DES que veremos en el capítulo 3 de Algoritmos de cifra simétrica.

En criptografía asimétrica, que veremos en el capítulo 5, puede darse otro caso similar, si bien aquí este fenómeno se conocerá como número no cifrable, al no producir la cifra cambio alguno en el texto en claro.

Estas anomalías en los sistemas de cifra pueden resumirse en las siguientes tres ecuaciones:

$$E_k(E_k(M)) = M$$

$$E_{k1}(E_{k2}(M)) = M$$

$$E_k(M) = M$$

En la ecuación $E_k(E_k(M)) = M$, el cifrado doble con la misma clave K sobre el mensaje M permitiría recuperar el texto en claro. En la ecuación $E_{k1}(E_{k2}(M)) = M$, el cifrado doble con dos claves distintas sobre el mensaje M también permitiría recuperar el texto en claro. Ambos casos no están permitidos, se conocen como soluciones falsas y podremos comprobarlas por ejemplo en el DES. La ecuación $E_k(M) = M$, típica en cifrados asimétricos como por ejemplo RSA, indica que, aunque el mensaje M se cifre con la clave K, éste se transmite en claro.

Clasificación de los criptosistemas

La primera clasificación de los sistemas de cifra hace diferencia entre la criptografía clásica y la criptografía moderna, entendiéndose por moderna la cifra que está unida al mundo de la computación y al desarrollo de los computadores (mediados del siglo XX) y, por lo tanto, en donde dicha cifra se realiza sobre bits y bytes y no sobre letras o caracteres como se hacía en la cifra clásica o antigua.

Por poner una separación temporal entre ambas, esta podría delimitarse hacia la mitad del siglo XX y décadas posteriores, fechas en las que ocurren tres hechos muy significativos y que cambian por completo el concepto de la criptografía como se venía entendiendo durante siglos:

1. La publicación en 1948 y 1949 por parte de Claude Shannon de sendos documentos donde se sientan las bases de la teoría de la información y del secreto de los sistemas.

2. La publicación como estándar mundial de cifra simétrica por parte del NIST National Institute of Standards and Technology en 1974 del algoritmo DES, *Data Encryption Standard*, convirtiéndose en el primer algoritmo de esas características que se usaría en aplicaciones civiles, ya no militares, de gobierno o la iglesia, como era lo habitual.

3. El nacimiento en 1976 de la criptografía de clave pública tras el trabajo realizado por los investigadores de la Universidad de Stanford Whitfield Diffie y Martin Hellman, en el que definen un protocolo para el intercambio de clave en un medio inseguro, usando para ello problemas matemáticos de una sola dirección.

La segunda clasificación, en este caso solo aplicable a los sistemas de cifra modernos, diferencia entre criptografía simétrica, o de clave secreta, y criptografía asimétrica, o de clave pública.

- **Criptografía simétrica o de clave secreta**. La clave usada para cifrar y descifrar es idéntica y deberá ser compartida entre el emisor y el receptor. Debido a esta circunstancia, el empleo de este tipo de criptosistemas precisa que emisor y receptor dispongan de un canal seguro para el intercambio de la clave, algo que no ocurre hasta la invención en 1976 de la criptografía de clave pública.

- **Criptografía asimétrica o de clave pública**. En este caso cada entidad (usuario, máquina, etc.) dispone de una clave pública y de una clave privada, inversas entre sí, que pueden ser usadas para cifrar un mensaje la primera o descifrar un criptograma la segunda. Cuando la clave pública es usada para cifrar, el descifrado del criptograma resultante debe ser realizado con la clave privada (y viceversa). El sistema de cifra será seguro si es computacionalmente difícil (en cómputo y en tiempo) averiguar la clave privada conociendo solamente la clave pública.

Es posible hacer una nueva clasificación en los sistemas de cifra moderna de acuerdo a cómo se trata a la información en su proceso de cifrado, diferenciando ahora entre sistemas de cifra en flujo y sistemas de cifra en bloque.

La cifra en flujo cifrará de uno en uno bits o bytes de un mensaje en claro con una secuencia de bits o bytes que actúa como clave. Esta clave va cambiando en cada bit o byte y por ello se conoce como secuencia de clave S_i. Por su parte, en la cifra en bloque el texto en claro se agrupa en bloques de un conjunto de bytes, típicamente 16 bytes (128 bits) o bien 8 bytes (64 bits), que se cifran con la misma clave K.

En la práctica, se utilizará la criptografía asimétrica (computacionalmente más exigente que la criptografía simétrica) para transmitir la clave secreta de sesión (de un algoritmo simétrico) al destinatario empleando la clave pública de éste, clave secreta de sesión que posteriormente será usada por ese algoritmo simétrico de cifra (mucho más rápidos que los asimétricos). De la misma manera, se puede usar la criptografía asimétrica para firmar digitalmente un documento, empleándose en este caso la clave privada del emisor para la firma, y la clave pública del emisor para comprobar dicha firma. Este uso de ambas técnicas con fines distintos se conoce como criptografía híbrida.

Otra técnica relacionada con la criptografía es la esteganografía, si bien en este caso no se persigue la transformación del texto en claro en un criptograma, sino su ocultación dentro de otro mensaje (o portadora)

de forma que el texto original pase desapercibido. Es común utilizar como portadora archivos de imagen, audio o vídeo, debido a la alta redundancia que muestran.

Si a estas técnicas de cifrar información (criptografía) o bien ocultarla dentro de otro archivo inocuo (esteganografía), unimos la técnica que permite romper (en ciertos casos y bajo ciertas condiciones) el sistema de la cifra y la clave de una forma elegante sin usar la fuerza bruta, conocida como criptoanálisis, obtenemos las ramas de la criptología. Nos centraremos en este libro solamente en la primera y en última de las técnicas, la criptografía y el criptoanálisis.

1.3. Matemáticas discretas

La matemática discreta, o aritmética modular, es la herramienta principal con la que ha contado y sigue contando la criptografía para el desarrollo de algoritmos de cifra. Los conceptos más importantes a estudiar en este apartado serán las operaciones dentro de un módulo o cuerpo de cifra, el cálculo de inversos multiplicativos, de inversos aditivos y de inversos módulo 2 u or exclusivo. Por último, introduciremos el concepto de las raíces primitivas dentro de un número primo.

Aritmética modular

Dados tres números $a, b, n \in \mathbb{N}$, se dice que *a* es congruente con *b* módulo *n* si se cumple:

$$a = b + kn \text{ Para algún } k \in \mathbb{Z}$$

Lo que queda expresado mediante la siguiente ecuación:

$$a \equiv b(mod\ n)$$

Si bien la ecuación anterior muestra la forma matemática correcta de expresar la congruencia, en criptografía es común expresar dicha congruencia mediante una igualdad tal y como se muestra en la ecuación siguiente. Usaremos esta otra notación en toda la asignatura.

$$a \bmod n = b$$

Por ejemplo:

131 mod 16 = 3	Porque 131 = 16 * 8 + 3
611 mod 47 = 0	Porque 611 = 47 * 13 = 611 + 0

-53 mod 200 = 147 Porque -53 + 200 = 147

Dentro del cuerpo de los enteros, es normal trabajar en lo que se domina la zona canónica de los restos o residuos de un número n, que va desde 0 hasta n-1, si bien existe lo que se denomina clase de equivalencia que son todos los números que dentro de ese módulo son equivalentes. Por ejemplo, el resto 5 en el módulo 13 tiene, entre muchos otros, estos restos equivalentes: -21, -8, 18, 31. Esto es así porque en todos los casos al reducir módulo 13 obtenemos el resto 5:

-21 + 13*2 = 5; -8 + 13*1 = 5; 18 = 13*1 + 5; 31 = 13*2 + 5

Puesto que en criptografía se usarán de manera frecuente las operaciones de suma, multiplicación y or exclusivo entre el número a cifrar (en ciertos casos el código del mensaje a cifrar) y la clave, es primordial conocer el concepto de inverso, puesto que nos permitirá descifrar el criptograma para recuperar el secreto (confidencialidad) o bien descifrar la firma digital para comprobar la misma (integridad).

Inversos en un cuerpo

Los inversos aditivos y or exclusivo (por ejemplo, usados en el algoritmo IDEA) son muy sencillos. El inverso aditivo será el complemento al cuerpo y el inverso del or exclusivo o XOR será simplemente el mismo valor. En los siguientes ejemplos se comprueban el inverso aditivo de 12 en módulo 27 y el inverso XOR de 251 en módulo 779.

inv(+) [12, 27] = 15 Porque 12 + 15 = 27 + 0 (identidad de la suma)

Es decir, si se cifra un número secreto N (por ejemplo, el 20) mediante un desplazamiento de 12 espacios en el cuerpo o módulo 27 (resultado: 20 + 12 mod 27 = 32 mod 27 = 5), se puede recuperar ese secreto N aplicando un desplazamiento de -12 espacios (sentido contrario por el signo) al criptograma 5 esto es (5 -12 mod 27 = -7 mod 27 = 20) o bien, lo que es igual, un desplazamiento de -12 + 27 = 15 espacios, ahora en el mismo sentido, con lo que obtenemos (5 + 15 mod 27 = 20).

Observa que el inverso aditivo siempre existirá, porque el complemento al cuerpo no tiene ninguna restricción matemática.

inv⊕ [251, 779] = 251 Porque XOR es una función involutiva.

Es decir, si ciframos el número N = 583 (1001000111) con la clave K = 251 (11111011) mediante una función XOR (bits iguales da 0, bits distintos da 1), se obtiene:

N = 583	1001000111
K = 251	0011111011 (se añaden a la izquierda los ceros que hagan falta)
⊕	1010111100 (C = 700)

Recuperaremos N = 583 a partir de C = 700, usando la misma clave K = 251.

C = 700	1010111100
K = 251	0011111011 (añadimos nuevamente ceros a la izquierda)
⊕	1001000111 (N = 583)

Al igual que sucede con el inverso aditivo, el inverso XOR siempre existirá.

Los inversos más importantes en la criptografía moderna son los inversos mutiplicativos. Serán precisamente los que permitan, entre otras cosas, que en criptografía asimétrica hablemos de una clave pública y de una clave privada.

Si en una cifra multiplicamos por una constante (clave) el número secreto a cifrar, en recepción podríamos recuperar ese secreto "dividiendo" por esa misma constante. Pero, en aritmética modular está prohibido dividir porque podría darnos como resultado un número con decimales, y como el cuerpo de cifra son los números enteros, esto no es válido.

No obstante, sí está permitido multiplicar por el inverso, si este existe. Volviendo al ejemplo, si en el cifrado se multiplica el secreto o código del texto en claro por una constante a, en el descifrado no estará permitido dividir el criptograma por a, pero sí multiplicarlo por a^{-1}, el inverso de a en el cuerpo de cifra, solo si este existe.

Para que exista el inverso de a en el cuerpo de cifra n, inv (a, n), deberá cumplirse:

$$mcd\ (a, n) = 1$$

Esto asegura que el inverso del resto a en el cuerpo n existe y que, además, es único.

inv (15, 49) sí existe porque mcd (15, 49) = 1, ya que 15 = 3 * 5 y 49 = 7 * 7

inv (21, 84) no existe porquemcd (21, 84) = 7, ya que 21 = 3 * 7 y 84 = 7 * 12

Si el módulo n es un primo, entonces todos sus restos a excepción del 0 y del 1, tendrán un inverso porque los restos y el módulo serán primos relativos o coprimos.

El siguiente ejemplo de la figura 1.2 muestra el significado de un inverso. En este caso, el inverso de 3 en módulo 10, que se obtiene tras multiplicar el valor 3 por todos los restos en módulo 10:

Multiplicación de **3** por todos los restos mod **10**				
3*0 mod 10 = 0	3*1 mod 10 = 3	3*2 mod 10 = 6	3*3 mod 10 = 9	3*4 mod 10 = 2
3*5 mod 10 = 5	3*6 mod 10 = 8	**3*7 mod 10 = 1**	3*8 mod 10 = 4	3*9 mod 10 = 7

Figura 1.2. Restos de 3*X mod 10.

Como se observa, el único resultado donde obtenemos la identidad de la multiplicación (el valor 1) es 3*7 mod 10 = 21 mod 10 = 1. Por lo tanto, como 3*7 mod 10 = 1, entonces 7 es el inverso de 3 en módulo 10 y, por su parte, 3 es el inverso de 7 en módulo 10.

Aunque para números tan pequeños se puede usar este método de fuerza bruta, para el cálculo de inversos se usará siempre el Algoritmo Extendido de Euclides AEE.

No obstante, en módulo 10 el 6 no tendrá inverso porque mcd (6, 10) = 2.

En este caso tenemos la tabla de la figura 1.3, en donde se observa que no se consigue el valor 1 buscado y los restos se repiten de forma cíclica.

Multiplicación de **6** por todos los restos mod **10**				
6*0 mod 10 = 0	6*1 mod 10 = 6	6*2 mod 10 = 2	6*3 mod 10 = 8	6*4 mod 10 = 4
6*5 mod 10 = 0	6*6 mod 10 = 6	6*7 mod 10 = 2	6*8 mod 10 = 8	6*9 mod 10 = 4

Figura 1.3. Restos de 6*X mod 10.

La solución para encontrar el inverso de un resto dentro de un cuerpo, independientemente de que trabajemos con números muy grandes será el Algoritmo Extendido de Euclides.

Algoritmo Extendido de Euclides AEE

Hacer (g_0, g_1, u_0, u_1, v_0, v_1, i) = (B, A, 1, 0, 0, 1, 1)

Mientras $g_i \neq 0$ hacer

Hacer y_{i+1} = parte entera (g_{i-1}/g_i)

Hacer $g_{i+1} = g_{i-1} - y_{i+1} * g_i$

Hacer $u_{i+1} = u_{i-1} - y_{i+1} * u_i$

Hacer $v_{i+1} = v_{i-1} - y_{i+1} * v_i$

Hacer i = i+1

Si ($v_{i-1} < 0$)

Hacer $v_{i-1} = v_{i-1} + B$

Hacer $x = v_{i-1}$

Vamos a encontrar el inverso de 9 en el cuerpo 275, esto es inv (9, 275).

Sabemos que el inverso existe pues mcd (9, 275) = 1, dado que $9 = 3^2$ y $275 = 5^2x11$.

i	y_i	g_i	u_i	v_i
0	-	**275**	1	0
1	-	**9**	0	1
2	30	5	1	-30
3	1	4	-1	31
4	1	1	2	**-61**
5	4	0	**-9**	**275**

Figura 1.4. Aplicación del AEE para el cálculo de inv (9, 275).

Recuerdo de las dos últimas operaciones en el último cálculo:

-9 = -1 - (4*2)

275 = 31 - [4*(-61)]

Como v_{i-1} = -61 ha salido un valor negativo, entonces:

X = - 61 + 275 = 214. Se puede comprobar que inv (9, 275) = 214, ya que 214 x 9 = 1.926 mod 275 = 7x275 + 1 mod 275 = 1.

Raíces primitivas dentro de un cuerpo

En el intercambio de claves de Diffie y Hellman así como en algoritmos de cifra y firma digital de Elgamal, será recomendable usar un valor α conocido como generador o raíz primitiva de un módulo primo. Pero, ¿qué es una raíz primitiva?

Las raíces primitivas de un primo, cuyos valores no se pueden conocer a priori, pero sí encontrarlos muy fácilmente, tienen la particularidad de que tomados como base y elevados a todos los restos del módulo, generan el Conjunto Completo de Restos, en este caso todos los restos excepto el 0; de ahí su nombre. Por ejemplo, el primo p = 17 tiene ocho raíces primitivas {3, 5, 6, 7, 10, 11, 12, 14}. Si elegimos el valor $\alpha = 5$ y lo elevamos a todos sus restos, se obtienen los valores de la figura 1.5.

$5^0 \bmod 17 = 1$	$5^1 \bmod 17 = 5$	$5^2 \bmod 17 = 8$	$5^3 \bmod 17 = 6$	$5^4 \bmod 17 = 13$
$5^5 \bmod 17 = 14$	$5^6 \bmod 17 = 2$	$5^7 \bmod 17 = 10$	$5^8 \bmod 17 = 16$	$5^9 \bmod 17 = 12$
$5^{10} \bmod 17 = 9$	$5^{11} \bmod 17 = 11$	$5^{12} \bmod 17 = 4$	$5^{13} \bmod 17 = 3$	$5^{14} \bmod 17 = 15$
$5^{15} \bmod 17 = 7$	$5^{16} \bmod 17 = 1$	Se obtienen todos los restos de 17, excepto el 0.		

Figura 1.5. Potencias del resto 5 en mod 17.

En cambio, el número 4 no es una raíz primitiva porque obtenemos ahora la tabla de la figura 1.6, en donde no se genera el cuerpo completo del primo p. Es más, como era de esperar, aparecen ciclos de números que se repiten, en este caso 1, 4, 16, 13.

$4^0 \bmod 17 = 1$	$4^1 \bmod 17 = 4$	$4^2 \bmod 17 = 16$	$4^3 \bmod 17 = 13$	$4^4 \bmod 17 = 1$
$4^5 \bmod 17 = 4$	$4^6 \bmod 17 = 16$	$4^7 \bmod 17 = 13$	$4^8 \bmod 17 = 1$	$4^9 \bmod 17 = 4$
$4^{10} \bmod 17 = 16$	$4^{11} \bmod 17 = 13$	$4^{12} \bmod 17 = 1$	$4^{13} \bmod 17 = 4$	$4^{14} \bmod 17 = 16$
$4^{15} \bmod 17 = 13$	$4^{16} \bmod 17 = 1$	No obtienen todos los restos de 17. Hay ciclos.		

Figura 1.6. Potencias del resto 4 en mod 17.

Las raíces primitivas se usan en criptografía porque, si el resultado de la operación de potencia es un valor público y el exponente es un valor secreto a proteger, dicho valor público solo puede provenir de una única operación y no de varias como sucede cuando α no es un generador de p.

Por ejemplo, si en la tabla de la Figura 1.5 el valor público fuese el 12, este proviene únicamente de la operación $5^9 \bmod 17 = 12$, en donde el valor 9 será nuestro secreto a proteger. Si alguien desea conocer nuestro secreto 9 conociendo que $\alpha = 5$ y que p = 17, podría hacerlo usando fuerza bruta calculando para todos los x posibles, el resultado de $\alpha^x \bmod p$, algo que para números muy grandes es computacionalmente imposible.

Otra opción pasa por despejar el valor de x del exponente en dicha ecuación, pero en este caso el atacante se enfrentará al problema del logaritmo discreto (PLD), también computacionalmente muy difícil para valores del primo p cerca de los mil bits, como veremos más adelante.

Exponenciación rápida

Una operación muy común en criptografía moderna es la de elevar a potencia dentro de un cuerpo, de la forma A^B mod n, o exponenciación. Si los números A, B y n son pequeños, por ejemplo 3^{12} mod 121, perfectamente podríamos resolver primero la potencia 3^{12} = 531.441 y luego reducir este valor módulo 121, para obtener 9.

¿Qué sucedería si estos valores son muy grandes, de centenas e incluso miles de bits? No podemos ahora resolver primero la potencia porque, entre otras cosas, no contamos con suficiente espacio para guardar un número tan grande. Puesto que en criptografía será común trabajar con módulos de 2.000 mil bits (unos 600 dígitos), resulta claro que debemos aplicar algunas propiedades de la aritmética modular para permitir realizar estas operaciones.

Uno de los métodos para resolver A^B mod n = x es de la exponenciación rápida que se muestra a continuación.

Representar el exponente B en formato binario de k bits: $b_{k-1}b_{k-2}...b_i...b_1b_0$
Hacer x = 1
Para i = k-1 hasta i = 0, hacer
  $x = x^2$ mod n
    Si (b_i = 1) entonces
      x = x*A mod n

Es decir, primero convertimos el exponente B a su valor binario. A continuación, en módulo n haremos operaciones consistentes en elevar al cuadrado restos de ese módulo y, en algunos casos y solo cuando corresponda, multiplicaremos además ese valor por la base A. Como estas operaciones se hacen desde i = k-1 hasta i = 0, solamente realizaremos k operaciones, siendo k el número de bits del exponente. Esto reduce de manera drástica el total de operaciones a realizar si lo comparamos con el básico de reducción sucesiva a módulo para calcular A^B mod n, haciendo primero A^2 mod n, el resultado multiplicarlo por A mod n, y así sucesivamente hasta B-1 veces.

Por ejemplo, si la cifra es 17^{300} mod 40.555, dado que 300 en binario es el valor de 9 bits $b_8b_7b_6b_5b_4b_3b_2b_1b_0$ = 100101100, siguiendo el algoritmo anterior, obtenemos el resultado 3.066 como se muestra en la figura 1.7.

$b_8 = 1$	$x = 1^2 * 17 \bmod 40.555$	$x = 17$
$b_7 = 0$	$x = 17^2 \bmod 40.555$	$x = 289$
$b_6 = 0$	$x = 289^2 \bmod 40.555$	$x = 2.411$
$b_5 = 1$	$x = 2.411^2 * 17 \bmod 40.555$	$x = 27.677$
$b_4 = 0$	$x = 27.677^2 \bmod 40.555$	$x = 13.489$
$b_3 = 1$	$x = 13.489^2 * 17 \bmod 40.555$	$x = 32.652$
$b_2 = 1$	$x = 32.652^2 * 17 \bmod 40.555$	$x = 5.498$
$b_1 = 0$	$x = 5.498^2 \bmod 40.555$	$x = 14.529$
$b_0 = 0$	$x = 14.529^2 \bmod 40.555$	$x = 3.066$

Figura 1.7 Aplicación de algoritmo de exponenciación rápida.

El método básico de multiplicación y reducción sucesiva a módulo habría aquí significado 299 multiplicaciones.

El algoritmo de exponenciación rápida aplicado anteriormente puede entenderse en este ejemplo observando que para resolver $17^{300} \bmod 40.555$, habría que realizar 150 veces la operación $17^2 \bmod 40.555$, pero como son todos los valores iguales a 289, se hace una sola vez y se guarda el valor. A continuación, con los 150 resultados 289 habría que hacer 75 operaciones $289^2 \bmod 40.555$ que es igual a 2.411. De estos 75 resultados 2.411, se harán 37 operaciones $2.411^2 \bmod 40.555$ y una multiplicación por 17 (al ser impar), etc. Si se sigue este proceso, al final esto va conformando las operaciones de los ceros (cuadrados) y los unos (cuadrados y multiplicación por la base).

Observa que, aunque 17^{300} sea un número de 1.227 bits (unos 370 dígitos), sigue siendo extremadamente pequeño para lo que es usual en criptografía.

1.4. Uso de problemas matemáticos en la criptografía

Existen varios problemas matemáticos definidos como funciones de un solo sentido. Se denominan así porque si bien el cálculo directo de dicha función es muy rápido y tiene un coste computacional bajo; por el contrario, el cálculo inverso de la misma se vuelve intratable cuando los números son grandes, un fenómeno conocido como comportamiento no polinomial, o bien comportamiento exponencial, dado que el tiempo de resolución crece de esa forma en función del tamaño de los datos de entrada.

Los dos problemas de un solo sentido más usados en la criptografía moderna son el problema del logaritmo discreto (PLD) y el problema de la factorización entera (PFE).

En el PLD será muy fácil calcular el valor de *y* en la expresión $y = \alpha^x \bmod p$. Sin embargo, si se conoce el valor de *y* pero no se conoce el exponente *x* y se desea encontrarlo, nos enfrentaremos a $x = log_\alpha\ y \bmod p$, cuya resolución se vuelve intratable cuando los números son muy grandes.

$y = \alpha^x \bmod p$ [conocidos *α, x, p, el* cálculo de y es fácil y rápido, independiente del tamaño]

$x = log_\alpha\ y \bmod p$ [conocidos *α, y, p, el* cálculo de x es difícil y lento para números grandes]

Por ejemplo, si vamos aumentando el tamaño del primo p, el cálculo de $y = \alpha^x \bmod p$ tarda cada vez más en encontrar el resultado, pero lo hace de una forma polinomial (lineal). Es lógico, cada vez hay más operaciones bit y, por lo tanto, se tardará más tiempo. Pero, por el contrario, si vamos aumentando el tamaño del primo p en la expresión $x = \log_\alpha y \bmod p$, el tiempo de resolución para encontrar x se vuelve no polinomial (exponencial).

Este problema matemático del PLD es en lo que fundamentaron la seguridad de la propuesta del intercambio de clave Diffie y Hellman, y que dio nacimiento a la criptografía de clave pública. El exponente x representará al valor privado (secreto) que elige cada uno de los interlocutores en dicho protocolo.

El Problema de la factorización Entera PFE puede adaptarse a la criptografía, e.g. algoritmo RSA, diciendo que la multiplicación de dos números primos (p * q = n) es lineal, pero si a partir del número compuesto n queremos encontrar los factores primos p y q, la resolución de este problema tiene un comportamiento exponencial. En realidad, el PLD y el PFE son muy similares en cuanto a su complejidad algorítmica.

Las ecuaciones que rigen el PFE son:

$n = p * q$ [conocidos *p, q, el* cálculo de n es fácil y rápido, independiente del tamaño]

$p = n/q\ o\ q = n/p$ [conocido n, *el* cálculo de p o q es difícil y lento para primos muy grandes]

En ambos casos existe un primer y básico algoritmo de ataque por fuerza bruta. En el PLD podríamos intentar en la ecuación $y = \alpha^x \bmod p$ todos los valores posibles de x {2, p-2} hasta dar con el valor que nos entregue la igualdad. Y en el PFE podríamos dividir el valor n por todos los números primos comprendidos entre 2 y $\sqrt{n}$ (criba de Eratóstenes). Pero en ambos casos se trata del peor algoritmo posible en cuanto a velocidad y eficiencia y, además, el que más cómputo consume.

Tanto para el PLD como para el PFE existen algoritmos mucho más eficientes que estos de fuerza bruta, pero ninguno es capaz de superar ese comportamiento no polinomial (exponencial), que los vuelve intratables para la actual capacidad de cómputo mundial cuando los números se acercan a los mil bits.

1.5. Seguridad de los algoritmos criptográficos

Los algoritmos de cifra pueden atacarse por fuerza bruta; esto es, intentar descifrar el criptograma con todos los valores posibles de la clave hasta dar con la verdadera, o bien atacarlos de una forma más elegante haciendo uso del criptoanálisis.

Los sistemas clásicos permitirán ataques mediante criptoanálisis, bien utilizando las estadísticas del lenguaje conocidas desde hace siglos y refrendadas además por Shannon en su estudio, o bien aplicando herramientas matemáticas.

Sin embargo, para los sistemas de cifra modernos, el único tipo de ataque que en la práctica funciona es el ataque por fuerza bruta, en tanto no cabe aquí aplicar estadísticas del lenguaje debido el tipo de cifra y el hecho de que otros tipos de ataques conocidos y publicados (por ejemplo, criptoanálisis diferencial) no dejan ser meramente teóricos.

En realidad, el problema que subyace con estos sistemas de cifra es que los sistemas clásicos son lineales, y por ello fáciles de atacar, y los sistemas modernos son no lineales y difíciles de atacar.

Así, podrán existir ataques conociendo solo el texto cifrado o bien incluso conociendo, además, el texto en claro. Cualquier algoritmo que se precie deberá soportar un ataque con texto en claro conocido; es decir, el atacante conoce el criptograma, conoce el texto en claro y obviamente conoce el algoritmo de cifra (el código fuente es público) pero, así y todo, será incapaz de encontrar la clave de cifrado K en un tiempo razonable.

1.6. Nociones de teoría de la información

Basándonos en el trabajo de Claude Shannon sobre Teoría de la Información, conviene destacar dos conceptos que nos serán de utilidad en este curso, el de la entropía de la información H(x) y el de la redundancia del lenguaje D(x). No se profundizará en ello y por lo tanto este apartado tiene como objetivo solo la presentación de estas nociones básicas.

Información es el conjunto de datos o mensajes inteligibles creados con un lenguaje de representación, que desde el punto de vista de la criptografía debemos proteger ante las amenazas del entorno en todo su ciclo de vida, a saber, durante su generación, su transmisión, su almacenamiento e incluso su destrucción. Por su parte, la teoría de la información mide la cantidad de información que contiene un mensaje a través del número medio de bits necesario para codificar todos los posibles mensajes con un codificador óptimo.

La entropía mide el desorden de las cosas, a mayor desorden o equiprobabilidad de los eventos, mayor entropía. En criptografía será muy importante que la clave tenga una alta entropía para que sea difícil adivinarla.

Por ejemplo, una clave en hexadecimal de 64 bits tendrá mayor entropía que una clave escrita en ASCII, simplemente porque en hexadecimal serán válidas las 2^{64} claves, desde la clave menor de 16 ceros 0000000000000000, hasta la clave mayor de 16 efes FFFFFFFFFFFFFFFF. En cambio, en ASCII no tendremos el mismo espacio de claves equiprobables, primero porque los humanos tendemos a poner caracteres o palabras que nos sean de fácil recuerdo y, segundo, porque habrá además algunos caracteres ASCII no imprimibles para esos 64 bits u ocho bytes de la clave, por lo que el espacio de claves en este caso será menor que 2^{64}.

Es conocido que la codificación ASCII no es óptima dado que asigna 8 bits para codificar caracteres muy frecuentes (espacio en blanco, la letra e, la letra a, etc.), y también 8 bits para codificar caracteres muy poco frecuentes (la letra k, la letra á, el signo %, etc.). Asignar en la codificación la misma cantidad de bits a todos los caracteres ASCII, equivale a aceptar que estos son equiprobables, lo cual sabemos que no es cierto.

En un alfabeto español de 27 letras, contando solo con las letras mayúsculas como el que se utilizaba en la criptografía clásica, si esas 27 letras tienen igual probabilidad o frecuencia de aparición en un texto en claro (lo cual no es cierto), la entropía y la ratio del lenguaje r vendrán dada por el valor $\log_2 27 = 4{,}75$ bits/letra, conocida como ratio absoluta o R.

Esto quiere decir que con 5 bits podremos codificar esos 27 caracteres, lo cual resulta obvio porque $2^4 = 16 < 27 < 2^5 = 32$. Pero si usamos una codificación óptima, asignando códigos cortos a las letras más frecuentes, nos encontramos con que esa ratio del lenguaje r es menor que 1,5 bits/letra.

Esto no quiere decir que podamos codificar todos los elementos de ese alfabeto de 27 letras con solo 2 bits; es evidente que no. Lo que viene a indicar este valor tan bajo es que la cantidad de información que posee cada letra debido a la redundancia del lenguaje, es de tan solo 1,5 bits.

A este resultado se llega codificando un mensaje solo con 27 letras pero asignando, una vez leído el mensaje, códigos cortos a los caracteres más repetidos y códigos más largos a los poco frecuentes, por ejemplo mediante el sistema de codificación de Huffman con la construcción de un árbol binario para asignar bits a los elementos.

Así, la redundancia del lenguaje D(x) en este caso será D(x)= R - r = 4,75 - 1,5 = 3,25. Esto significa que de los 5 bits que se utilizarían para codificar esas 27 letras, son innecesarios 3,25 bits.

Si hacemos la razón D(x)/R = 3,25/4,75 obtenemos 68,4 %. Si hablamos de bytes, se ahorra casi un 32% del espacio de almacenamiento, un porcentaje que nos recuerda mucho a la razón de compresión alcanzada en un archivo cuando le aplicamos una función zip. Aquí se manifiesta claramente la redundancia del lenguaje.

1.7. Codificación de la información

Es bastante común oír que códigos secretos es un sinónimo de criptografía, pero esto no es correcto puesto que codificar no es lo mismo que cifrar.

Codificar es una acción estática, en tanto que el código siempre será el mismo, no cambia con el tiempo, ni con las personas, ni con los países (por ejemplo, el ASCII de A = 0100 0001 siempre).

En cambio, la acción de cifrar es una acción dinámica, en el sentido de que en función de que se use una u otra clave en la cifra, el criptograma será lógicamente uno u otro.

Existen muchos códigos, como por ejemplo el código Morse, Baudot, ASCII normal, ASCII extendido, decimal, hexadecimal, binario, octal, Base64, etc., siendo algunos de ellos muy comunes en la criptografía, como por ejemplo Base64.

Test del capítulo 1

1. Los fundamentos de la criptografía residen en:

A. Teoría de la información, reparto de secretos, matemáticas discretas

B. Aritmética modular, teoría de la computación, complejidad algorítmica

C. Matemáticas discretas, teoría de la información, complejidad algorítmica

D. Complejidad algorítmica, aritmética modular, inteligencia artificial

2. Si se cumple que $E_K(E_K(M)) = M$, diremos que:

A. La clave K es semidébil

B. La clave K es débil

C. La clave K es fuerte

D. Ninguna de las anteriores

3. Los tres hitos que han marcado el paso de la criptografía clásica a la moderna son:

A. El algoritmo DES, el algoritmo AES y el algoritmo RSA

B. Los estudios de Shannon, el algoritmo DES, el algoritmo RSA

C. El algoritmo DES, el intercambio de clave de DH, las funciones hash.

D. Los estudios de Shannon, el algoritmo DES, el intercambio de clave de DH

4. La diferencia entre sistemas de cifra simétrica y cifra asimétrica es que:

A. La simétrica usa una clave para cifrar y otra distinta para descifrar

B. La asimétrica usa una clave para cifrar y otra distinta para descifrar

C. La simétrica cifra en bloques y la asimétrica lo hace en flujo

D. La simétrica cifra en flujo y la asimétrica lo hace en bloques

5. La congruencia correcta es:

A. -18 mod 20 = 2

B. 25 mod 31 = 6

C. 33 mod 10 = 4

D. -9 mod 6 = 2

6. El inverso multiplicativo de 5 en módulo 27 es:

A. 10

B. 7

C. 4

D. 11

7. Aplicando el AEE para encontrar el inv (17, 39.900) llegamos al valor -2.347:

A. El inverso buscado es entonces 2.347

B. El inverso buscado es entonces 27.553

C. El inverso buscado es entonces 37.347

D. El inverso buscado es entonces 37.553

8. Un número α es una raíz primitiva del primo p si:

A. Es número impar y genera todos los restos del primo p

B. Es número par y genera todos los restos del primo p

C. Es número par o impar y genera todos los restos del primo p

D. Es un número que no genera todos los restos del primo p

9. En un intercambio de clave se usa la ecuación $2^x \bmod 19 = 7$.

A. El valor privado x es igual a 6

B. El valor privado x es igual a 5

C. El valor privado x es igual a 4

D. El valor privado x es igual a 3

10. Un algoritmo se considera seguro si y solo si soporta un ataque:

A. Por redundancia del lenguaje

B. Por código fuente conocido

C. Por criptograma conocido

D. Por texto en claro conocido

Tema 2

Criptografía clásica y cifra moderna

2.1. Introducción

2.2. Principios de Kerckhoffs

2.3. Clasificación de los sistemas de cifra clásica

2.4. Cifrado por permutación

2.5. Cifrado por sustitución

2.6. Cifrado por matrices

2.7. Características de los sistemas de cifra modernos

2.8. Cifra simétrica versus cifra asimétrica

2.9. Usos de la criptografía moderna

2.1. Introducción

Para profundizar en este tema, lee el documento Criptografía Clásica y el capítulo 9 del siguiente libro.

Ramió, J. (2006). Libro electrónico de seguridad informática y criptografía, versión 4.1.

Disponible en: http://www.criptored.upm.es/guiateoria/gt_m001a.htm

Lee también las lecciones 4, 6, 7, 8, 9 y 10 del siguiente libro:

Ramió, J. (2016). Introducción a la seguridad informática y la criptología clásica.

Disponible en:

http://www.criptored.upm.es/crypt4you/temas/criptografiaclasica/leccion1.html

Se recomienda la visualización de las siguientes píldoras formativas Thoth del mismo autor:

Píldora 3: ¿Desde cuándo existe la criptografía?

Píldora 4: ¿Por qué hablamos de criptografía clásica y de criptografía moderna?

Píldora 7: ¿Qué son los principios de Kerckhoffs?

Píldora 8: ¿Qué relación existe entre Alan Turing y la criptografía?

Píldora 9: ¿Por qué busca la criptografía la confusión y la difusión?

Píldora 10: ¿Cómo se clasifican los sistemas de cifra clásica?

Píldora 13: ¿Qué es la cifra por sustitución monoalfabética?

Píldora 14: ¿Qué es la cifra por sustitución polialfabética?

Píldora 15: ¿Qué es la cifra por transposición o permutación?

Píldora 16: ¿Qué es la cifra del César?

Píldora 17: ¿Qué es la cifra afín?

Píldora 18: ¿Cómo se ataca la cifra por sustitución monoalfabética?

Píldora 19: ¿Qué es la cifra de Vigenère?

Píldora 20: ¿Cómo se ataca por Kasiski la cifra de Vigenère?

Píldora 21: ¿Qué es la cifra por matrices de Hill?

Píldora 22: ¿Cómo se ataca por Gauss-Jordan la cifra de Hill?

Píldora 26: ¿Cómo se clasifican los sistemas de cifra moderna?

Píldora 27: ¿Qué es mejor, la criptografía simétrica o la asimétrica?

Píldora 31: ¿Qué son los rellenos y los modos de cifra en bloque?

Píldora 36: ¿Qué es el código Base64?

Disponibles en:

https://www.youtube.com/watch?v=7MqTpfEreJ0&list=PL8bSwVy8_IcNNS5QDLjV7gUg8dIeMFSER

En este capítulo conoceremos los principios de Kerckhoffs sobre la seguridad en los sistemas de cifra, un conjunto de observaciones que hay que cumplir para lograr el objetivo principal de la seguridad. Este objetivo no es otro que proteger a la información. Estos principios fueron presentados en el año 1883, pero siguen vigentes a fecha de hoy.

Clasificaremos los sistemas de cifra clásica y presentaremos los sistemas de cifra por permutación, los sistemas de cifra por sustitución y el cifrado por matrices.

Además, profundizaremos en los conceptos relacionados con la criptografía moderna, analizando las razones de su clasificación, comparando la criptografía simétrica con la asimétrica, para terminar observando el uso de estos algoritmos en protocolos y servicios de seguridad.

Puesto que lo más importante de esta asignatura son lógicamente los sistemas de cifra modernos, que trataremos al final de este capítulo y en los tres siguientes, en este capítulo solo se presentarán los principios de la cifra clásica de una manera breve, dejando al alumno que profundice en estos temas —si tiene interés en ello— mediante un extenso conjunto de 16 píldoras formativas Thoth, así como el software de prácticas Criptoclásicos V 2.1

Accede al *software* Criptoclásicos V 2.1 desde la siguiente dirección web:
http://www.criptored.upm.es/software/sw_m001c.htm

La Figura 2.1 muestra una captura de pantalla de este programa tras prosperar un criptoanálisis a la cifra de Vigenère mediante el método de Kasiski.

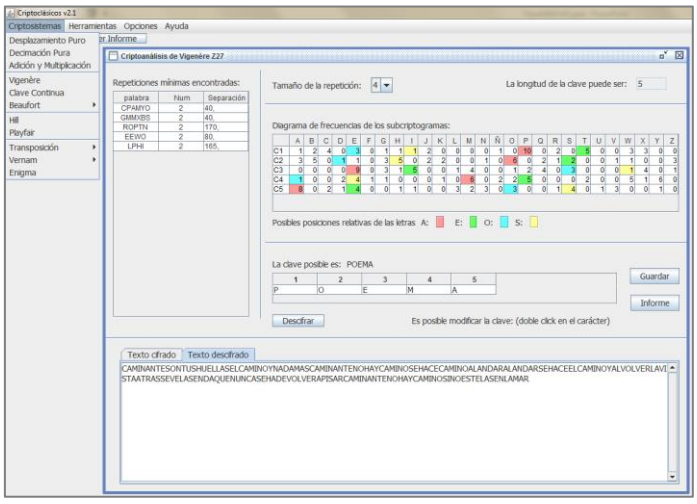


Figura 2.1. Software de prácticas de criptografía clásica Criptoclásicos V 2.1.

2.2. Principios de Kerckhoffs

En su traducción literal, los seis principios que el criptógrafo holandés Auguste Kerckhoffs propone en 1883 sobre la criptografía, son los siguientes:

1. El sistema debe ser en la práctica indescifrable, en caso de que no lo sea matemáticamente.
2. El sistema no debe ser secreto y no debe ser un problema que éste caiga en manos del enemigo.
3. La clave del sistema debe ser fácil de memorizar y comunicar a otros, sin necesidad de tener que escribirla. Será además cambiable y modificable por los interlocutores válidos.
4. El sistema debe poder aplicarse a la correspondencia telegráfica.
5. El sistema debe ser portable y su uso no deberá requerir la intervención de varias personas.
6. El sistema debe ser de fácil uso, no requerirá conocimientos especiales ni tendrá una larga serie de reglas o instrucciones.

La aportación más importante de los Principios de Kerckhoffs es la segunda, que dice: “el sistema no debe ser secreto y no debe ser un problema que este caiga en manos del enemigo”.

Hoy en día hemos simplificado su enunciado, diciendo que: “la seguridad del sistema debe recaer solamente en la clave”, todo lo demás deberá ser público.

2.3. Clasificación de los sistemas de cifra clásica

La clasificación principal de los sistemas de cifra clásica atiende al tipo de operación que se realizará al texto en claro durante la cifra, bien sea la de transposición para lograr la difusión, o bien la de sustitución para lograr la confusión. La transposición consiste en cambiar las letras del lugar que ocupan en el texto en claro y la sustitución consiste en cambiar una letra o conjunto de letras del texto en claro por otra letra u otro conjunto de letras de un alfabeto de cifrado.

La cifra por permutación tiene un corto recorrido en la historia de la criptografía clásica, no así la de sustitución, en donde aparecen conceptos como cifra monoalfabética y cifra polialfabética, así como la cifra monográmica y la cifra poligrámica.

En la Figura 2.2. se recoge esta clasificación de la cifra clásica.

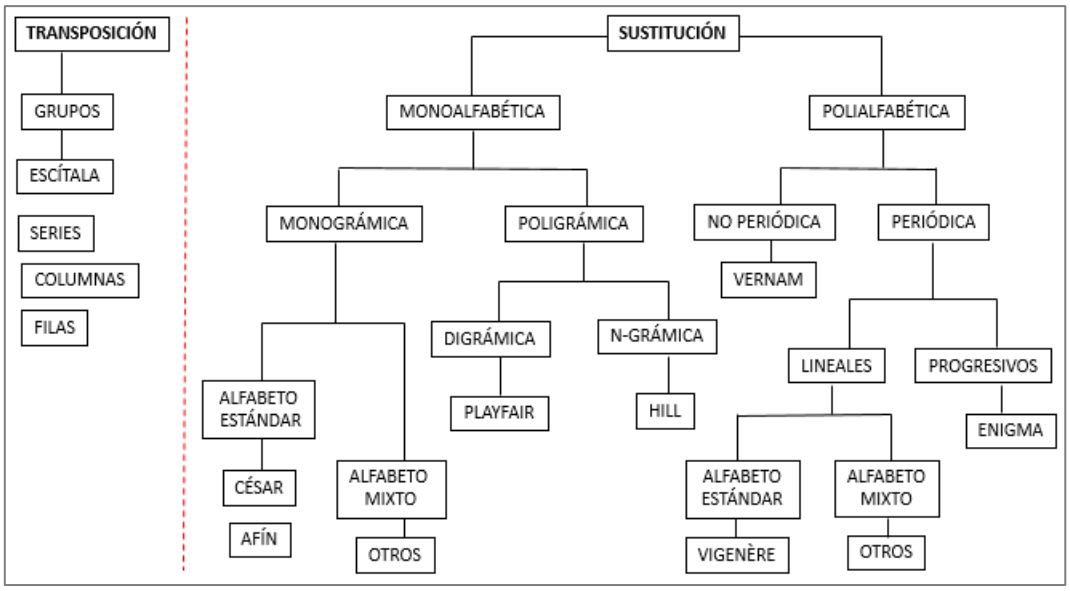


Figura 2.2. Clasificación de los sistemas de cifra clásica.

Todos estos métodos y principios se siguen utilizando hoy en día, obviamente ya no sobre letras sino sobre bits y bytes y, en su caso, trabajando con el código ASCII extendido. Los sistemas de cifra actuales utilizan tanto las técnicas de transposición como las de sustitución, y son "monográmicos" en tanto algunos sistemas cifran en flujo bits o bytes (e.g. A5 y RC4) o bien "poligrámicos", cuando la cifra se realiza por bloques (e.g. AES, 3DES).

Por este motivo, es normal e incluso recomendable, dedicar una hora de clase a estos temas de criptografía clásica, de fácil comprensión y con una clara influencia en la cifra moderna.

2.4. Cifrado por permutación

La escítala, usada en el siglo V a. C. por el pueblo griego de los lacedemonios, es el sistema de cifra más antiguo que se conoce. Consistía en un bastón de madera en el que se enrollaba una cinta de cuero y luego se escribía en ella de forma longitudinal el mensaje que se deseaba esconder. Hecho esto, al desenrollar la cinta las letras aparecerán desordenadas. Para descifrar el criptograma y recuperar en recepción el texto en claro, había que enrollar dicha cinta en un bastón con el mismo diámetro que el usado en el extremo emisor y tras ello leer el mensaje de forma longitudinal.

La clave del sistema se encontraba obviamente en el diámetro del bastón. Se trata de entonces de una cifra por transposición pues los caracteres del criptograma son los mismos que en el texto en claro, pero están distribuidos de otra forma dentro del criptograma.

Con el paso del tiempo fueron apareciendo otros sistemas, entre ellos la cifra por columnas y la cifra por filas.

En la cifra por columnas se escribe el texto en claro de manera horizontal pasando a la siguiente fila cuando se ha alcanzado un número de columnas, que será la clave, igual a 5 en el caso de la Figura 2.3. El criptograma TPIRM ROOCN ASNOA NIPLS SCOUX resulta de la lectura por columnas del texto en claro TRANSPOSICION POR COLUMNAS.

Antiguamente era común agrupar el criptograma en bloques de 5 letras.

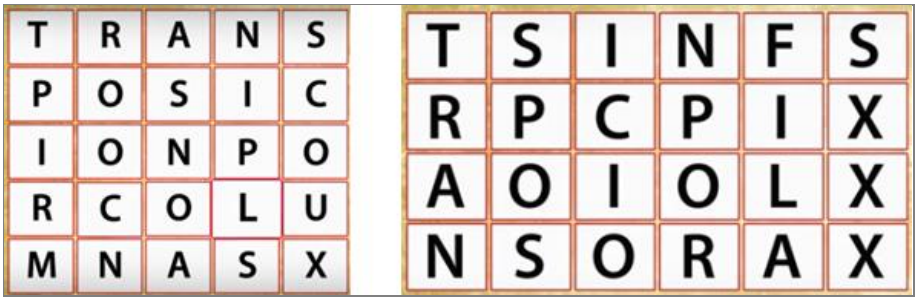


Figura 2.3. Cifrado por columnas (izquierda) y cifrado por filas (derecha).

Por el contrario, en la cifra por filas se escribe el texto en claro en vertical y se pasa a la columna siguiente cuando se ha alcanzado un número de filas o clave, que en este caso en la Figura 2.3 es igual a 4. El criptograma resultante TSINF SRPCP IXAOI OLXNS ORAX es la lectura por filas del texto en claro TRANSPOSICION POR FILAS.

Observa que, al trabajar dentro de un rectángulo, es muy posible que el texto en claro no quepa exactamente en él y, por tanto, debamos incluir rellenos. Como se observa en la Figura 2.3, esto se ha hecho usando la letra X que se supone conocen emisor y receptor. En los sistemas de cifra simétrica modernos también se usará este relleno (en este caso bits o bytes) al cifrar en bloques.

2.5. Cifrado por sustitución

Cifrado del César

	0										10										20						
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
M	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Figura 2.4. Alfabeto de cifrado del César mod 27 (M = Mensaje, C = Criptograma).

En el siglo I a. C., Julio César usaba este tipo de cifrado. El algoritmo consiste en el desplazamiento de 3 espacios hacia la derecha de los caracteres del texto en claro, como se muestra en la figura 2.4. Es un cifrador por sustitución monoalfabético en el que las operaciones se realizan módulo n, siendo n el número de elementos del alfabeto (en aquel entonces el latín). Por ejemplo:

M = CÉSAR DESPLAZA LAS LETRAS DEL TEXTO TRES ESPACIOS
C = FHVDU GHVSÑ DCDÑD VÑHWU DVGHÑ WHAWR WUHVH VSDFL RV

El descifrado es elemental: se desplazan las letras del criptograma 3 espacios a la izquierda. Matemáticamente podemos representar la cifra genérica tipo César, o por desplazamiento puro, en módulo n con las siguientes expresiones:

$$c = m + b \bmod n$$
$$m = c - b \bmod n$$

Cifrado afín

Si además de desplazar el texto como en el caso del César también multiplicamos el texto en claro por una constante, nos encontramos ante el cifrado afín, con una clave algo mayor y cuyas ecuaciones de cifrado y descifrado son:

$$c = a * m + b \bmod n$$
$$m = (c - b) * inv(a, n) \bmod n$$

Observa que no es válido escribir m = (c – b)/a mod n, pero en cambio sí es válido m = (c – b) * a^{-1} mod n, o como indica la ecuación m = (c – b) * inv (a, n) mod n, siempre que dicho inverso exista.

Por ejemplo, si en módulo 27 ciframos el mensaje de cinco letras AMIGO con las constantes a = 2 y b = 5, se obtiene el criptograma C = FCUQI como se muestra en la figura 2.5.

Texto m	A	M	I	G	O
Alfabeto	0	12	8	6	15
x 2 mod 27	0	24	16	12	3
+ 5 mod 27	5	2	21	17	8
Criptograma c	F	C	U	Q	I

Figura 2.5. Cifrado afín mod 27 con a = 2 y b = 5.

Para criptoanalizar la cifra afín, podemos usar las estadísticas y redundancia del lenguaje que se manifiestan en el criptograma. Planteamos dos ecuaciones independientes para poder encontrar las incógnitas a y b.

Dado el siguiente criptograma C = SVPSW P**M**DNA SGH**M**X **M**X**M**BH **M**KSTS CNBB**M** B, observamos que las letras más frecuentes del criptograma son la letra M (en negrita) con 6 apariciones (19 %) y la letra S (subrayada) con 5 apariciones (16 %). Si suponemos que corresponden a la cifra de las letras más frecuentes del alfabeto español, la E y la A respectivamente:

M = a*E + b mod 27
S = a*A + b mod 27

Como:

A = 0, E = 4, M = 12, S = 19
19 = a*0 + b mod 27. (b = 19)

Reemplazamos b: 12 = a*4 + 19 mod 27

-7 = 20 = a*4 mod 27
a = 20 inv (4, 27) mod 27 = 20*7 mod 27 = 5 (a = 5)

Descifra tú mismo este sabio proverbio.

Cifrado de Vigenère

Se trata de uno de los más famosos algoritmos de cifra clásica, inventado por el criptógrafo francés Blaise de Vigenère 1586, que se resistió al criptoanálisis durante más de 250 años. Es una cifra polialfabética al contar con más de un alfabeto de cifrado. Las ecuaciones que rigen el cifrado y el descifrado en módulo n son:

$$c_i = m_i + k_i \bmod n$$
$$m_i = c_i - k_i \bmod n$$

Donde m_i y k_i son las letras en la posición i del texto en claro y de la clave, como se muestra en el ejemplo de la figura 2.6 al cifrar el texto RESISTIÓ TRESCIENTOS AÑOS con la clave SEGURO en mod 27.

Texto m	R	E	**S**	I	**S**	T	I	O	…
Alfabeto	18	4	19	8	19	20	8	15	…
Clave	S	E	G	U	R	O	S	E	…
Alfabeto	19	4	6	21	18	15	19	4	…
$m_i + k_i$ mod 27	10	8	25	2	10	8	0	19	…
Criptograma c	K	I	Y	C	K	I	A	S	…

Figura 2.6. Cifrado de Vigenère mod 27 del ejemplo.

M = RESISTIÓ TRESCIENTOS AÑOS

C = KIYCK IASZM VHUMK HLDLE TJK

Observa que la letra S (negrita) del texto en claro se cifra como primero Y después como K, y que la letra I (subrayada) del criptograma proviene de la cifra de dos letras distintas del texto en claro, la letra E y la letra T, obviamente por la posición de la letra de la clave en el momento de la cifra.

Debido a lo anterior, con este tipo de cifra las estadísticas del lenguaje ya no se manifiestan tan claramente en el criptograma. Es más, si la clave tiene 8 o más letras, las frecuencias de las letras en el criptograma son todas muy similares y la distribución de frecuencias tiende a ser uniforme. No obstante, el militar alemán Friedrich Kasiski observa en 1863 que la redundancia del lenguaje no se manifiesta solo en textos que se leen letra a letra (lectura normal y lógica), sino que también lo hace cuando de dicho texto en claro se van leyendo letras en saltos, cada x posiciones, que es en el fondo lo que hace la cifra de Vigenère, en tanto la clave se va repitiendo y se usan de esta manera los mismos alfabetos de cifrado cada x posiciones o longitud de la clave.

Por lo tanto, las repeticiones de tres o más letras en el criptograma deberían provenir obligatoriamente de n-gramas muy comunes en el lenguaje como, por ejemplo, terminaciones del tipo ado, ido, ando, iendo, mente, así como otras combinaciones comunes de letras en las palabras del lenguaje utilizado.

Los pasos a seguir en el ataque de Kasiski son:

1. Buscar en el criptograma repeticiones de al menos 3 caracteres y anotar la distancia que separa a todas esas repeticiones.
2. Encontrar el máximo común divisor de todas esas separaciones. El mcd nos indicará la posible longitud L de la clave.
3. Se procede a dividir el criptograma en L subcriptogramas tomando las letras de L en L espacios.
4. Para cada uno de los L subcriptogramas, se apunta la frecuencia de aparición de cada letra.

5. Se busca en cada uno de los L subcriptogramas las cuatro frecuencias más altas y que, además, cumplan con la distancia que separa a las letras con mayor frecuencia del alfabeto español mod 27, es decir la A, la E, la O y la S. Esto es, que los espacios entre ellas cumplan la siguiente distribución: Letra A -> + 4 = Letra E -> + 11 = Letra O -> + 4 = Letra S.
6. Ubicada la posición de la Letra A, que es la relativa a la letra A del texto en claro y cuyo código es igual a 0, se mira con qué letra se ha cifrado, dando así la letra correspondiente de la clave en esa posición.
7. Se repite este proceso con todos los subcriptogramas para obtener la clave buscada.

2.6. Cifrado por matrices

$$\begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ .. \\ C_N \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & k_{13} & \dots & k_{1N} \\ k_{21} & k_{22} & k_{23} & \dots & k_{2N} \\ k_{31} & k_{32} & k_{33} & \dots & k_{3N} \\ .. & .. & .. & & .. \\ k_{N1} & k_{N2} & k_{N3} & \dots & k_{NN} \end{pmatrix} X \begin{pmatrix} M_1 \\ M_2 \\ M_3 \\ .. \\ M_N \end{pmatrix} \quad \text{mod n}$$

A comienzos del siglo XX, Lester Hill inventa un sistema de cifra ngrámico, utilizando matrices y siendo la ecuación de cifrado como se muestra arriba.

En donde C_i y M_i serán las letras del criptograma y texto en claro, respectivamente, y k_{ij} los valores de la clave, lógicamente números dentro del cuerpo de cifra n. Observa que la matriz debe ser cuadrada; es decir, si ciframos cada dos letras (digramas), la matriz clave tendrá una dimensión de 2x2; si ciframos cada tres letras (trigramas), la matriz clave será de dimensión 3x3, etc.

Puesto que la cifra se hace en módulo n y para descifrar hay que hacer la misma operación que hacíamos en la cifra afín (en vez de dividir, multiplicar por el inverso), en este caso la matriz clave K debe tener inversa, es decir, es necesario que exista K^{-1} mod n. Para que esto se cumpla, el determinante de K no debe ser cero (no ser singular) y además no debe tener factores en común con el módulo n, es decir, mcd (|K|, n) = 1.

Como ejemplo cifraremos el mensaje AMIGO CONDUCTOR en bloques de 3 letras, con cifrado trigrámico, usando para ello una matriz de 3x3 como la que se muestra:

$$\begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 16 & 4 & 11 \\ 8 & 6 & 18 \\ 15 & 19 & 15 \end{pmatrix} X \begin{pmatrix} M_1 \\ M_2 \\ M_3 \end{pmatrix} \quad \text{mod 27}$$

Como las letras AMI del texto en claro tienen como código 0, 12 y 8, se cifrará:

$$C1 = (16*0 + 4*12 + 11*8) \bmod 27 \qquad 136 \bmod 27 = 1 = B$$
$$C2 = (8*0 + 6*12 + 18*8) \bmod 27 \qquad 216 \bmod 27 = 0 = A$$
$$C3 = (15*0 + 19*12 + 15*8) \bmod 27 \qquad 348 \bmod 27 = 24 = X$$

C = BAX (Continúa tú el resto del criptograma)

El espacio de claves que puede obtenerse en las matrices clave si el módulo n es un número primo y se cifra en bloques de d letras, se acerca a:

$$n^{d^2}$$

Por ejemplo, para matrices de 8x8, una cifra con bloques de ocho letras y n = 191 como cuerpo de cifra (un número primo y un subconjunto del código ASCII extendido con todos sus caracteres imprimibles), definido en el software CriptoClásicos V2.1, el espacio de claves será aproximadamente de $191^{64} \approx 2^{485}$, un valor inmenso. Por lo tanto, estos sistemas soportan muy bien un ataque en el que solo se conozca el texto cifrado. En resumen, resulta computacionalmente imposible intentar romper la cifra por fuerza bruta.

Sin embargo, serán esos sistemas matriciales altamente vulnerables ante ataques con texto en claro conocido, aplicando la técnica de Gauss-Jordan. Con muy poco texto en claro, normalmente inicios o finales del mismo, seremos capaces de encontrar la matriz clave y por lo tanto descifrar todo el criptograma. Básicamente, en este ataque se van buscando los vectores unitarios (la matriz de identidad) de forma que de ello pueda deducirse la clave.

Como moraleja final de los sistemas de cifra clásica podemos concluir que:

- Son sistemas muy sencillos, en algún caso hasta rudimentarios, en donde resulta fácil aplicar fuerza bruta en el ataque para algunos cifrados (por ejemplo la cifra del César).
- En algunos casos, las estadísticas y la redundancia del lenguaje nos permiten realizar ataques elegantes o criptoanálisis (por ejemplo la cifra afín, la cifra de Vigenère).
- En otros casos, son las matemáticas las que nos permiten criptoanalizar el sistema (por ejemplo el cifrado de Hill con texto en claro conocido).
- Son sistemas lineales y por ello no son seguros. Algo que no ocurrirá con la criptografía moderna.

En resumen: será muy fácil cifrar, será muy fácil descifrar y no nos resultará nada difícil aplicar métodos de criptoanálisis para romper la cifra de una forma elegante.

2.7. Características de los sistemas de cifra modernos

En los criptosistemas modernos, a diferencia de los sistemas clásicos, la cifra se realiza sobre bits o bytes y tanto la representación del texto en claro (sea este cualquier documento o archivo) como la del criptograma, se realiza sobre todos los caracteres del código ASCII, los 256 caracteres representados en bytes de 8 bits.

Como existe una cantidad de caracteres ASCII no imprimibles, es común representar los códigos de estos 256 caracteres por su valor en hexadecimal. De esta manera, los tres códigos que se muestran a continuación representan al mensaje en ASCII “Códigos para todos los gustos.”, el punto y final incluido.

Código ASCII binario de: Códigos para todos los gustos.
01000011 11110011 01100100 01101001 01100111 01101111 01110011 00100000 01110000 01100001
01110010 01100001 00100000 01110100 01101111 01100100 01101111 01110011 00100000 01101100
01101111 01110011 00100000 01100111 01110101 01110011 01110100 01101111 01110011 00101110

Código ASCII decimal de: Códigos para todos los gustos.
067 243 100 105 103 111 115 032 112 097 114 097 032 116 111 100 111 115 032 108 111 115 032 103 117 115 116 111 115 046

Código hexadecimal de: Códigos para todos los gustos.
43 F3 64 69 67 6F 73 20 70 61 72 61 20 74 6F 64 6F 73 20 6C 6F 73 20 67 75 73 74 6F 73 2E

Los algoritmos de cifra moderna usan por lo general una operación algebraica en Z_n, un cuerpo finito, sin que necesariamente este módulo deba corresponder con el número de elementos del alfabeto o código utilizado, como sí sucedía siempre en la cifra clásica. Es más, nunca coincidirán porque en criptografía moderna siempre será mucho mayor el cuerpo de trabajo que el alfabeto usado, en este caso ASCII extendido, octetos o bytes.

Siguiendo los principios de Kerckhoffs, su fortaleza debe basarse en la imposibilidad computacional de descubrir una clave secreta única, en tanto que el algoritmo de cifra es (o al menos debería serlo) público. Por imposibilidad computacional se entiende que matemáticamente sí es posible resolver el problema, pero que, debido a los desorbitados requerimientos de cómputo, de tiempo, y obviamente de dinero, hacen imposible abordar este problema, es decir intentar descubrir esa clave secreta.

Como ya ha sido comentado en el capítulo 1, la cifra moderna puede clasificarse de acuerdo a cómo se trata a la información antes de cifrarla, distinguiendo entre la cifra en flujo y la cifra en bloque. Existe otra división relacionada con el tipo de clave utilizada, diferenciando ahora entre la criptografía simétrica o de clave secreta y la criptografía asimétrica o de clave pública.

La Figura 2.7 muestra esta clasificación:

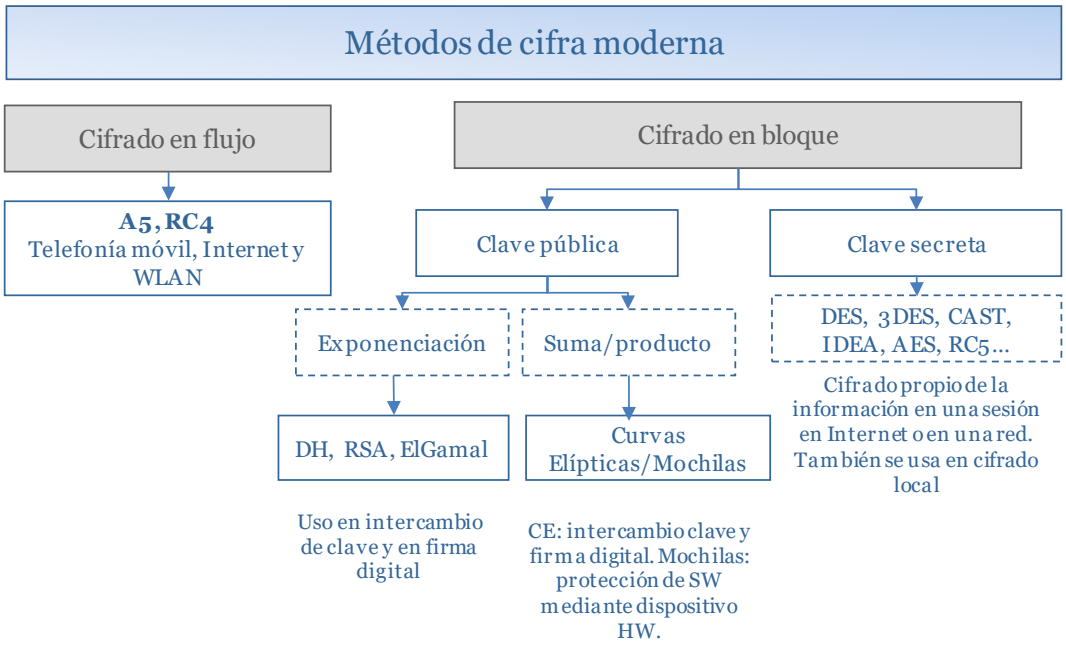


Figura 2.7. Clasificación de los sistemas de cifra moderna.

La cifra en flujo

La Figura 2.8 muestra el esquema de la cifra en flujo.

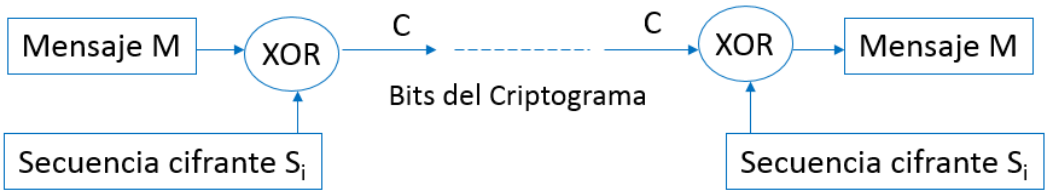


Figura 2.8. Esquema de la cifra en flujo.

Emisor y receptor intercambian una clave conocida como semilla (s = *seed*), del orden de 100 bits o mayor. A continuación y mediante un algoritmo determinista, por ejemplo un LFSR (*linear feedback shift register)* o registro de desplazamiento realimentado linealmente, generan la misma secuencia cifrante S_i, cuyos bits cifrarán or exclusivo a los bits del texto en claro, uno a uno.

Como la función XOR es involutiva, para descifrar el criptograma se hace la misma operación XOR, ahora entre los bits C_i del criptograma y los bits de la misma secuencia de clave S_i, recuperando el texto en claro M_i.

La cifra en bloque

Aquí el mensaje se agrupa en bloques, por lo general de 8 o 16 bytes (64 o 128 bits) antes de aplicar el algoritmo de cifra a cada bloque de forma independiente con la misma clave. No obstante, como veremos más adelante, este modo de cifra por bloques independientes no se usará al haber otros modos de cifra más seguros.

Si el bloque fuese muy pequeño, por ejemplo, de uno o de dos bytes, esto facilitaría un ataque por estadísticas del lenguaje. Se trataría de un cifrado por monogramas similar al del César o por digramas, muy débil al manifestarse la redundancia del lenguaje en el criptograma, y lógicamente no adecuado para los tiempos actuales. Y si el bloque fuese muy grande, por ejemplo, de cientos o miles de bytes, el sistema no tendría un buen rendimiento.

Los valores indicados de 64 y 128 bits son un término medio adecuado porque, además, permiten romper la estructura del lenguaje en el criptograma.

Modos de cifra en bloque

Si se cifran bloques independientes, el modo de cifra se conoce como ECB, (*electronic codebook)* o libro electrónico de códigos. Este modo está prohibido pues permite ataques por repetición de bloques conocidos y ataques por inicios y finales repetidos.

Aunque existen varios modos de cifra para los sistemas por bloques, los más conocidos son el modo CBC, (*cipher block chaining)* o encadenamiento de bloques cifrantes, y el modo CTR, (*counter)* o contador. Los esquemas correspondientes de estos modos y el análisis de cada uno de ellos se verán en el siguiente capítulo Algoritmos de cifra simétrica.

Características de la cifra simétrica o con clave secreta

Los sistemas de cifra simétrica tienen las siguientes características:

- No tienen una gestión de claves eficiente, el número de claves a recordar es n(n-1)/2.
- No tienen intercambio de clave ni firma digital.
- El tamaño de sus claves está entre 128 y 256 bits (año 2016).
- La duración de la clave de sesión (Internet) es corta (segundos, minutos).
- Tienen una tasa de cifra de centenas de MBytes por segundos (son rápidos).

Características de la cifra asimétrica o con clave pública

Los sistemas de cifra asimétrica tienen las siguientes características:

- Tienen una gestión de claves eficiente, el número de claves a recordar es 1.
- Tienen intercambio de clave y firma digital.
- El tamaño de sus claves está entre 2.048 y 4.096 bits (año 2018).
- La duración de la clave en un certificado digital es larga (1 o 2 años).
- Tienen una tasa de cifra de centenas de Kbytes por segundos (son lentos).

2.8. Cifra simétrica versus cifra asimétrica

¿Qué tipo de cifra es recomendable utilizar, simétrica o asimétrica, bloque o flujo? Vamos a contestar a estas preguntas.

Como se ha indicado en el apartado anterior, la cifra asimétrica (clave pública) tiene como fortalezas una eficiente gestión de claves, permite el intercambio de claves y posee firma digital, aspectos de los que adolece la cifra simétrica (clave secreta). Pero los sistemas de cifra asimétricos son unas mil veces más lentos que los simétricos.

Debido a la baja velocidad o tasa de cifra del sistema asimétrico, la criptografía asimétrica se utilizará solo para las operaciones de intercambio de clave y de firma digital, siempre con valores relativamente pequeños de entrada (la clave a intercambiar o bien la función hash del documento a firmar), y no para la cifra de los datos propiamente tal, que se hará con la simétrica.

Y esto es así ya que en el intercambio de clave sesión (una clave K de cifra simétrica), dicha clave tendrá un tamaño de unas centenas de bits, típicamente entre 128 y 256 bits, y que para la firma digital no se firmará el documento completo sino un hash del mismo, típicamente entre 160 y 256 bits. Así, aunque la tasa de los sistemas asimétricos sea tan solo de centenas de Kbytes por segundo, como lo que se cifra son pocas centenas de bits, la velocidad y eficiencia del sistema asimétrico no se resienten.

Por lo tanto, en este entorno de cifra híbrida la criptografía asimétrica se usa para el intercambio de clave de sesión y la autenticación mediante firma digital, y la criptografía simétrica se usa para el cifrado propiamente dicho de la información.

En cuanto a si es mejor usar cifrado en flujo o en bloque, la decisión no es tan drástica, si bien los algoritmos de cifrado en flujo tienen una velocidad mayor que los de cifra en bloque, como mínimo más del doble. Sin embargo, hoy en día prevalece el cifrado en bloque AES frente al cifrado en flujo RC4, que desde 2015 ha dejado de usarse en protocolos como TLS debido a una serie de vulnerabilidades detectadas.

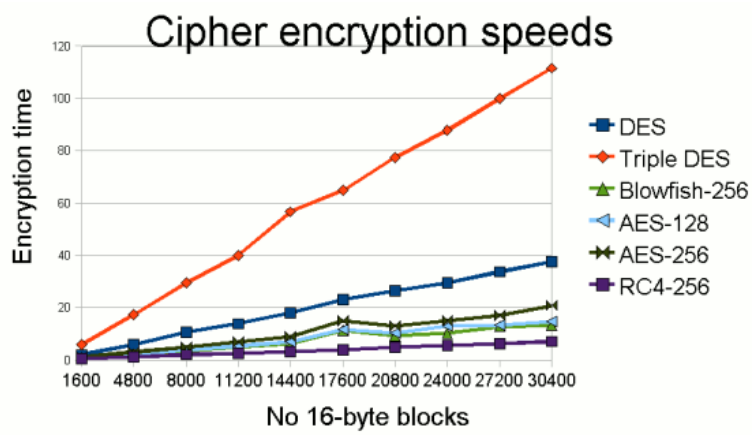


Figura 2.9. Tasa de cifra de algoritmos conocidos. Fuente: http://www.javamex.com/tutorials/cryptography/ciphers.shtml

La Figura 2.9 muestra la tasa de cifra de varios algoritmos conocido: AES, DES, 3DES, RC4 y Blowfish. Observa que RC4 era aproximadamente 4 veces más rápido que AES y que este último es el doble de rápido que el DES. Obviamente 3DES es el que peor tasa de cifra tiene, dado que se trata de tres cifrados sucesivos del mismo algoritmo DES, un modo de cifrado múltiple que veremos en el capítulo siguiente.

2.9. Usos de la criptografía moderna

Los algoritmos modernos que estudiaremos en los siguientes capítulos aparecerán, por ejemplo, en un certificado digital X.509 y en el protocolo SSL/TLS de sesión segura en Internet.

En la figura 2.20 se observan los algoritmos a los que han llegado a un acuerdo el cliente (navegador) y el servidor (Google) en un *handshake* de sesión TLS, en donde ECDHE significa *Elliptic Curve Diffie-Hellman key Exchange*, ECDSA *Elliptic Curve Digital Signature Algorithm* y GCM *Galois/Counter Mode*, un modo de cifra tipo contador.

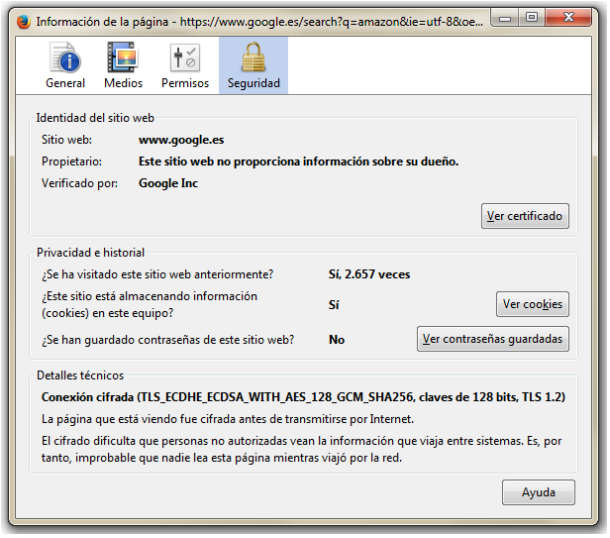


Figura 2.20. Algoritmos usados en una conexión segura TLS con el servidor de Google.

La figura 2.21 muestra zonas de un certificado digital X.509 (en este caso de Amazon) en donde puede verse el uso del algoritmo RSA. Si buscamos más datos dentro de ese certificado digital, encontraremos por ejemplo que el algoritmo de firma del certificado es PKCS #1 SHA-256 con cifrado RSA.

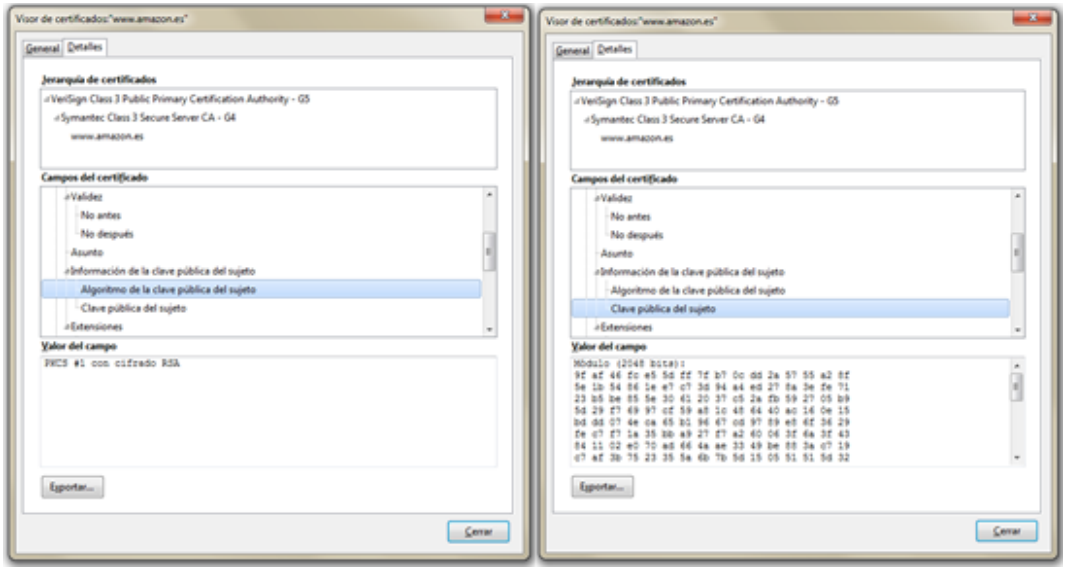


Figura 2.21. Algoritmos usados en el certificado digital X.509 de Amazon.

Test del capítulo 2

1. El principio de Kerckhoffs más importante dice que:

A. La seguridad del sistema de cifra debe recaer solo en el algoritmo

B. La seguridad no es un producto sino un proceso

C. La seguridad del sistema de cifra debe recaer solo en la clave

D. Los sistemas de cifra nunca pueden ser seguros al 100 %

2. El algoritmo de Vigenère corresponde a una cifra del tipo:

A. Sustitución monoalfabética monográmica

B. Sustitución monoalfabética poligrámica

C. Sustitución polialfabética monográmica

D. Sustitución polialfabética poligrámica

3. El algoritmo de Hill corresponde a una cifra del tipo:

A. Sustitución monoalfabética monográmica

B. Sustitución monoalfabética poligrámica

C. Sustitución polialfabética monográmica

D. Sustitución polialfabética poligrámica

4. Vamos a cifrar un texto de 19 letras mediante una permutación de 6 columnas:

A. No aplicaremos ningún relleno

B. Aplicaremos seis rellenos con una letra poco frecuente

C. Aplicaremos cuatro rellenos con una letra poco frecuente

D. Aplicaremos cinco rellenos con una letra poco frecuente

5. Si la cifra del César con b distinto de 3 es XVIW, el texto en claro será:

A. TRES

B. ASNO

C. SEIS

D. NOTA

6. Si se cifra el texto ABAD con un método de sustitución monoalfabética definido por c = 2*m+3 mod 27, el criptograma será:

A. DFDK

B. DJDE

C. DFDJ

D. DFEK

7. En un criptoanálisis de cifra afín se llega a estas dos ecuaciones: {1 = c*0 + b mod 27] y [2 = a*4 + b mod 27]. ¿Cuáles son los valores de a y de b?

A. a = 7, b = 1

B. a = 4, b = 1

C. a = 7, b = - 1

D. a = - 4, b = 1

8. Se cifra con Vigenère el texto CIELO con la clave AZUL. ¿Cuál es el criptograma? Se sabe que: C=2, I=8, E=4, L=11, O=15, A=0, Z=26, U=21.

A. CHVHP

B. CHYVO

C. CAYHO

D. CHYYQ

9. Un ataque de Kasiski a Vigenère nos dice que la longitud de la clave es 7 y que hay 3 subcriptogramas con una distribución de frecuencias muy similar a la del texto en claro.

A. La clave puede ser FRUTERO

B. La clave puede ser SANDIAS

C. La clave puede ser PLATANO

D. La clave puede ser NARANJA

10. Para cifrar con el método de Hill bloques de trigramas, deberemos usar:

A. Una matriz de 2x3

B. Una matriz de 3x2

C. Una matriz de 3x3

D. Una matriz de 2x2

Tema 3

Algoritmos de cifra simétrica

3.1. Introducción

3.2. Generalidades de la cifra en flujo

3.3. Algoritmos de cifra en flujo: A5, RC4

3.4. Generalidades de la cifra en bloque

3.5. Algoritmos de cifra en bloque: DES, 3DES, IDEA, AES

3.1. Introducción

Para profundizar en el tema, lee el capítulo 10 Introducción a la Cifra Moderna del siguiente libro.
Ramió, J. (2006). Libro electrónico de seguridad informática y criptografía, versión 4.1.
Disponible en: http://www.criptored.upm.es/guiateoria/gt_m001a.htm

Además, se recomienda la visualización de las siguientes píldoras formativas Thoth del mismo autor:

Píldora 28: ¿Cómo funcionan los algoritmos DES y 3DES?
Píldora 29: ¿Por qué sucumbe el DES ante un ataque en red?
Píldora 30: ¿Cómo se cifra con el algoritmo AES?
Píldora 31: ¿Qué son los rellenos y los modos de cifra en bloque?
Píldora 32: ¿Qué son los postulados de Golomb?
Píldora 33: ¿Cómo se usan los registros de desplazamiento en la cifra?
Píldora 34: ¿Cómo ciframos en flujo con A5, RC4 y en modo CTR?
Píldora 35: ¿Cómo funciona el algoritmo RC4?
Disponibles en:
https://www.youtube.com/watch?v=7MqTpfEreJ0&list=PL8bSwVy8_IcNNS5QDLjV7gUg8dIeMFSER

En este capítulo profundizaremos en la criptografía simétrica o de clave única y secreta, diferenciando entre cifra en flujo y cifra en bloque. Entre los algoritmos a estudiar, se encuentran A5 y RC4 de cifra en flujo y DES, 3DES, IDEA y AES de cifra en bloque.

En la cifra en flujo el texto en claro se cifrará bit a bit, o byte a byte, con una secuencia de bits o bytes que forman la clave, conocida como secuencia de clave S_i. No se trata de una clave cuyo valor sea único y que se use durante todo el proceso de cifra, como se sucederá en la cifra en bloques, sino que esta va cambiando a medida que se recorre el texto en claro. Por este mismo motivo, es recomendable que, entre otras cosas, dicha secuencia de clave tenga al menos tantos bits como el mensaje que se va a cifrar, para que la clave no sea periódica y, por tanto, no dé pistas al criptoanalista.

Por su parte, en la cifra en bloque se usará una única clave, que comparten emisor y receptor y que no cambiará durante el proceso de cifra. El emisor que cifra el mensaje formará bloques de tamaños típicos entre 64 y 128 bits, es decir ocho y dieciséis octetos o bytes, siendo este último tamaño el más común hoy en día. Cada bloque se cifrará durante varias vueltas (esto depende de cada algoritmo) con claves diferentes para cada una de ellas, que el mismo algoritmo genera, lo que obviamente se repetirá de forma inversa en el

descifrado. Además, en este caso existirán diferentes modos de cifra como ya se ha comentado en la lección anterior.

3.2. Generalidades de la cifra en flujo

La figura muestra el esquema de un sistema de cifra en flujo.

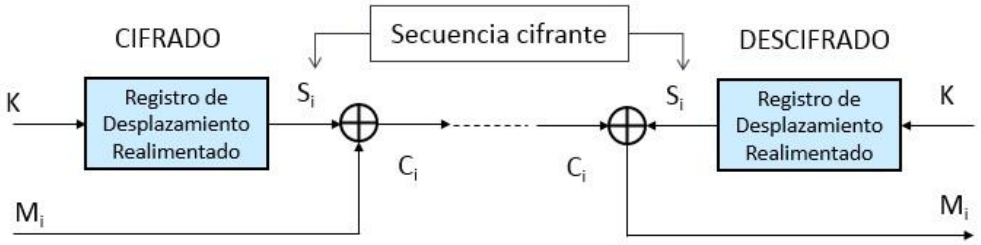


Figura 3.1. Esquema de la cifra en flujo.

$$C_i = M_i \oplus S_i$$

$$M_i = C_i \oplus S_i$$

Siguiendo el esquema de la figura anterior, estos sistemas de cifra se resumen en las siguientes tres características:

- Un algoritmo de cifrado y descifrado basado en la función xor.
- Una secuencia cifrante binaria y pseudoaleatoria S_i que se obtiene a partir de una clave secreta K, compartida de forma segura entre emisor y receptor, y un algoritmo generador de esa secuencia S_i con registros de desplazamiento.
- Una misma operación de cifrado y descifrado debido al carácter involutivo de la función xor.

Toda la fortaleza de un sistema de cifra en flujo residirá en las características de esa secuencia de clave S_i, en tanto el algoritmo de cifra es muy básico, una simple función XOR. No se utilizan en este caso técnicas de difusión (permutación) ni de confusión (sustitución), que sin embargo sí son habituales y necesarias en sistemas de cifra simétrica en bloques. Una de las condiciones más importantes que deben cumplir estas secuencias para que puedan ser consideradas como pseudoaleatorias, son las que tienen que ver con los postulados de Golomb que se indican a continuación.

Primer postulado de Golomb G1

Los bits 0 y 1 de la secuencia cifrante deberán ser equiprobables. Por lo tanto, la secuencia deberá tener mitad de unos y mitad de ceros, aunque lo normal es que tengan un uno más que los ceros puesto que las secuencias que se utilizan en la práctica serán impares, al provenir de un sistema lineal como veremos más adelante. Esto quiere decir que la probabilidad de encontrarse con un cero o con un uno en cualquier posición de esa secuencia, deberá ser del 50 %.

Por ejemplo, cumple con este primer postulado de Golomb la secuencia S_1 con 16 unos y 15 ceros, pero no lo cumple la secuencia S_2 con 7 unos y 8 ceros.

S_1 = 00111110001101110101000010010 11

S_2 = 110101011001000

Segundo postulado de Golomb G2

En este caso, la equiprobabilidad que los bits unos y ceros mostraban en G1 debería seguir manifestándose, independientemente de los bits que hayamos leído o conocido con anterioridad en dicha secuencia. Esto es, si hemos recibido la cadena 1000, la probabilidad de que el quinto bit de esa cadena sea un 0, es decir 10000, o bien un 1, es decir 10001, debería seguir siendo del 50 %. Por ejemplo, la secuencia anterior S_1 cumple con este segundo postulado de Golomb como se indica más abajo, subrayando los bits en cuestión.

S_1 = 00111<u>10001</u>10111010<u>10000</u>1001011

Lo importante de este segundo postulado de Golomb es que, si se cumple, significa que la secuencia S_i pasará por todos sus estados (restos) posibles y, por lo tanto, la secuencia será máxima, o de periodo máximo, que es una de las características que se buscan para estas secuencias de clave.

En el caso de la secuencia anterior S_1, se trata de un registro con 5 celdas, que entrega un periodo igual a $2^5 - 1 = 31$ y que, excepto el resto de todos ceros (00000) que veremos está prohibido porque el sistema es lineal, pasará por todos los demás restos, desde 00001, 00010, 00011, 00100, 00101, etc. hasta 11111.

Para que esto suceda, deberá cumplirse que las rachas de ceros y de unos, esto es un conjunto de ceros entre dos unos, o bien un conjunto de unos entre dos ceros, siga una distribución geométrica. Por ejemplo, una secuencia máxima de 63 bits ($2^6 - 1$) con un registro lineal de 6 celdas, deberá tener:

- 16 rachas de longitud uno: ocho del tipo 010 y ocho del tipo 101
- 8 rachas de longitud dos: cuatro del tipo 0110 y cuatro del tipo 1001

- 4 rachas de longitud tres: dos del tipo 01110 y dos del tipo 10001
- 2 rachas de longitud cuatro: una del tipo 011110 y una del tipo 100001
- 1 única racha de longitud cinco (n-1) solo de ceros: del tipo 1000001
- 1 única racha de longitud seis (n) solo de unos: del tipo 01111110

Observa que las rachas se van dividiendo por dos desde 2^{n-2}, en este caso $2^{6-2} = 2^4 = 16$, y al final se obtiene una única racha de n-1 ceros (ninguna de unos) y una única racha de n unos (ninguna de ceros). Estas secuencias se conocerán como m-secuencias.

Tercer postulado de Golomb G3

El tercer postulado de Golomb dice que la autocorrelación fuera de fase AC(k) entre la secuencia original y dicha secuencia desplazada k bits deberá ser contante. Se define dicho valor AC(k) como la diferencia entre los Aciertos y los Fallos al comparar bit a bits la secuencia original con la secuencia desplazada.

$$\mathrm{AC(k)} = (\mathrm{Aciertos} - \mathrm{Fallos})/\mathrm{T}$$

Siendo Aciertos el número de bits que coinciden en igual posición entre las dos cadenas y Fallos el número de bits que no coinciden, con T el periodo de la secuencia.

Si esto se cumple, significa que la probabilidad de éxito para atacar a la secuencia cifrante analizando un conjunto de bits en una parte de ella o bien en otra, será la misma. Es decir, no habrá ventajas al estudiar la secuencia en una zona u otra.

Registros de desplazamiento

Un registro de desplazamiento es un circuito digital secuencial que está compuesto por n celdas que almacenan un 1 o un 0, y que al ritmo que marca un reloj, los bits se van desplazando a la celda contigua y por tanto entregando un bit de salida. Ese bit de salida, que se pierde y que en nuestro caso se usa como parte de la secuencia S_i de clave para cifrar, se realimenta hacia el sistema mediante una función que puede ser lineal o no lineal y en la que intervienen, además, los valores que contienen en ese momento una o más celdas conectadas al sistema de realimentación.

La figura 3.2 muestra un registro lineal (solo una función XOR como realimentación) conocido como LFSR (Linear Feedback Shift Register) de 4 celdas, donde están conectadas a ese or exclusivo la celda 4 (al ser la última, lo estará siempre), la celda 3 y la celda 1, no así la celda 2.

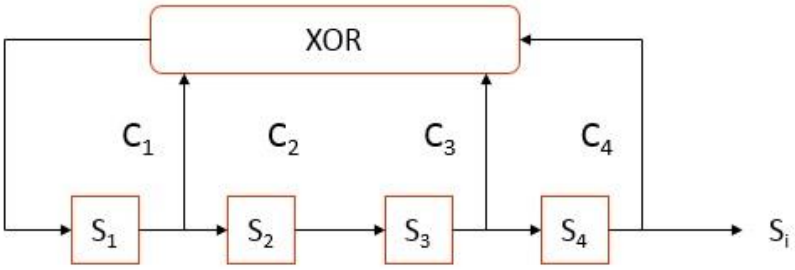
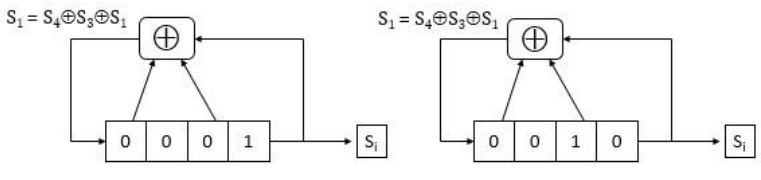


Figura 3.2. Registro de desplazamiento realimentado linealmente LFR.

La disposición de conexión de celdas al XOR se asocia a un polinomio. En el caso, el polinomio $x^4 + x^3 + x + 1$. Este polinomio indicará si el registro es de los que entregan un periodo máximo, que es lo que nos interesa, y si cumple, además, con los tres postulados de Golomb. Esto último nos confirmará que S_i es una buena secuencia pseudoaleatoria. Como los bits de la secuencia S_i del registro de la figura 3.2 salen hacia la derecha, los bits de dicha secuencia serán $S_4S_3S_2S_1S_5S_6$… hasta completar el periodo. Es decir, primero se transmite la semilla $S_1S_2S_3S_4$ al revés ($S_4S_3S_2S_1$) y, a continuación, los bits siguientes según evoluciona el registro.

Existirán tres tipos de polinomios, los factorizables o que pueden representarse por dos o más polinomios de menor grado, los irreducibles que no pueden factorizarse pero que no generan el conjunto completo de restos, y los primitivos que, además de ser irreducibles, sí generan el conjunto completo de restos.

El polinomio $x^4 + x^3 + x + 1$ es de tipo factorizable y estos no son recomendables puesto que no generan una secuencia máxima. Además, el periodo de S_i va a depender de la semilla, algo inaceptable. Como se observa en la figura 3.3, para la semilla $S_1S_2S_3S_4 = 0001$ se obtiene la secuencia cifrante $S_i = 100011$ con periodo 6, y para la semilla $S_1S_2S_3S_4 = 0010$ se obtiene la secuencia cifrante $S_i = 010$ con periodo 3.



t	Registro	S_i
1	0001	1
2	1000	0
3	1100	0
4	1110	0
5	0111	1
6	0011	1
7	0001	

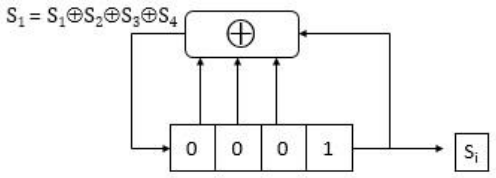
Si = 100011

t	Registro	S_i
1	0010	0
2	101	1
3	0100	0
4	0010	

Si = 010

Figura 3.3. LFSR factorizable de 4 celdas con semillas 0001 y 0010.

La figura 3.4 muestra un LFSR irreducible con polinomio $x^4 + x^3 + x^2 + x + 1$ y la misma semilla 0001. Aquí el periodo ya no será variable en función de la semilla, pero no se obtendrá el periodo máximo sino un divisor del mismo, en este caso S_i = 10001 de longitud 5.

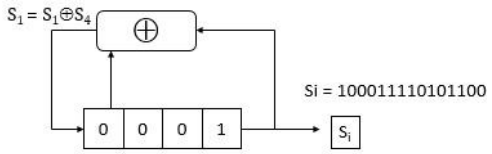


Si = 10001

t	Registro	S_i
1	0001	1
2	1000	0
3	1100	0
4	0110	0
5	0011	1
6	0001	

Figura 3.4. LFSR irreducible de 4 celdas con semilla 0001.

Por último, la figura 3.5 muestra un LFSR de polinomio primitivo $x^4 + x + 1$ y la misma semilla 0001. Estos son los polinomios recomendables puesto que generan secuencias máximas, con periodo igual a $2^n - 1$, cumpliendo además con los tres postulados de Golomb. Para este caso, S_i = 100011110101100, con periodo $15 = 2^4 - 1$.



Si = 100011110101100

t	Registro	S_i	t	Registro	S_i
1	0001	1	9	1010	0
2	1000	0	10	1101	1
3	1100	0	11	0110	0
4	1110	0	12	0011	1
5	1111	1	13	1001	1
6	0111	1	14	0100	0
7	1011	1	15	0010	0
8	0101	1	16	0001	

Figura 3.5. LFSR primitivo de 4 celdas con semilla 001.

Ataques a las m-secuencias

Con LFSRs primitivos se consiguen m-secuencias, esto es, secuencias con un periodo máximo $T = 2^n - 1$.

Observa que, si n es igual a 100 bits, entonces 2^{100} = 1.267.650.600.228.229.401.496.703.205.376 bits y el periodo de la secuencia de clave será inmenso, más de 150 mil billones de Terabytes. Comprueba que no

existe en el mundo sistema alguno que almacene o procese esa cantidad de información. Por ejemplo, ¿te has preguntado cuánta información almacena y/o procesa Google?

No obstante, el hecho de que la secuencia cifrante S_i sea máxima, significa que la misma pasa por todos sus estados o restos. Y esto hace que dicha secuencia sea predecible y, por lo tanto, no sea válida para la cifra. Ello se debe a que esta predecibilidad permite un ataque conocido como Berlekamp-Massey, mediante el cual es posible romper una secuencia de 2^n bits conociendo tan solo 2n bits consecutivos de dicha cadena.

Es decir, una secuencia de 2^{25} = 33.554.432 bits, más de 33 millones de bits, podría romperse conociendo tan solo 2*25 = 50 bits consecutivos en cualquier lugar de esa cadena. Por tal motivo, nunca se usará un único LFSR para obtener la secuencia cifrante, sino dos o más de ellos, conectados mediante una función lógica, preferentemente una puerta XOR, coo se muestra en la figura 3.6, en donde LC (Linear Complexity) es la complejidad lineal del sistema.

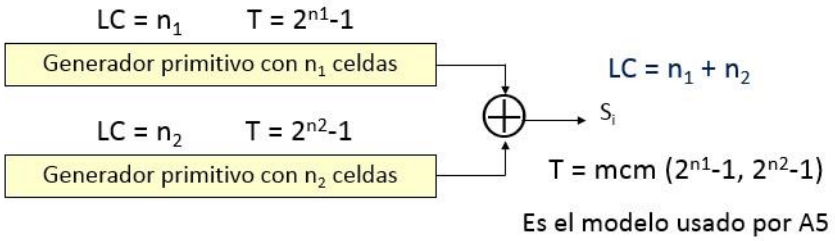


Figura 3.6. Dos registros LFSR conectados a través de una puerta XOR.

El generador de la figura 3.6 no será predecible como los anteriores. Tendrá un periodo igual al mínimo común múltiplo de los periodos de cada LFSR y en algunos casos cumplirá solo con el primer postulado de Golomb G1. No obstante, sí será un buen generador de secuencia cifrante al tener una mayor complejidad lineal.

Observa que nos interesa que cada LFSR cumpla con G1, G2 y G3, básicamente para que su periodo sea el máximo posible, pero ya no es necesario que la secuencia cifrante final S_i cumpla con todos los postulados de Golomb. Aquí es mucho más importante que la secuencia de clave, con la cual vamos a cifrar la información, tenga una alta complejidad lineal o, lo que es lo mismo, no sea fácil poder determinar la secuencia completa a partir del conocimiento de un trozo de ella.

3.3. Algoritmos de cifra en flujo: A5, RC4

La figura 3.7 muestra el esquema del algoritmo A5.

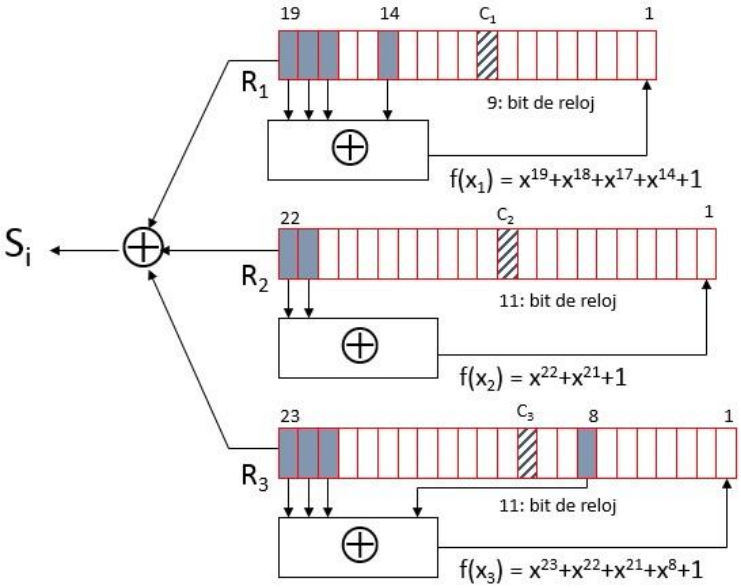


Figura 3.7. Algoritmo de cifra A5.

Este algoritmo fue creado en 1987 para GSM Global System for Mobile Communications. El uso habitual de lo encontramos en el cifrado del enlace entre el abonado y la central de un teléfono móvil tipo GSM. Existen dos versiones, A5/1 y A5/2, este último mucho más débil y diseñado a propósito con esas características.

Aunque el código fuente se oculta (un grave error), en 1999 se demuestra que el diseño del algoritmo es muy débil, especialmente por su corta longitud de clave o semilla. El sistema A5/1 contaba en 1999 con cerca de 130 millones de usuarios en Europa y otros 100 millones de usuarios en el resto del mundo, y sucumbe por primera vez en diciembre de ese año ante un ataque con texto en claro conocido, realizado por Alex Biryukov, Adi Shamir y David Wagner. Ante estos ataques, se sustituye A5/1 por A5/3, algoritmo llamado Kasumi y que en realidad es un cifrador de bloque y no de flujo. Actualmente, en telefonía móvil 3G y 4G se usa el algoritmo Snow 3G.

El algoritmo A5 consta de 3 registros LFSR primitivos como los que hemos visto en el apartado anterior de 19, 22 y 23 celdas. Por lo tanto, la semilla del sistema es igual a la suma de estas cantidades, 64 bits, un valor muy bajo y desaconsejable para la fecha de su creación. Para que la salida de bits en el XOR de los tres registros sea menos predecible, se usa una función denominada Mayoría y que consiste en lo siguiente: el bit de la celda 9 del registro R1 y los bits de las celdas 11 de los registros R2 y R3 pueden pasar durante la generación de la secuencia por estos ocho estados posibles: 000, 001, 010, 011, 100, 101, 110, 111. Así, en función del valor que tenga el bit en cada una de esas tres celdas, los correspondientes registros desplazarán o no desplazarán de acuerdo a la función Mayoría que se muestra en la siguiente figura.

C_1	C_2	C_3		C_1	C_2	C_3	
0	0	0	F = 0: Desplazan todos	1	0	0	F = 0: No desplaza R_1
0	0	1	F = 0: No desplaza R_3	1	0	1	F = 1: No desplaza R_2
0	1	0	F = 0: No desplaza R_2	1	1	0	F = 1: No desplaza R_3
0	1	1	F = 1: No desplaza R_1	1	1	1	F = 1: Desplazan todos

Figura 3.8. Función Mayoría en A5.

Por tanto, los registros que tienen su bit dentro de la mayoría de ceros o de unos (indicado como F = 0 y F = 1) desplazarán y, en cambio, el registro que tenga su bit fuera de esa mayoría, no desplazará. Como se observa, siempre desplazarán al menos dos registros, y en dos casos desplazarán todos los registros.

En cuanto a RC4, acrónimo de Ron's Code 4, se trata de un algoritmo de cifra en flujo byte a byte diseñado por Ron Rivest en 1987, cuyo esquema se muestra en la figura 3.9. Su uso más habitual se encontraba hasta el año 2014 en el protocolo SSL/TLS, especialmente al utilizar Internet Explorer, aunque se usa también en WEP Wired Equivalent Privacy y en WPA Wi-Fi Protected Access, para el cifrado en redes inalámbricas. Como se observa en la figura 3.10, el algoritmo consta dos rutinas: KSA con la que se genera la clave y PRGA con la que se cifra cada byte del texto en claro con un nuevo byte de la secuencia de clave.

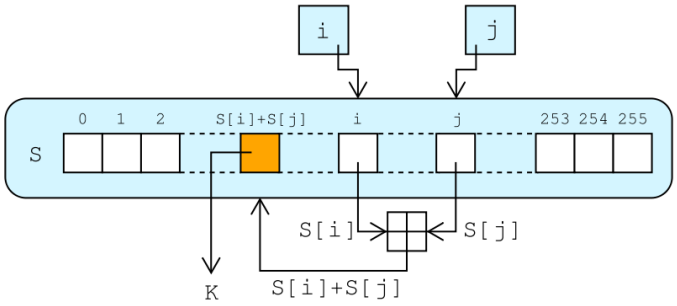


Figura 3.9. Esquema del algoritmo RC4. Fuente: Wikipedia.

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

KSA
Key Scheduling Algorithm
Inicialización del array S
1 < keylength < 256 bytes
5 ≤ Típico ≤ 16 bytes (40 - 128 bits)

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

PRGA
Pseudo-Random Generation Algorithm
Mientras sea necesario, modifica el estado y entrega el byte de clave K.
El byte de salida K se obtiene sumando S[i] + S[j] mod 256

Figura 3.10. Rutinas KSA y PRGA de RC4.

Su popularidad estaba basada fundamentalmente en su alta velocidad y en la simplicidad de su diseño, tanto en hardware como en software, siendo casi diez veces más rápido que el algoritmo DES, el estándar mundial de cifra simétrica hasta finales de la década de los noventa, y el doble de rápido que AES, el actual estándar de cifra simétrica.

El código del algoritmo RC4 era secreto hasta el año 1994, en el que aparece una descripción de su funcionamiento en un post anónimo enviado a la lista de correo de Cipherpunks. Como es una marca registrada, las implementaciones no oficiales de RC4 se conocen con otros nombres como ARC4, ARCFOUR o Alleged-RC4. En febrero de 2015, la organización internacional IETF Internet Engineering Task Force recomienda la eliminación del cifrado RC4 en las conexiones TLS entre clientes y servidores, debido a una serie de vulnerabilidades críticas descubiertas, por lo que actualmente ha quedado prácticamente en desuso, en beneficio del AES.

3.4. Generalidades de la cifra en bloque

En la cifra simétrica en bloque la información a cifrar o texto en claro se agrupa en bloques n de bits, típicamente 128 bits o bien 16 octetos. Estos bloques de bits entran al cifrador y se mezclan con los bits la clave mediante diversas operaciones tanto de permutación como de sustitución. Por este motivo, este tipo de cifradores son conocidos también como cifradores de producto.

Cada bloque de texto en claro se convierte en un bloque de texto cifrado, como se muestra en a figura 3.11. Al final del texto en claro se ha añadido un relleno por si éste no es congruente con el tamaño del bloque.

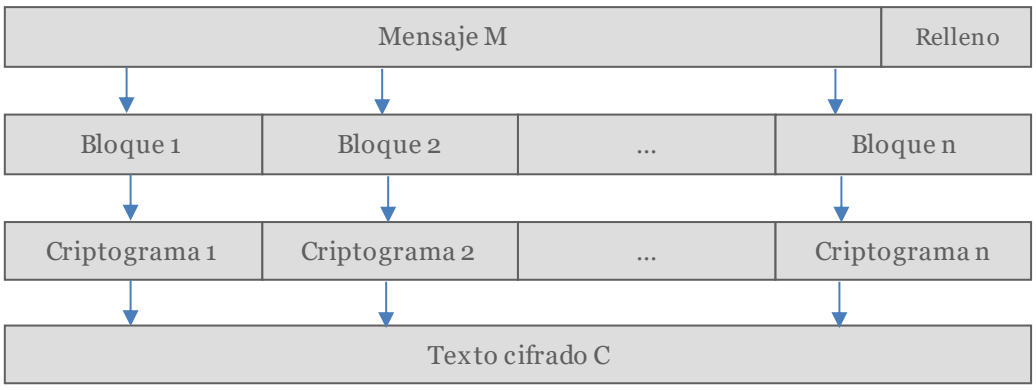


Figura 3.11. Bloques en la cifra simétrica.

Para que los bits del texto en claro se mezclen lo suficiente con los bits de la clave, habrá que aplicar permutaciones y sustituciones en esos bits varias veces. En criptografía a estas operaciones repetitivas se les llama vueltas o rondas del algoritmo. Con ello se logra, entre otras características interesantes, el denominado efecto de avalancha, esto es, que el cambio de un solo bit en la entrada (o en la clave) produzca un cambio en

aproximadamente el 50% de los bits de salida. En otras palabras, que cada bit de salida sea una función compleja que dependa en un 50 % de cada bit de entrada.

De la misma forma que los bloques M_1, M_2, etc. del texto en claro corresponden a información en claro cuyos bits están concatenados formando el mensaje M, el criptograma final C será el resultado de la concatenación de los criptogramas C_1, C_2, etc., también llamados subcriptogramas.

En la figura 3.12 se muestra el esquema de una cifra simétrica en bloques y las operaciones características que se realizan en cada fase del algoritmo. Cada bloque se cifra durante varias vueltas en las que se usa una subclave, que se calcula a partir de la clave de cifra K con una función que se conoce como expansión de clave, de forma que en cada vuelta ésta sea distinta. Se muestra, además, la analogía entre este proceso de cifra con varias vueltas y la elaboración de una mayonesa, en donde aceite y huevo hacen las funciones de texto y clave.

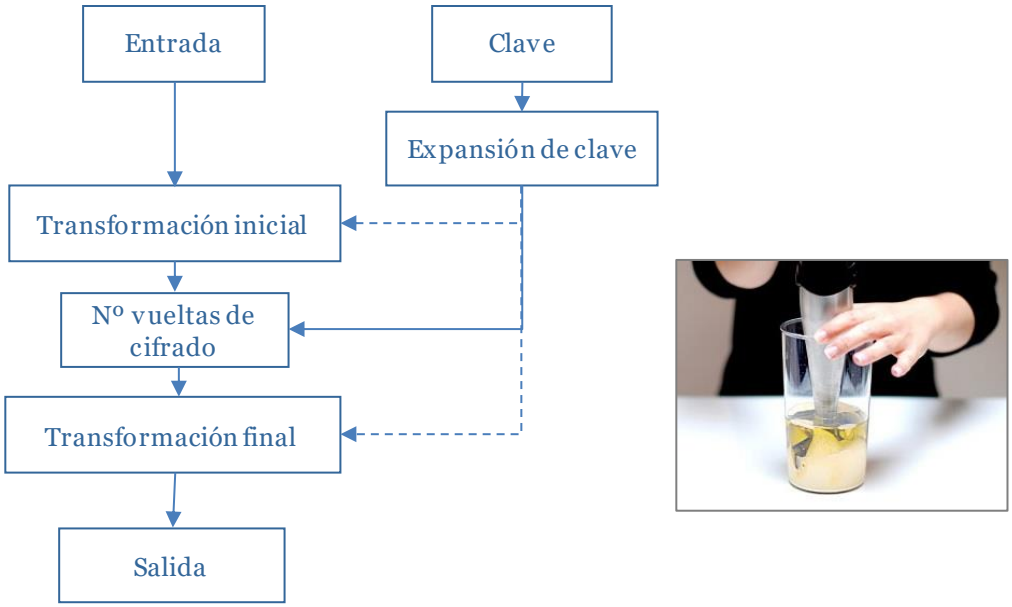


Figura 3.12. Esquema de la cifra simétrica.

Modos de cifra en bloque

Aunque existen una media docena modos de cifra, los más conocidos son:

- Modo de cifra Libro Electrónico de Códigos ECB: Electronic CodeBook
- Modo de cifra por Encadenamiento de Bloques Cifrados CBC: Cipher Block Chaining
- Modo de cifra Contador CTR: Counter

El primero de ellos, ECB, es el modo de cifra por defecto, es decir exactamente como viene escrito el código básico del algoritmo, pero que no debe usarse pues presenta varias vulnerabilidades. El segundo modo CBC, ha

sido el más popular de todos y se sigue usando de manera frecuente en las comunicaciones seguras en Internet en conexiones SSL/TLS. Por último, el tercero de ellos CTR, presenta interesantes ventajas con respecto al modo CBC, es más eficiente y se va abriendo camino como el modo de cifra idóneo y actualmente más recomendado, incluyéndose una versión más moderna con Campos de Galois.

Modo ECB

El modo de cifra por libro electrónico de códigos ECB (Electronic Codebook) consiste en cifrar bloques de texto en claro de manera independiente. Así, cada bloque de x bits del texto en claro se cifra con la clave K del algoritmo y entrega un bloque de texto cifrado de x bits. No se recomienda el uso de este modo porque es vulnerable a ataques por inicios y finales iguales entre dos documentos diferentes, así como a ataques por repetición de bloques elegidos o bien modificados.

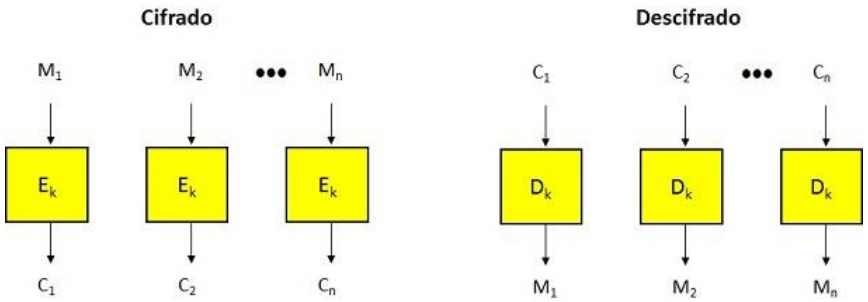


Figura 3.13. Modo de cifra ECB Electronic Codebook.

No obstante, el modo de cifra ECB tiene como aspecto positivo que, al cifrarse bloques de texto en claro de manera independiente, cualquier error de transmisión afectará solamente al bloque en cuestión y no a todo el criptograma.

Modo CBC

En el modo de cifra por encadenamiento de bloques cifrantes (Cipher Block Chaining) CBC, se usa una segunda clave de igual tamaño que el bloque de texto a cifrar, llamada vector inicial IV, que se recomienda sea también secreta, aunque ello no sea obligatorio, y con la cual se realiza un OR exclusivo con el primer bloque del texto en claro a cifrar, antes de comenzar la cifra.

El resultado de la cifra de esta nueva entrada con la clave K, es decir C_1, se usará como vector inicial del siguiente bloque de texto a cifrar, y así sucesivamente. Por tanto, se va encadenando el cifrado anterior con el nuevo cifrado, usándolo como vector de suma OR exclusivo para el nuevo bloque de texto a cifrar.

Para descifrar con este método, una vez se ha aplicado el algoritmo de descifrado al primer subcriptograma con la clave K, se recupera el texto en claro realizando una suma OR exclusivo con ese vector inicial IV. Los siguientes bloques de sucriptogramas se descifran de igual manera, pero utilizando ahora como vector inicial el criptograma anterior, que hemos guardado previamente en un registro temporal.

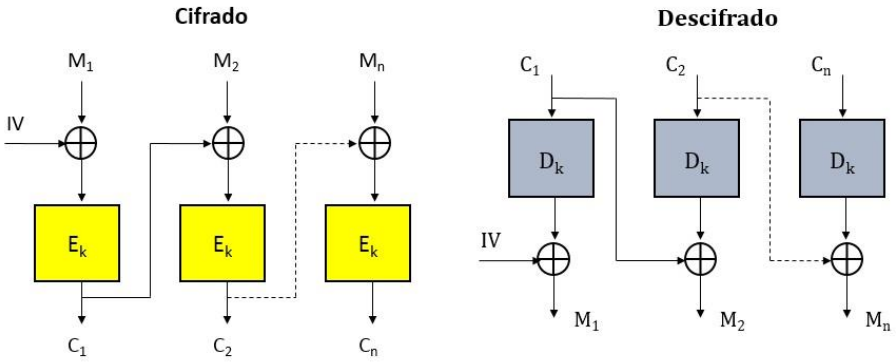


Figura 3.14. Modo de cifra CBC Cipher Block Chaining.

CBC es inmune a los ataques comentados para ECB, pero ahora un error en la transmisión afectará a todo el resto del criptograma. Unido a esto, aparece el problema asociado de que CBC no permite paralelizar la operación de descifrado, algo a tener en cuenta cuando hablamos de cifrado de grandes volúmenes de información.

Modo CTR

El modo CTR es similar al modo CBC, pero soluciona el problema del no paralelismo que tiene CBC ya que al ir encadenada la cifra en CBC, era obligatorio tener descifrado el bloque anterior para descifrar el próximo. Además, a diferencia del modo CBC, en CTR un error de transmisión afectará solo al bloque en cuestión, no a los siguientes.

CTR usa ese mismo vector inicial IV, pero en modo contador y no sumándolo XOR al texto en claro, sino como entrada al propio algoritmo de cifra, de forma que para el primer bloque su valor es IV, para el segundo bloque es IV+1, etc., lo que permite solucionar los problemas comentados del modo CBC. La salida de esa cifra se suma or exclusivo con el bloque de texto en claro a cifrar.

Observa que al cambiar la entrada al algoritmo de cifra por este vector IV, IV+1, IV+2, etc. en vez del texto en claro, el algoritmo de cifra de bloque se convierte en un sistema de cifra en flujo porque la clave con la cual se hace el XOR con los bloques texto en claro es distinta en cada bloque. Se trata entonces de una secuencia de clave S_i como en anteriores sistemas de cifra en flujo.

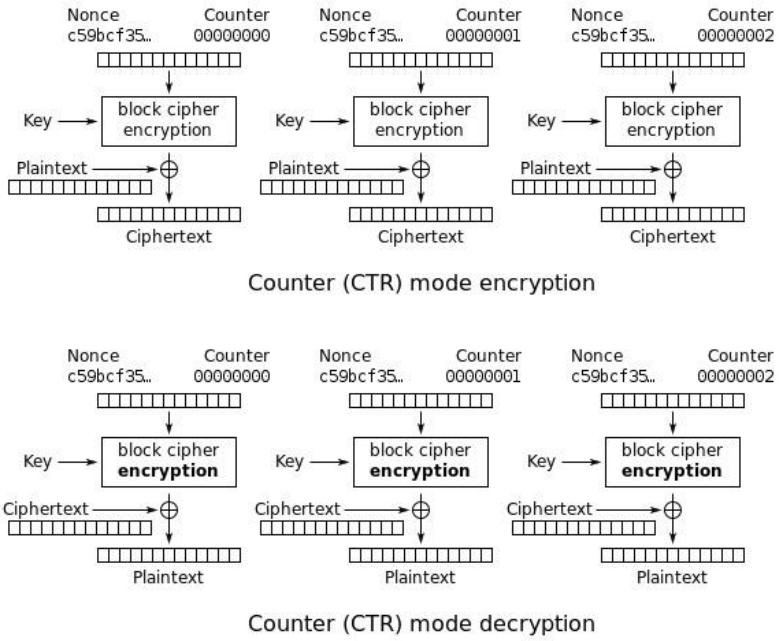


Figura 3.15. Modo de cifra CTR Counter. Fuente: Wikipedia.

3.5. Algoritmos de cifra en bloque: DES, 3DES, IDEA, AES

Existe una gran cantidad de algoritmos de cifra simétrica interesantes, más de una docena. No obstante, los más característicos y usados son:

- Data Encryption Standard (DES)
- Su variante triple DES o 3DES
- International Data Encryption Algorithm (IDEA)
- Advanced Encryption Standard (AES)

Analizaremos aquí en detalle DES, 3DES y AES, y solo superficialmente IDEA.

Algoritmo DES

El Data Encryption Standard DES ha sido el estándar mundial de cifra simétrica utilizado durante 25 años, desde 1974 hasta finales de 1999. Si bien el algoritmo nunca ha sido criptoanalizado ni se le han encontrado debilidades graves, tenía fecha de caducidad anunciada debido a que la clave real era solo de 56 bits y su tamaño era fijo. En los años 70 romper por fuerza esos 56 bits era computacionalmente imposible, pero a finales de los años 90 esto podía realizarse mediante un ataque distribuido en red y romper la clave en menos de un día.

La clave en un sistema de cifra en bloque será un número aleatorio, secreto y único que se intercambian previamente los interlocutores, o bien un número que genera quien desea cifrar un documento o archivo de forma local. Esto último se conoce como cifrado convencional.

El espacio de claves serán todos los valores posibles que esta podría tomar. En el caso anterior de una clave de 64 bits, estos valores van desde el valor 0, una cadena de 64 ceros, hasta el valor 2^{64} - 1 = 18.446.744.073.709.551.615, una cadena de 64 unos. Como la clave de cifra es un único número dentro de ese espacio de claves, y esta no cambia dinámicamente como sucedía con la cifra en flujo, los algoritmos simétricos deben diseñarse con un tamaño de clave que no haga viable un ataque distribuido por fuerza bruta. Obviamente, para encontrar un valor cualquiera dentro de un espacio de claves de n bits, hará falta en media realizar 2^{n-1} intentos o, lo que es lo mismo, recorrer en media la mitad de dicho espacio de claves.

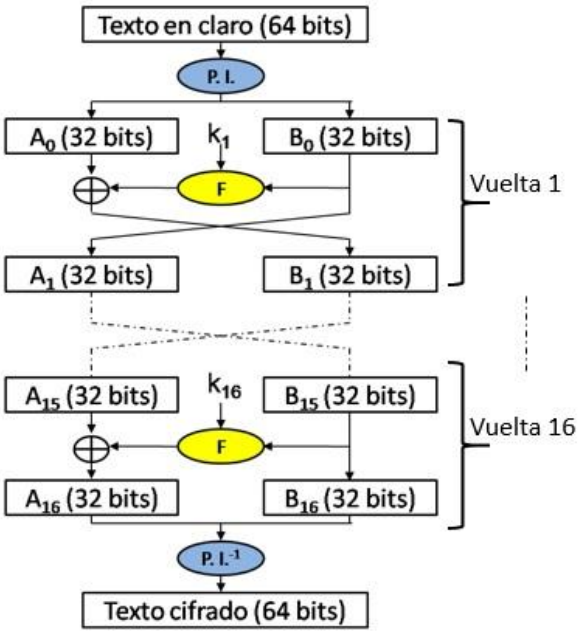


Figura 3.16. Esquema del algoritmo DES.

El algoritmo DES es un cifrador de tipo Feistel. Esto quiere decir que las operaciones de cifra de un bloque de texto en claro se realizan solo sobre una mitad de ese bloque de texto en claro y no sobre el bloque completo. Como el algoritmo tiene varias vueltas, en la siguiente vuelta se intercambian esas mitades izquierda y derecha como se aprecia en la figura 3.16.

La figura 3.17 muestra el bloque de 64 bits de texto en claro que se divide en dos mitades de 32 bits cada una, la mitad izquierda A_i y la mitad derecha B_i. La mitad derecha se mezcla con la clave de esa primera vuelta en una función denominada F, y su resultado se suma or exclusivo con la mitad izquierda del texto en claro. Hecho esto, este resultado se pasa como mitad derecha de texto en claro de la vuelta siguiente y la mitad derecha de la vuelta anterior se pasa como mitad izquierda de la nueva vuelta. Este proceso se repite durante

las 16 vueltas del algoritmo, tras lo cual se obtiene un bloque de 64 bits cifrados correspondiente a la cifra de ese primer bloque de texto en claro, acción que se repite hasta el último bloque de ese texto en claro.

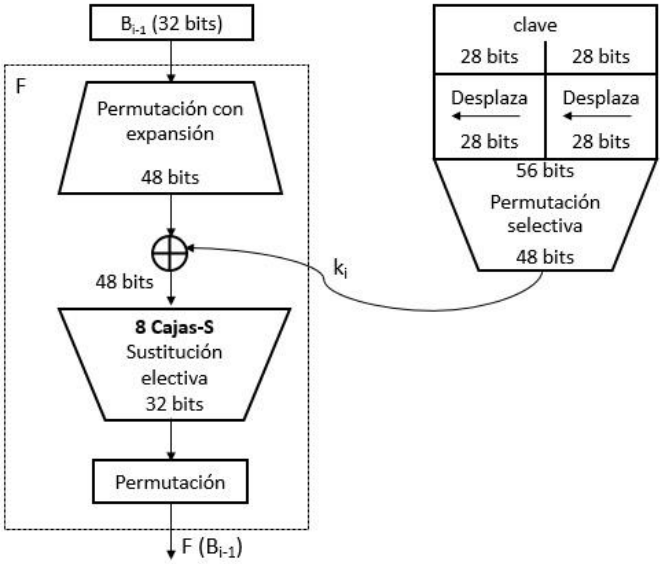


Figura 3.17. Operaciones en una vuelta del DES.

DES cifrará bloques de texto de 64 bits, con una clave de 64 bits que se reduce a 56 bits, y realizará 16 vueltas o rondas, donde se intercambian las posiciones de las mitades del bloque de texto en claro como se ha comentado. En cada vuelta usará una subclave k_i de 48 bits, derivada de la clave principal. Para lograr los objetivos de difusión y confusión, en cada una de esas vueltas se realizarán operaciones de sustitución y de permutación.

La reducción de la clave de 64 a 56 bits, que se hace nada más comenzar a ejecutarse el algoritmo, se debe a en aquellos años el ASCII que se usaba no era el extendido que hoy conocemos de 8 bits de datos, sino con solo 7 bits de datos y el octavo bit se reservaba para comprobar paridad. Por lo tanto no era un bit de clave y se elimina. Lo mismo sucede si la clave es un octeto numérico, el séptimo bit de cada byte es de paridad. Como 64 bits son 8 bytes, eliminaremos los últimos bits de cada byte, es decir, los bits 8, 16, 24, 32, 40, 48, 56 y 64.

Para poder realizar la operación XOR entre la mitad izquierda del texto en claro y el resultado de varias operaciones hechas sobre la mitad derecha, se procede a ampliar los 32 bits de texto en claro hasta los 48, añadiendo 16 bits, y rebajar los 56 bits de la clave a 48, reduciendo o eliminando 8 bits, tal y como se muestra en la figura 3.17. Esto se hace así porque la operación más importante de la función F son las cajas S que convierten una cadena de 48 bits en una cadena de 32 bits (los necesarios para el XOR final de cada vuelta) y que, al ser una función no lineal, dota de fortaleza al algoritmo.

Como se observa en la figura 3.17, existirán 8 cajas-S de forma que los 48 bits se distribuirán en cada una de ellas de seis en seis. Así, la caja S_1 tendrá como entrada los bits 1 al 6, la caja S_2 los bits 7 al 12, etc., y la caja S_8 los bits 41 al 48. En cada caja entrarán 6 bits y mediante un proceso que veremos a continuación saldrán 4 bits.

De los seis bits que entran en cada una de las 8 cajas, se leen los valores de los 2 bits de los extremos, que entregan 4 números posibles (00, 01, 10 y 11) y con ello tenemos el primer dato, un número decimal del 0 al 3 que marcará una de las 4 filas de esa caja. A continuación, se leen los cuatro bits interiores de dicha cadena de seis bits, que pueden entregar ahora 16 valores posibles (0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111), es decir números decimales entre 0 y 15, y que nos señalarán una de las 16 columnas de esa caja.

Por ejemplo, si como se muestra en la figura 3.18, los primeros 6 bits de la cadena de 48 bits, que entran por tanto en la caja S_1, son **0**1000**1**, entonces los bits 01 (negrita) marcan la fila 1 y los bits 1000 (subrayado) marcan la columna 8. La intersección será el número decimal 10 que en binario es 1010.

S_1	COLUMNAS															
FILAS		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Figura 3.18. Caja S_1 para una entrada 010001.

Las cajas S dan fortaleza al DES porque para romper el algoritmo deberíamos romper dichas cajas. Observa que, en el ejemplo de la figura, la salida decimal 10 puede obtenerse con los valores de fila (1) y columna (8) ya visto y, además, con estas otras tres entradas:

Fila 0, columna 9: Salida 10

Fila 2, columna 13: Salida 10

Fila 3, columna 12: Salida 10

Como hay 8 cajas S, para romper el algoritmo en sus 16 vueltas habrá que realizar $4^{8^{16}}$ operaciones, es decir 2^{256} intentos, que es mucho más difícil que romper el criptograma por fuerza bruta descifrando con cada una de las 2^{56} claves reales posibles.

Si para cifrar se usaban las 16 subclaves de 48 bits desde K_1 hasta K_{16}, para descifrar se usa el mismo algoritmo, pero las claves se utilizan ahora desde K_{16} a K_1. Esto es así porque en cada vuelta se produce un desplazamiento de las dos mitades de 28 bits de la clave de -1 o -2 bits, en donde el signo negativo quiere decir que es un desplazamiento hacia la izquierda. Si sumamos el desplazamiento aplicado a cada cadena de 28 bits en las 16 vueltas, observamos que es igual a 28 y por lo tanto las cadenas izquierdas C_0 y C_{16} estarán en fase, lo mismo que las cadenas derechas D_0 y D_{16}. Como los 56 bits de $C_{16}D_{16}$ están en fase con los 56 bits iniciales C_0D_0, para descifrar se usarán los mismos desplazamientos, pero en sentido inverso, es decir +1 y +2 bits, y así encontrar las claves que se usaron en el cifrado, recorriendo por tanto el algoritmo de forma inversa.

El DES ha sufrido diversos ataques distribuidos exitosos en red conocidos como *DES Challenge* desde finales del siglo pasado, propuestos por *RSA Laboratories*, básicamente porque su espacio de claves era muy pequeño. Después de 3 desafíos que comienzan en enero de 1997, DES finalmente sucumbe en enero de 1999 en el cuarto desafío. Aquí se unen la máquina DES Cracker y la organización distributed.net con 100.000 ordenadores conectados en Internet para romper la clave en solamente 22 horas, evaluando casi 250.000 millones de claves por segundo.

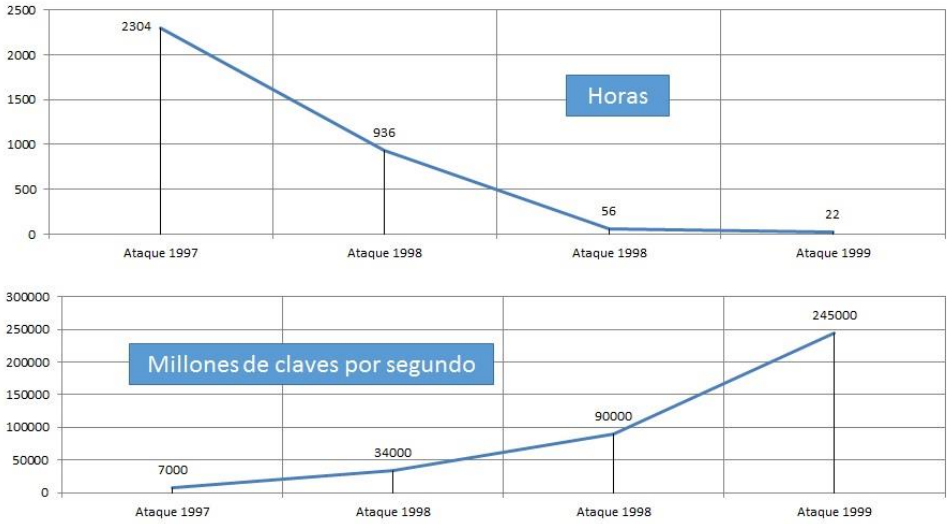


Figura 3.19. Resumen de los 4 DES Challenge.

Esta debilidad manifestada por el DES en estos ataques en red por el principio de "divide y vencerás", es el talón de Aquiles de todos los algoritmos de cifra simétricos. Al ser la clave única, es muy fácil establecer espacios de clave delimitados para el ataque y repartirlos entre varios clientes en red. Por este motivo, es recomendable usar hoy en día claves de al menos 128 bits.

Para tener una idea de qué significa en esfuerzo computacional romper una clave de n bits por fuerza bruta, hay que recordar que como dicha clave es aleatoria, el esfuerzo medio será de 2^{n-1} intentos. Es como intentar

adivinar un número del 0 al 15, es decir, de 4 bits. Si repetimos este experimento varias veces y vamos apuntando los intentos hasta que damos con ese secreto, en media ese valor será 8, es decir $2^{4-1} = 2^3$.

La figura 3.20 muestra una tabla con el tiempo que significaría romper una clave de diversos valores en bits, con un 50% de probabilidad como media, teniendo en cuenta la capacidad de cómputo lograda en el último DES Challenge. Por mucho que haya cambiado esa capacidad de cómputo o que queramos aplicar la ley de Moore, estos tiempos son astronómicos e inalcanzables para el cómputo actual, si no tenemos en cuenta los avances en la computación cuántica.

Longitud de la clave	Tiempo necesario para romper la clave
40 bits	2 segundos
48 bits	9 minutos
56 bits	40 horas
64 bits	14 meses
72 bits	305 años
80 bits	78.250 (2^{16}) años
96 bits	5.127.160.311 (2^{32}) años
112 bits	336.013.578.167.538 (2^{48}) años
128 bits	22.020.985.858.787.784.059 (2^{64}) años

Referencia de tiempo con números grandes	
Edad planeta	10.000.000.000 ($10^{10} = 2^{34}$) años
Edad universo	100.000.000.000 ($10^{11} = 2^{37}$) años

Figura 3.20. Tiempos de ataque para una capacidad computacional de 250.000.000.000 claves/segundo.

No obstante, recuerda que esto son solo probabilidades. Se podrían acertar todos los bits de una cadena de 128 bits al primer intento, si bien esa probabilidad es completamente despreciable.

Algoritmo 3DES

Como la clave tan pequeña del DES lo hacía vulnerable ya a comienzos de la década de los 90, en el año 1998 IBM propone un cifrado múltiple, es decir, se vuelve a cifrar el criptograma una o más veces con otras claves. En estos sistemas no lineales, esto hace que la clave efectiva aumente de tamaño, en el sentido de que cada vez se necesitan más operaciones para poder romperla. Nace así el triple DES o 3DES.

No se usa un cifrado doble porque mediante un ataque con texto en claro conocido, denominado Meet In The Middle (no confundir con Man In The Middle), se concluye que la clave del sistema aumenta tan solo en un bit

y, por lo tanto, no sale a cuenta tanto esfuerzo computacional para tan poco beneficio. En el cifrado triple, si se cifra con tres claves K_1, K_2 y K_3, todas de 56 bits, la fortaleza del 3DES será igual a 56*3, es decir, 168 bits.

El sistema que se utiliza, propuesto por IBM, es el denominado EDE, Encrypt Decrypt Encrypt, que se muestra en la figura 3.21.

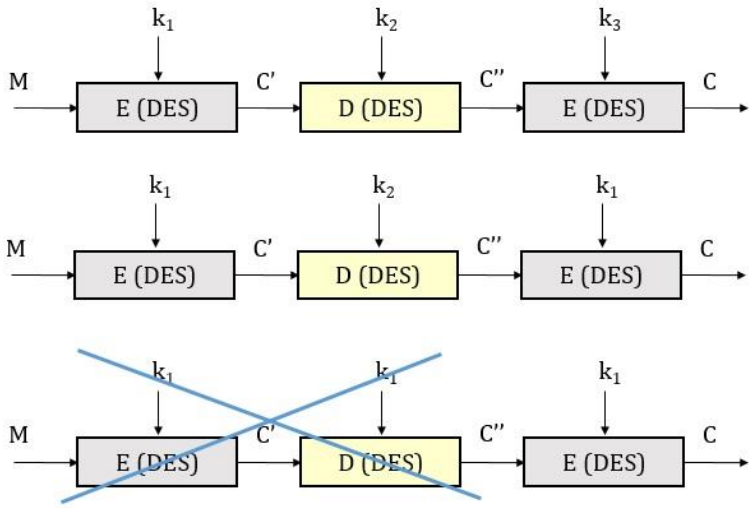


Figura 3.21. Esquemas del 3DES en modo EDE.

Observa en el primer esquema de la figura que primero se cifra con k_1, a continuación, se descifra con k_2 y finalmente se vuelve a cifrar ahora con k_3. Esto es exactamente igual que realizar tres cifrados con tres claves distintas y la fortaleza total será de $2^{56*3} = 2^{168}$, un valor muy seguro aunque el sistema de cifra sea muy lento.

Una segunda opción del algoritmo 3DES es la que se muestra en los dos esquemas siguientes, usar k_1 y k_2 como claves diferentes en el primero y hacer $k_2 = k_1$ en el segundo. En el primero de dos claves, la fortaleza del sistema será igual a $2^{56*2} = 2^{112}$. Tiene como ventaja que solo hay que intercambiar con el destino dos claves, no tres, pero su fortaleza podría ser algo débil a fecha de hoy. Este formato además permitía la opción de que los interlocutores definiesen las tres claves iguales, esto es $k_1 = k_2 = k_3 = k$, lo cual era muy interesante en aquellos primeros años en que se usaba este algoritmo, en tanto hacía compatibles sistemas modernos que tuviesen implementado el cifrado 3DES (por ejemplo un servidor) con sistemas más antiguos que tuviesen implementado solamente el DES (por ejemplo un cliente), como se muestra en el último esquema.

Algoritmo IDEA

En 1990 Xuejia Lai y James Massey proponen el algoritmo PES, Proposed Encryption Standard. No obstante, debido a los avances de los investigadores Eli Biham y Adi Shamir en el criptoanálisis diferencial aplicado al DES, en 1991 los autores proponen una mejora del algoritmo, denominado IPES Improved Proposed

Encryption Standard. Finalmente, en 1992 nace la versión definitiva del algoritmo IDEA, International Data Encryption Algorithm. A finales del siglo XX el algoritmo IDEA, mucho más seguro que el DES y que la versión simple del 3DES, se comienza a usar ampliamente en el sistema de correo electrónico seguro PGP.

IDEA es un cifrador no de tipo Feistel, que cifra bloques de 64 bits con una clave de 128 bits. Para ello, divide la entrada del texto en claro M en cuatro bloques de 16 bits y aplica un conjunto de operaciones lógicas a esos bloques de texto de 16 bits, con 52 subclaves también de 16 bits, calculadas a partir de la clave maestra de 128 bits, durante 8 vueltas o rondas, y añade por último una transformación final. En cada una de esas 8 vueltas usará 6 subclaves (8x6 = 48) y las últimas 4 subclaves (para llegar a las 52) las usará en la trasformación final, tal y como se muestra en la figura 3.22.

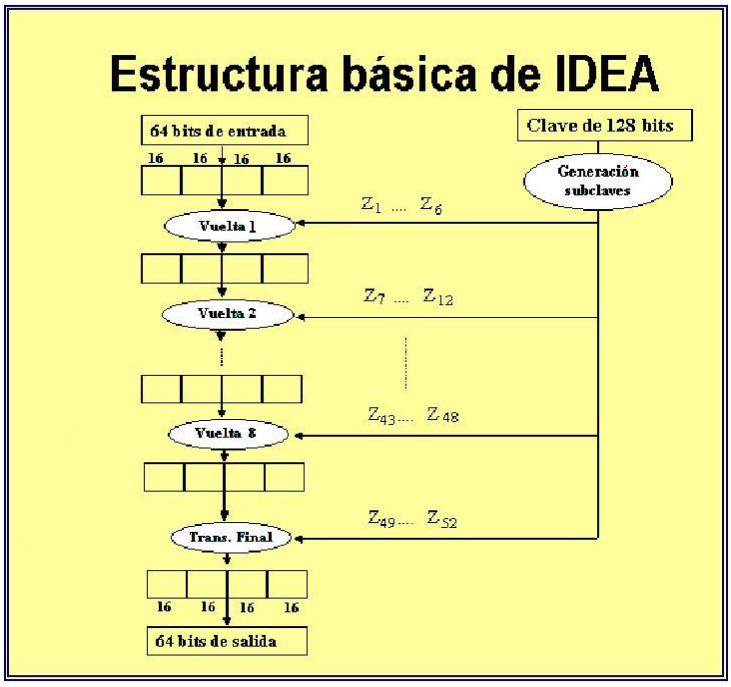


Figura 3.22. Esquema del algoritmo IDEA.

Para la generación de las 52 subclaves a partir de la clave de 128 bits, se procede de la siguiente manera: con los primeros 128 bits se generan 8 subclaves de 16 bits cada una, a continuación, se produce un desplazamiento de 25 bits a la izquierda y se obtiene una nueva cadena de 128 bits de la que nuevamente se extraen 8 subclaves, acción que se repite en 7 fases. Las operaciones lógicas del algoritmo entre palabras de 16 bits del texto en claro y de la subclave en cuestión, son suma módulo 2 o XOR, la suma módulo 2^{16}, es decir mod 65.536, y la multiplicación mod $2^{16}+1$, es decir mod 65.537. Por lo tanto, todas las operaciones en IDEA se realizan dentro de un cuerpo finito.

En la figura 3.23 se muestran los detalles de las operaciones del algoritmo IDEA, con las operaciones de texto en claro con subclaves de suma (Z_2 y Z_3), multiplicación (Z_1 y Z_4) y XOR (Z_5 y Z_6).

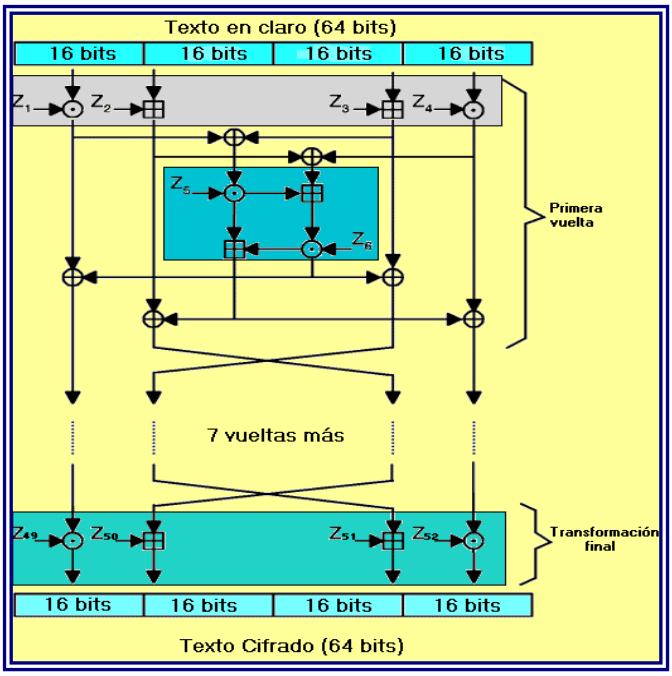


Figura 3.23. Operaciones del algoritmo IDEA.

Como las operaciones XOR, suma mod 65.536 y multiplicación mod 65.537 tendrán inversos en cada uno de esos módulos, para descifrar se recorre el algoritmo de forma inversa (por ello la existencia de la transformación final para que haya simetría) y se usan los valores inversos de las claves, es decir, Z_i^{-1} donde y cuando corresponda.

Algoritmo AES

El algoritmo AES nace como el sustituto del DES, el estándar mundial de cifra simétrica. Haciendo un poco de historia, el DES se adopta como estándar en 1976, el NIST lo certifica nuevamente como el estándar de cifra simétrica primero en 1987 y después en 1993. Durante esos años se populariza como algoritmo de cifra en todo el mundo y su uso más amplio lo encontramos en el cifrado de la información intercambiada en transacciones de dinero entre un cajero automático y el banco respectivo. En 1999 el NIST vuelve a certificar al DES como estándar pero solo en la versión 3DES, y llama a un concurso internacional para buscar un nuevo estándar mundial de cifra que se denominará AES, acrónimo de Advanced Encryption Standard. Precisamente entre 1997 y 1999 el DES se había enfrentado a los cuatro ataques o desafíos DES Challenge que hemos visto, una acción que impulsa y promociona la compañía RSA, en esa fecha y aún hoy estándar de cifra asimétrica, para demostrar que esa cifra simétrica estándar del DES era muy débil.

En las bases de la convocatoria del NIST se establece que el nuevo estándar debería soportar una longitud de bloque de 128 bits y una longitud de clave de 128, 192 y 256 bits, con la intención de que el nuevo estándar

pudiera utilizarse hasta bien entrado el siglo XXI. Se presentan 15 candidatos al concurso y después de más dos años, a finales del año 2000, se proclama vencedor al algoritmo Rijndael de los investigadores belgas Vincent Rijmen y Joan Daemen. En noviembre de 2001 el NIST anuncia oficialmente a través del Federal Information Processing Standard Publications FIPS 197 que el nuevo estándar para cifra simétrica del siglo XXI será el AES.

AES es un algoritmo no de tipo Feistel, que procesa por tanto bloques completos de texto en claro de 128 bits, con claves estándar de 128, 192 ó 256 bits. Para ello usa una matriz de estado de tamaño 4x4, cuyas 16 celdas o bytes van cambiando de valor de acuerdo a los procesos que ejecuta el algoritmo. En el cifrado se utilizan técnicas de sustitución y permutación, en algunos casos con operaciones polinómicas dentro de un cuerpo. Todas las operaciones de cifra dentro de la matriz de estado se realizan sobre bytes, en palabras de 32 bits que se escriben de arriba hacia abajo y de izquierda a derecha.

La figura 3.24 muestra el esquema del algoritmo AES y la matriz de estado de 16 bytes.

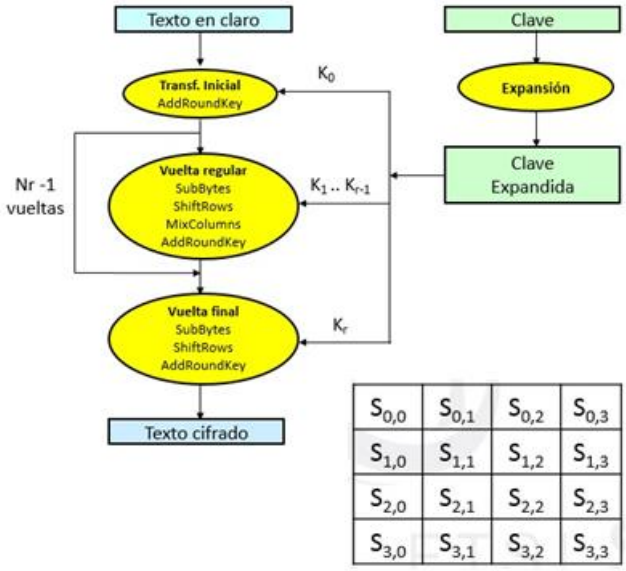


Figura 3.24. Esquema del algoritmo AES y matriz de estado.

Las cuatro operaciones básicas de cifrado en el AES son AddRoundKey, SubBytes, ShiftRows y MixColumns.

- La operación AddRoundKey ejecuta la suma or exclusivo entre los bytes del mensaje y los bytes de la clave.
- La operación SubBytes ejecuta una sustitución de cada uno de los 16 bytes de la matriz de estado mediante una tabla.
- La operación ShiftRows consiste en una permutación de las filas del estado de forma que la primera fila no rota, la segunda rotará un byte, la tercera rotará 2 bytes y la cuarta rotará 3 bytes.

- La operación MixColumns es algo más compleja y consiste en multiplicar cada una de las columnas de la matriz de estado por una matriz polinómica.

Aunque las operaciones que realizan algunas de estas funciones pueden representarse por tablas, de muy fácil comprensión, lo cierto es que están soportadas por ecuaciones polinómicas. Es decir, el algoritmo AES tiene un importante fondo matemático.

Operación SubBytes

Su finalidad es introducir no linealidad en el proceso. Cada uno de los bytes de la matriz de estado es sustituido a través de una Caja S de 8x8, es decir ante 8 bits de entrada, la salida muestra un valor distinto, pero también de 8 bits.

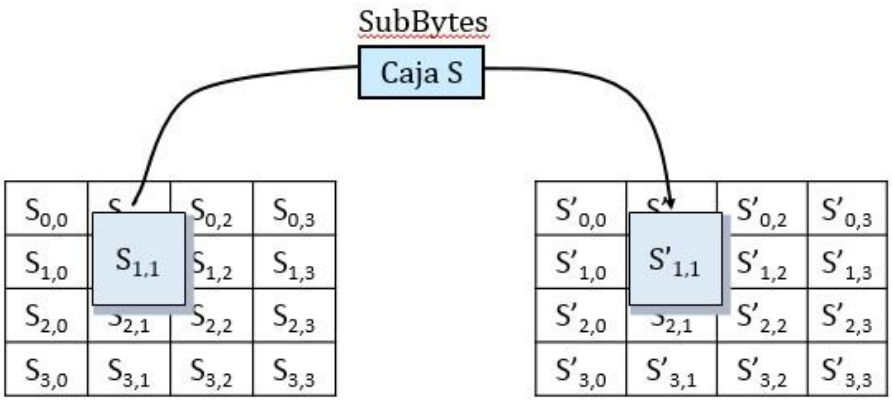


Figura 3.25. Esquema de la operación SubBytes. Fuente: Wikipedia.

Aunque hay una función matemática para encontrar estos valores y cuyo diseño pretende minimizar la relación entre la entrada y la salida, es más fácil observar que se trata de una matriz de tamaño 16x16, es decir 256 celdas en donde se introducen los 256 valores posibles en hexadecimal de 8 bits, desde el 00 hexadecimal igual a 0 en decimal, hasta la FF hexadecimal igual a 255 en decimal. Es decir, se trata de una simple tabla de reemplazo.

Operación AddRoundKey

Se realiza una suma módulo 2 (XOR) del estado con la subclave Kr, que es la última de las subclaves obtenidas a partir de la clave maestra K y siempre tendrá una longitud de 128 bits (16 bytes).

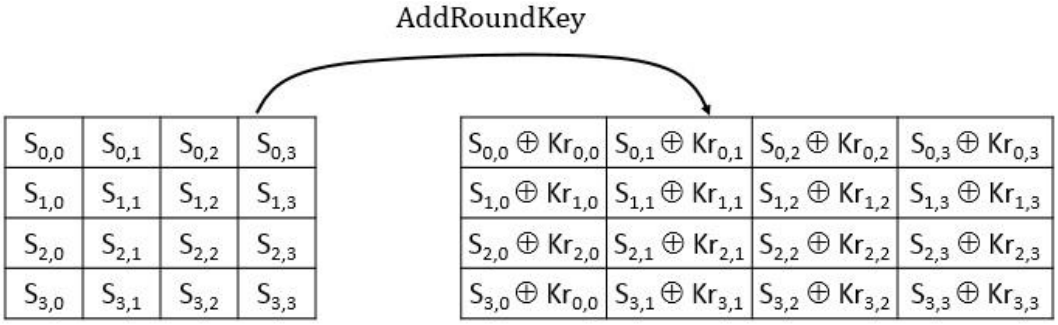


Figura 3.26. Esquema de la operación AddRoundKey. Fuente: Wikipedia.

Operación ShiftRows

En esta transformación se permutan cíclicamente los contenidos de las filas del estado. Tiene por objeto aumentar la difusión. Concretamente, la fila 0 no desplaza, la fila 1 desplaza un byte, la fila 2 desplaza dos bytes y la fila 3 desplaza tres bytes.

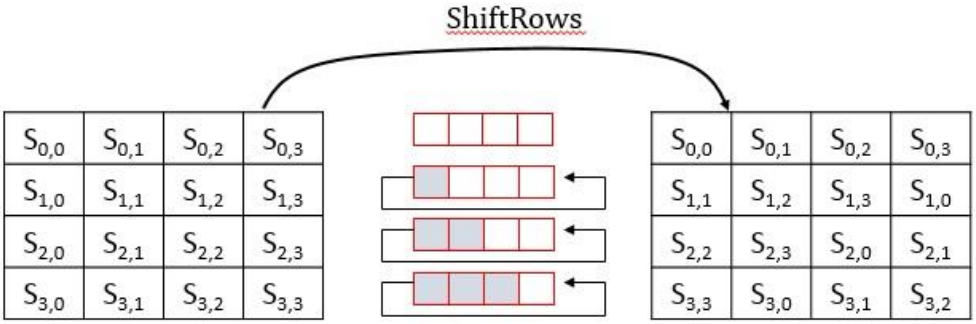


Figura 3.27. Esquema de la operación ShiftRows. Fuente: Wikipedia.

Operación MixColumns

Esta transformación opera sobre el estado, columna por columna, para maximizar la difusión. Concretamente, cada columna i se modifica de la manera que se indica en la figura 3.28.

$$\begin{pmatrix} S'_{0,i} \\ S'_{1,i} \\ S'_{2,i} \\ S'_{3,i} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} S_{0,i} \\ S_{1,i} \\ S_{2,i} \\ S_{3,i} \end{pmatrix}$$

Figura 3.28. Esquema de la operación MixColumns. Fuente: Wikipedia.

Se trata de operaciones con polinomios, en que cada byte es considerado como un polinomio de grado 8. Cada columna se multiplica módulo $x^4 + 1$ con el polinomio:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}, \text{ donde } \{03\} = x + 1; \{02\} = x; \{01\} = 1.$$

Expansión de clave

Como hemos visto, la longitud estándar de la clave en AES es de 128, 192 ó 256 bits, realizando 10, 12 ó 14 vueltas, respectivamente. Sin embargo, este algoritmo utiliza un total de Nr + 1 subclaves de 128 bits, es decir 16 bytes. Para ello, AES en primer lugar expande la clave mediante una serie de transformaciones hasta obtener una clave expandida de 16 (Nr + 1) bytes, a partir de la cual se obtienen las Nr + 1 subclaves.

Clave AES	Subclaves (Nr + 1)	Clave expandida
128 bits	11 de 128 bits	1.408 bits
192 bits	13 de 128 bits	1.664 bits
256 bits	15 de 128 bits	1.920 bits

Figura 3.29. Valores estándar de las claves en AES.

Para la generación de las 10 sub claves necesarias en el caso de una cifra con AES de 128 bits, se utiliza una función denominada Expansión de Clave, que consiste en modificar la matriz de estado de la clave mediante la operación RotWord que rota el primer byte de la última palabra de 4 bytes de la matriz, aplicar luego a esa palabra resultante la operación SubBytes y sumar después or exclusivo esta palabra con la palabra que se encuentra tres posiciones más atrás de esa matriz de estado y un vector conocido como Rcon, diferente para cada vuelta. Esto dará lugar a la primera palabra de la nueva matriz de estado de la clave.

Dichas operaciones se repiten 3 veces más, con lo que se obtiene una matriz de estado para la clave de 4 nuevas palabras de 32 bits cada una, correspondiente en este caso a la clave de la vuelta 1. Este proceso se realiza 10 veces para obtener 10 subclaves necesarias para cada una de las vueltas del AES 128.

El algoritmo de cifrado comienza con la función denominada AddRoundKey que realiza la suma or exclusivo entre los bytes del mensaje y los bytes de la clave. Para una clave de 128 bits, se calcularán 10 subclaves, una por cada vuelta, y se realizarán las siguientes cuatro operaciones durante nueve vueltas: SubBytes, ShiftRows, MixColumns y AddRoundKey. Para finalizar, se repiten solo las operaciones SubBytes, ShiftRows y AddRoundKey, dando lugar a una matriz de estado final con los 16 bytes que formarán el criptograma resultado de cifrar el primer bloque de texto en claro. Para el descifrado, se recorrerá el algoritmo por el camino inverso y se usarán las funciones inversas de estas operaciones conocidas como InvSubBytes, InvShiftRows, InvMixColumns e InvAddRoundKey.

Test del capítulo 3

1. Los registros de desplazamientos lineales entregan una secuencia máxima igual a:

A. 2^n+1 bits

B. 2^n bits

C. 2n-1 bits

D. 2^n-1 bits

2. En el algoritmo DES:

A. Se cifra bloques de 56 bits

B. Las claves internas tienen 32 bits de longitud

C. Se realizan 16 vueltas para cada bloque

D. Las claves internas tienen 64 bits de longitud

3. En el algoritmo DES las cajas S:

A. Reciben 4 bits de entrada y producen 6 bits de salida

B. Reciben 4 bits de entrada y producen 4 bits de salida

C. Reciben 6 bits de entrada y producen 6 bits de salida

D. Reciben 6 bits de entrada y producen 4 bits de salida

4. El algoritmo DES sucumbe a finales del siglo XX debido a:

A. Un ataque por criptoanálisis de estadísticas del lenguaje

B. Un ataque por criptoanálisis diferencial

C. Un ataque por fuerza bruta en red

D. Un ataque por correlación de bits

5. El 3DES generalmente se usa en un modo que sigue un esquema de:

A. Cifrado - Descifrado - Cifrado

B. Cifrado – Cifrado - Cifrado

C. Descifrado – Cifrado - Descifrado

D. Descifrado – Descifrado - Descifrado

6. El algoritmo DES y el algoritmo IDEA son idénticos en:

A. El tamaño de las claves utilizadas

B. El tamaño de los bloques para cifrar

C. El número de vueltas que ejecutan

D. Las operaciones que realizan dentro de un cuerpo finito

7. El algoritmo IDEA realiza en cada bloque un total de:

A. 64 vueltas

B. 52 vueltas

C. 8 vueltas

D. 16 vueltas

8. El algoritmo AES:

A. Es el sustituto de RSA como estándar de cifrado asimétrico

B. Es el sustituto de IDEA como estándar de cifrado simétrico

C. Es el sustituto de RC4 como estándar de cifrado simétrico

D. Es el sustituto de DES como estándar de cifrado simétrico

9. En el algoritmo AES, el tamaño estándar de la clave de cifra es:

A. 64, 128 y 256 bits

B. 128, 256 y 384 bits

C. 128, 192 y 256 bits

D. 128, 256 y 512 bits

10. Las vueltas del algoritmo AES en función de los tamaños estándar de clave son:

A. 10, 12, 14

B. 12, 14, 16

C. 10, 11, 12

D. 8, 16, 24

Tema 4

Autenticación y funciones hash

4.1. Introducción

4.2. Integridad y esquemas de autenticación

4.3. Características y propiedades de las funciones hash

4.4. Función hash MD5

4.5. Función hash SHA-1

4.6. Funciones hash SHA-256 y SHA-3

4.1. Introducción

Para profundizar en el tema puedes leer el capítulo 15 Funciones has en criptografía del siguiente libro.

Ramió, J. (2006). Libro electrónico de seguridad informática y criptografía, versión 4.1.

Disponible en: http://www.criptored.upm.es/guiateoria/gt_m001a.htm

Además, se recomienda la visualización de las siguientes píldoras formativas:

Píldora 43: ¿Qué son y para qué sirven las funciones hash?

Píldora 44: ¿Cómo funciona el hash MD5?

Píldora 45: ¿Cómo funciona el hash SHA-1?

Píldora 46: ¿Qué son SHA-2 y SHA-3?

Disponibles en:

https://www.youtube.com/watch?v=7MqTpfEreJ0&list=PL8bSwVy8_IcNNS5QDLjV7gUg8dIeMFSER

En este capítulo, haremos una introducción a los esquemas de autenticación, que usan o no cifrado. Analizaremos las características y propiedades de las funciones hash, así como su utilidad en la criptografía, para terminar estudiando las funciones hash más populares como son MD5, SHA-1, SHA-256 y SHA-3.

En capítulos anteriores se ha comentado que el uso de la criptografía nos permitiría proteger a la información, dotándole al menos de confidencialidad e integridad, siempre que aquella lo requiriera. La confidencialidad puede lograrse, por ejemplo, con los sistemas de cifra simétrica vistos en el capítulo anterior. En cuanto a la integridad, es menester indicar que en criptografía se habla de integridad del mensaje (que este sea íntegro, no modificado, completo, etc.) y de la integridad del emisor (que este pueda demostrar su identidad), si bien en este último caso lo normal es hablar de autenticidad del mismo. Es decir, la integridad en general se interpreta muchas veces como las dos cosas, integridad del mensaje y autenticidad del emisor.

Las funciones hash, en tanto actúan como una huella dactilar de un texto o documento, van a jugar un importante papel en ambas vertientes de la integridad.

4.2. Integridad y esquemas de autenticación

Por integridad de un mensaje se entiende que cuando dicho mensaje se envíe de una persona a otra, o bien se envíe de una máquina a otra, o simplemente se almacene en un sistema o base de datos, se pueda probar por cualquier interlocutor que éste no ha sido modificado durante su transmisión o almacenamiento, sin que el destinatario o interesado pueda percatarse de ello.

Así, la autenticación completa persigue que el mensaje se reciba o se guarde exactamente igual a como fue emitido y que, además, el emisor coincida con el que supuestamente dice que envía ese mensaje. Podemos puntualizar aquí que identificar sería la acción de aportar pruebas para demostrar una identidad, y que autenticar sería la acción de comprobar que la identidad aportada es la correcta.

Para verificar la integridad de los mensajes, usaremos mecanismos de autenticación, que pretenden evitar por ejemplo este tipo de ataques:

- Ataques por usurpación de identidad
- Ataques por modificación de contenidos
- Ataques por modificación de la secuencia de mensajes, incluyendo borrado e inserción
- Ataque por modificación del momento de llegada de los mensajes

Los ataques por usurpación de identidad, o bien inserción de mensajes de un emisor fraudulento, pueden solucionarse mediante el uso de certificados digitales. Estos son emitidos por una Autoridad de Certificación (una tercera parte de confianza, en inglés Trusted Third Part TTP) que permite autenticar la identidad de un usuario y garantizar que la información que se transmite en la operación electrónica es íntegra entre los usuarios o las entidades que la llevan a cabo.

En cuanto a los ataque por modificación de contenidos del mensaje, esto es, alterar datos o crear datos falsos, la solución se encuentra en el uso de funciones hash. El término hash se refiere a una función o método para generar resúmenes que representen de manera unívoca a un documento, registro, archivo, etc. También se utiliza el término función resumen o huella digital.

Los ataques por modificación de la secuencia de mensajes, incluyendo borrado e inserción, se solucionan mediante el uso de indicadores de secuencia en mensajes.

Por último, los ataques por modificación del momento de llegada de los mensajes, esto es, retrasar o repetir mensajes, tienen solución mediante el uso de sellos de tiempo, una técnica que se usa para proporcionar

información temporal sobre los documentos electrónicos, permitiendo validar el instante en el que un documento ha sido creado y, por lo tanto, comprobar que dicho documento no ha sido alterado desde entonces.

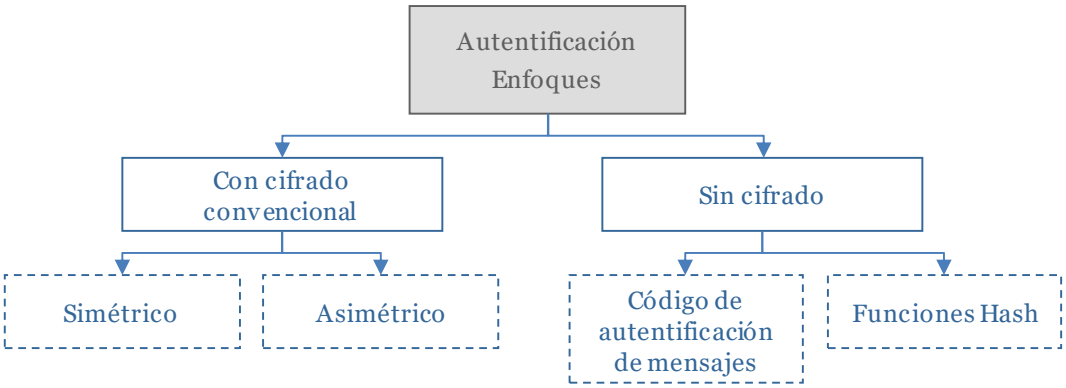


Figura 4.1. Esquemas de autenticación.

Aunque existen varios esquemas de autenticación, con cifrado y sin cifrado, en la práctica el más utilizado es aquel que incluye el uso de funciones hash y algoritmos de cifra asimétrica, dando origen a las firmas digitales.

Autenticación sin cifrado

En este mecanismo se genera una etiqueta de autenticación que se incorpora al mensaje para su transmisión. El mensaje mismo no está cifrado y se puede leer en el destino, independientemente de la función de autenticación en el destino.

Se puede usar un código de autenticación de mensaje MAC, Message Authentication Code, que genera un valor de longitud fija producido a partir de una función pública y una clave secreta compartida. O bien usar un hash, una función pública que genera un valor único, de longitud fija, a partir de un mensaje de entrada de cualquier longitud. La figura 4.2 muestra una autenticación sin cifra usando funciones MAC.

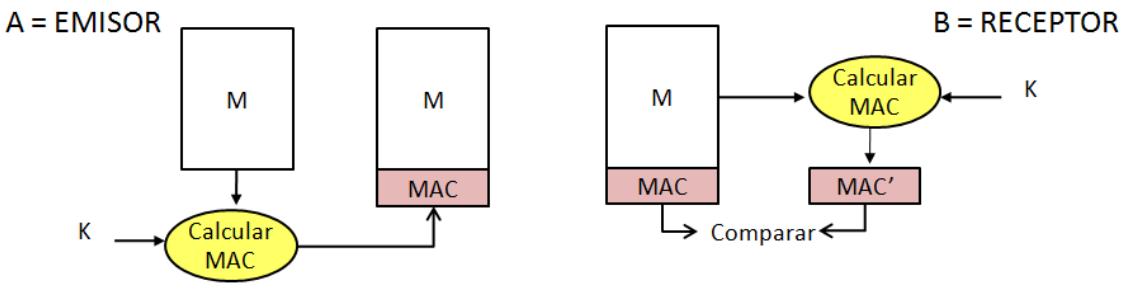


Figura 4.2. Autenticación sin cifrado vía MAC.

Siguiendo la figura 4.2, dos usuarios A y B comparten una clave secreta K. Dado el mensaje M de longitud variable y la clave K, el usuario A genera el Message Authentication Code, es decir un código de autenticación de tamaño fijo y pequeño (de unos 64 bits) denominado MAC = f (K, M). Por ejemplo, podría ser el último bloque de una cifra DES modo CBC. A partir del MAC y M no se puede obtener K y, además, la función f no debe ser reversible. Así, A envía a B el mensaje M y el MAC. El receptor B calcula el MAC con la clave K del mensaje M recibido y comprueba que coincide con el MAC recibido.

La figura 4.3 muestra un esquema de autenticación sin cifra mediante una función hash.

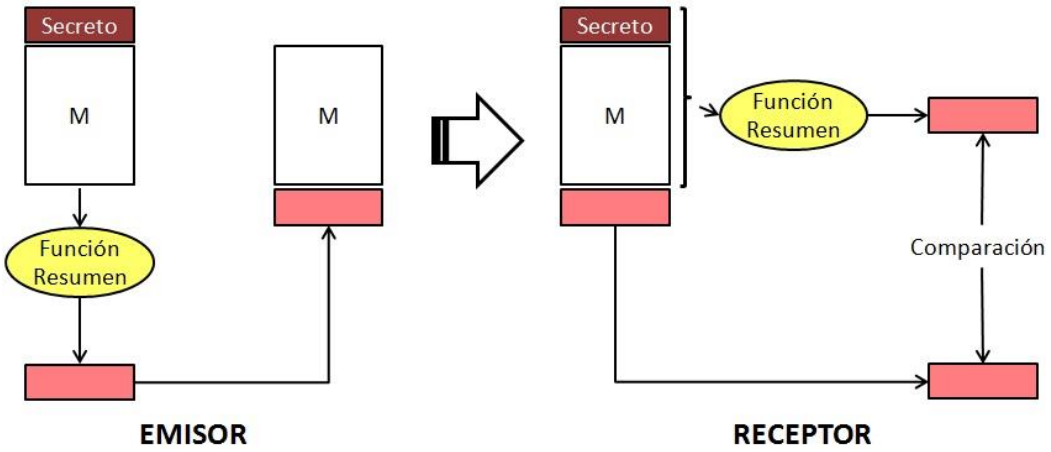


Figura 4.3. Autenticación sin cifrado vía función hash.

Al igual que en el caso anterior, el emisor y el receptor tienen que compartir un valor secreto, que se le añade al texto en claro solo para calcular el hash en emisión y en recepción, y que mientras éste permanezca en secreto, ningún intruso podrá generar un nuevo mensaje. Tampoco se produce cifra alguna y el protocolo que siguen emisor y receptor es el mismo.

Vamos a indicar que un hash o resumen h(M) realiza una función similar a los MAC, además son muy rápidas y más eficientes. Una función hash acepta un mensaje de longitud variable M como entrada y produce como salida una etiqueta de tamaño fijo h(M), conocida como huella o resumen del mensaje y, a diferencia del MAC, una función hash no usa ninguna clave secreta.

Para autenticar el mensaje, el resumen se envía junto al mensaje, de tal forma que el resumen del mensaje se puede utilizar en el receptor para autenticarlo de manera similar al esquema del MAC. Además de autenticación, el resumen proporciona integridad de los datos ya que, si se altera algún bit accidentalmente en el tránsito o de forma deliberada, el resumen de ese otro mensaje será distinto.

Autenticación con cifrado

Al usar cifrado en la autenticación, tenemos la opción de hacerlo mediante un cifrado convencional con clave secreta (simétrico) o un cifrado con clave pública (asimétrico). El uso más sencillo para conseguir autenticación de un mensaje M es enviar dicho mensaje en claro, junto con su resumen h(M) cifrado con la clave k compartida entre emisor y receptor de una cifra simétrica, como se muestra en la figura.

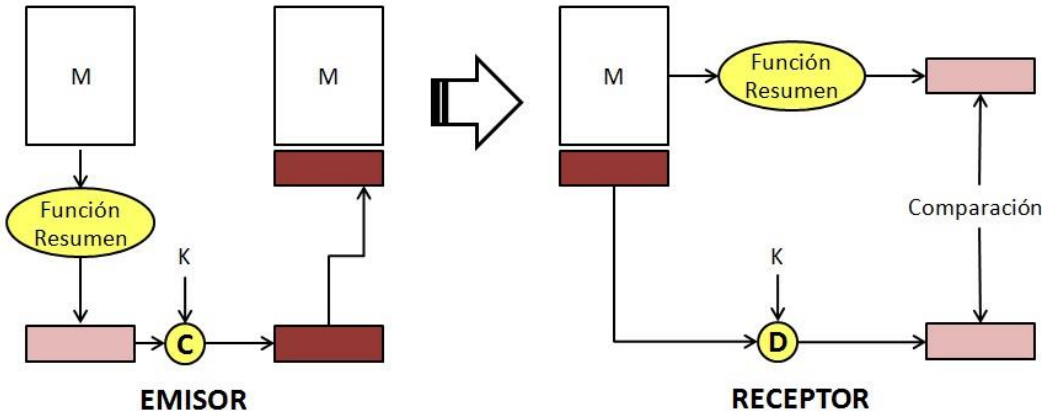


Figura 4.4. Autenticación vía hash con cifrado convencional.

En el cifrado convencional, si solamente el emisor y el receptor comparten la clave (que es lo que debería ocurrir), entonces solo el auténtico emisor sería capaz de cifrar un mensaje satisfactoriamente para la otra parte. Si el mensaje incluye un código de detección de errores y un número de secuencia, entonces se le asegura al receptor que no se han hecho alteraciones y que la secuencia es la adecuada. Y si el mensaje también incluye una marca de tiempo, entonces el receptor tiene la seguridad de que el mensaje no se ha retrasado más de lo normalmente esperado durante su tránsito por la red.

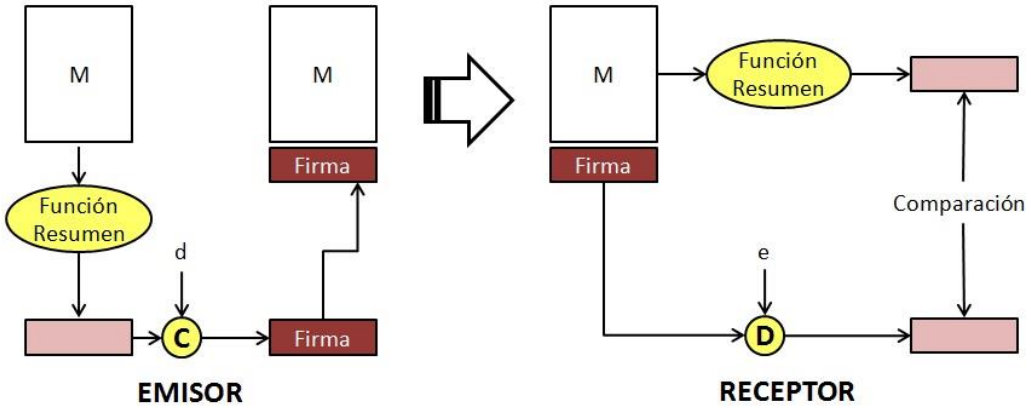


Figura 4.5. Autenticación con hash y cifrado asimétrico.

La figura 4.5 muestra la autenticación vía hash y criptografía asimétrica. En este caso se envía el mensaje M en claro junto con su resumen h(M) cifrado con la clave privada d del emisor. En recepción se descifra la firma con la clave pública e del emisor y se aplica la misma función hash sobre el mensaje.

4.3. Características y propiedades de las funciones hash

Características de las funciones hash

Como ya se ha indicado, las funciones hash son algoritmos que al aplicarlos sobre un mensaje, archivo o texto M, entregan un resumen de x bits conocido como h(M). Se trata de un número que a modo de huella digital representará a dichos documentos de forma supuestamente única. A diferencia de la popular familia de aplicaciones tipo zip para la compresión de archivos, las funciones hash entregan siempre un resumen de un número fijo de bits, independientemente del tamaño de ese archivo.

Es importante recalcar que, en tanto carecen de claves, las funciones hash no pueden considerarse propiamente como algoritmos de cifra.

Las funciones hash juegan dos papeles muy importantes en la criptografía:

- Como reducen el tamaño de un documento a una huella digital de un conjunto pequeño de bits, normalmente unas centenas de bits, permiten adaptar ese documento para su firma digital, en tanto los algoritmos de clave pública que se usan para la firma digital son muy lentos (Kbytes/seg) y no sería por tanto eficiente firmar documentos de miles o millones de bytes.

- Como una de las propiedades de estas funciones es que, si se altera algún bit del mensaje, accidentalmente durante el tránsito o bien de forma deliberada, el resumen de ese otro mensaje será completamente distinto, permitirá autenticar el mensaje de la siguiente manera:

    - Se envía al destinatario el mensaje M (supuestamente no es secreto) junto al hash calculado en origen.
    - El destinatario recibe el mensaje, que puede o no ser el mismo enviado y que llamaremos M', y calcula en destino el hash del mensaje recibido h(M').
    - Si el valor h(M) recibido y generado en emisión coincide con el valor de h(M') calculado en recepción, se acepta el mensaje como íntegro. En caso contrario, se rechaza.

Aunque actualmente existen muchos protocolos de autenticación que requieren contraseñas, como pueden foros, servidores de correo, etc. que trabajan bajo conexiones seguras, inicialmente lo que se almacenaba en el servidor no era la contraseña del usuario sino el hash de esa contraseña. Esto último no se hacía con el propósito de comprimir la contraseña, que con toda seguridad será más corta que el hash generado, sino con el propósito de que si un atacante tenía acceso al almacén de contraseñas, no podía ver la contraseña como texto en claro y, por lo tanto, no podía tener acceso al sistema, a no ser que rompiese esa contraseña por fuerza bruta u otro tipo de ataque. Hoy los tiempos han cambiado esta filosofía.

El caso anterior es un ejemplo clásico de una autenticación de usuario ante el sistema, en donde éste se identifica mediante algo que conoce, es decir una contraseña. Se trata de una autenticación débil, incluso si esa contraseña fuese fuerte. Lo ideal sería cubrir al menos dos de los tres métodos típicos de autenticación:

- Autenticación mediante algo que se sabe: contraseña o password, passphrase, …
- Autenticación mediante algo que se tiene: tarjeta de coordenadas, token, …
- Autenticación por algo que se es: fondo de iris, huella dactilar, palma mano, …

El tamaño de h(M) dependerá del algoritmo utilizado. Por ejemplo, como se muestra en la figura 4.6, para el mensaje Este es el texto de prueba, se devuelven 128 bits en MD5, 160 bits en SHA1 y 256 bits en SHA256.

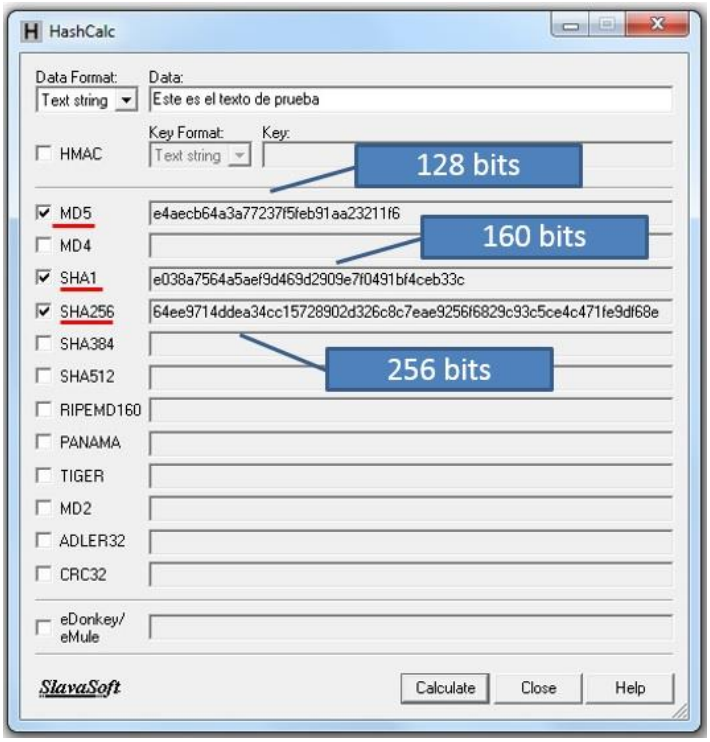


Figura 4.6. Diferentes hashes para la cadena de caracteres Este es el texto de prueba.

Como el resumen es un valor finito y pequeño de bits, la pregunta es ¿qué seguridad nos entregará un hash de x bits? Es decir, ¿qué tamaño mínimo en bits debería tener ese hash para asegurar que es único y que no existirá otro menaje diferente que entregue el mismo hash? Independientemente del algoritmo en cuestión, el hash nunca asegurará que no existe ese otro mensaje que produzca una colisión. Sin embargo, para valores de hash de centenas de bits, esa probabilidad será tan baja que puede despreciarse.

Supongamos por un momento que hemos creado una función hash de forma que el resultado es solo de 4 bits, independiente del tamaño del mensaje de entrada. Esto significa que el posible resultado será uno de estos 16 valores:

0000	0001	0010	0011	0100	0101	0110	0111
1000	1001	1010	1011	1100	1101	1110	1111

En este ejemplo, cada valor tiene una probabilidad igual a 1/16 y al ser el hash tan pequeño podría darse el caso de que fácilmente dos mensajes diferentes, incluso contradictorios, diesen el mismo hash. Piensa qué sucedería si el mensaje 1 Rechazamos el contrato y el mensaje 2 Aceptamos ese contrato, ambos mensajes de 22 caracteres, entregasen el hash 1011. Recibido el hash, ¿con qué mensaje nos quedamos?

Para evitar lo anterior, las funciones hash entregan resúmenes que van desde los 128 bits el más pequeño, hasta valores de 512 bits, siendo hoy en día recomendable usar 256 bits. Así, la probabilidad de que un hash colisione estará en torno a $1/2^{128}$ en el primer y peor caso, un valor extremadamente pequeño. Lo que sucede es que no solo intentaremos usar un hash que evite una colisión como la indicada anteriormente, sino que habrá otro tipo de ataques que reducirá de forma drástica este valor, como veremos a continuación.

Observa que es posible generar n documentos distintos pero que en el fondo digan lo mismo, y cuyos hashes sean obviamente distintos. El siguiente ejemplo muestra $1.024 = 2^{10}$ combinaciones de textos posibles con solo 10 cambios (los marcados en {negrita}) de un documento que en el fondo entrega el mismo mensaje.

> Estimado **{Querido}** amigo **{compañero}**:
> Te envío **{hago llegar}** esta carta **{nota}** para indicarte **{darte a conocer}** que tu **número {cupón}** ha sido premiado **{agraciado}** con el premio principal **{gordo}** de la lotería.
> Te envío **{Recibe}** un cordial saludo **{fuerte abrazo}**.
> María

Figura 4.7. Un texto cuyo mensaje es el mismo en 1.024 versiones.

Así, podríamos crear 2^n mensajes diferentes que digan lo mismo, como en el documento anterior, y simultáneamente, crear otros 2^n mensajes falsos que digan algo completamente distinto, por ejemplo que "no ha sido premiado" hasta que un hash de un documento verdadero coincida con el hash de un documento falso. Y, cuando esto suceda, usar uno de los mensajes verdaderos y uno de los mensajes falsos, ambos con hashes iguales, para realizar ataques a la integridad. Para este ejemplo, dicho ataque podría consistir en que la víctima acepta haber recibido el mensaje verdadero (ha sido premiado), y cuando lo quiere comprobar ante terceros o cobrar su premio, el atacante muestra que lo que la víctima había recibido firmado era realidad el hash del mensaje falso (no ha sido premiado). Esto es así porque lo que se firma es el hash y no el mensaje.

Por este motivo, para que las funciones hash sean interesantes y seguras en criptografía deberán cumplir un conjunto de propiedades que veremos a continuación.

Unidireccionalidad. Conocido un resumen h(M), será computacionalmente imposible encontrar el mensaje M a partir de dicho resumen.

Figura 4.8. Unidireccionalidad del hash.

Compresión. Como lo normal es que el mensaje M tenga una longitud de bits mayor que la que entrega el hash, la función hash actuará normalmente como un compresor. Si el mensaje tuviese menos bits que el hash, el resumen siempre tendrá ese valor fijo de bits.

Figura 4.9. Compresión del hash.

Facilidad de cálculo. Debe ser fácil calcular h(M) a partir de un mensaje M, y así sucede en la práctica. Además, el hash es una función muy rápida.

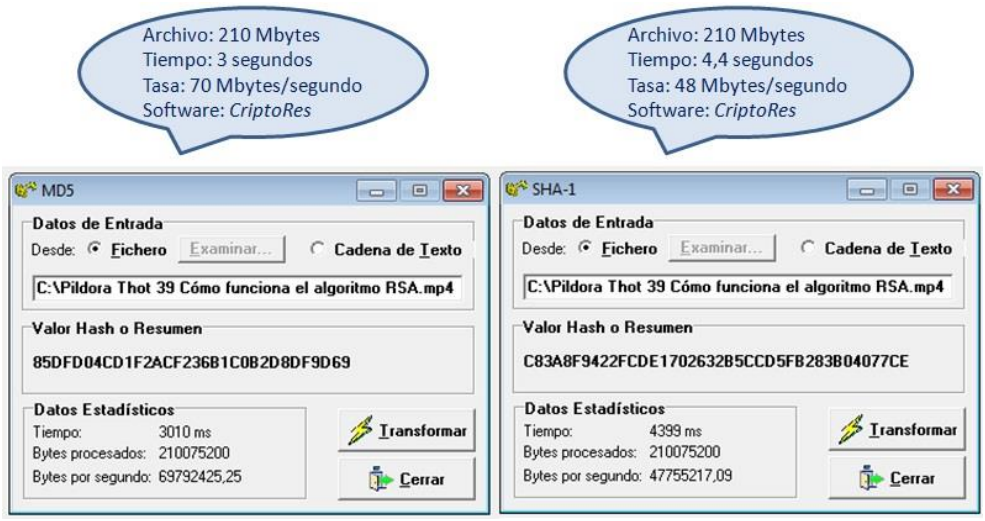


Figura 4.10. Facilidad de cálculo y rapidez del hash. Nota: datos según *software* CriptoRes. La tasa en MB/s con *software* o aplicaciones sin carácter educativo es más del doble de la mostrada.

Difusión de bits. Esta propiedad, también conocida como efecto de avalancha, indica que el resumen h(M) debe ser una función compleja de todos los bits del mensaje M. Por lo tanto, si se modifica tan solo un bit del mensaje M, el nuevo hash debería cambiar aproximadamente en la mitad de sus bits con respecto al anterior. Como se observa en la figura 4.11, el cambio de un solo bit entre los dos mensajes (la letra B = 0100001<u>0</u> y la letra C = 0100001<u>1</u>), hace que cambien 62 de los 128 bits del hash MD5. Son los unos en el resultado del XOR.

http://www.mobilefish.com/services/big_number_bitwise_calculation/big_number_bitwise_calculation.php#calculation_output

```
00110100111000010100001110010000111101100100101111110111100101010111111010110000111011111000101111010000011001110100001011011
10110111000010000001110100000010001101001111101011100101011110111010100100101011100101100110001101111011111011011000010100
                                                XOR
10000011111010010101111010010010110000101011011010000101001010001010101001101100110110011001110001111001111101100000100011100010
```

Figura 4.11. Difusión o avalancha en hash.

Resistencia débil a colisiones o primera preimagen. Esta propiedad se cumplirá si es computacionalmente imposible que, conocido un mensaje M, podamos encontrar otro mensaje M' tal que h(M) = h(M'). Por ejemplo, si el hash tuviese solo 20 bits, la posibilidad de que se dé una colisión sería de $1/2^{20}$ = 1/1.048.576, es decir, del 0,000095 %. Para hashes de 128 bits, este valor ($1/2^{128}$) ya es completamente despreciable.

Resistencia fuerte a colisiones o segunda preimagen. En este caso, nos indica que será computacionalmente imposible encontrar un par aleatorio de mensajes (M, M') de forma que h(M) = h(M'). Si el hash no cumple con esta propiedad, se facilitará su ataque por la paradoja del cumpleaños, que dice lo siguiente: en una sala con 23 personas, ya existe más de un 50 % de probabilidad que dos de ellas estén de cumpleaños en la misma fecha. Parece una paradoja, pero no lo es, se trata solamente de una serie y distribución de probabilidades.

Este es el talón de Aquiles de las funciones hash porque mediante el ataque basado en la paradoja del cumpleaños, podemos lograr confianza, es decir una probabilidad mayor o igual que el 50 %, en encontrar dos mensajes distintos con el mismo resumen h(M), buscando solo en un espacio de $2^{n/2}$ valores, en vez del espacio de 2^n valores del caso anterior de resistencia débil. La complejidad del ataque se reduce por tanto de forma drástica. Ante este tipo de ataque, una función hash MD5 tendría una fortaleza de solo 2^{64} en vez de 2^{128}. Más adelante veremos que MD5 tiene en la actualidad una fortaleza tan solo de 2^{24}.

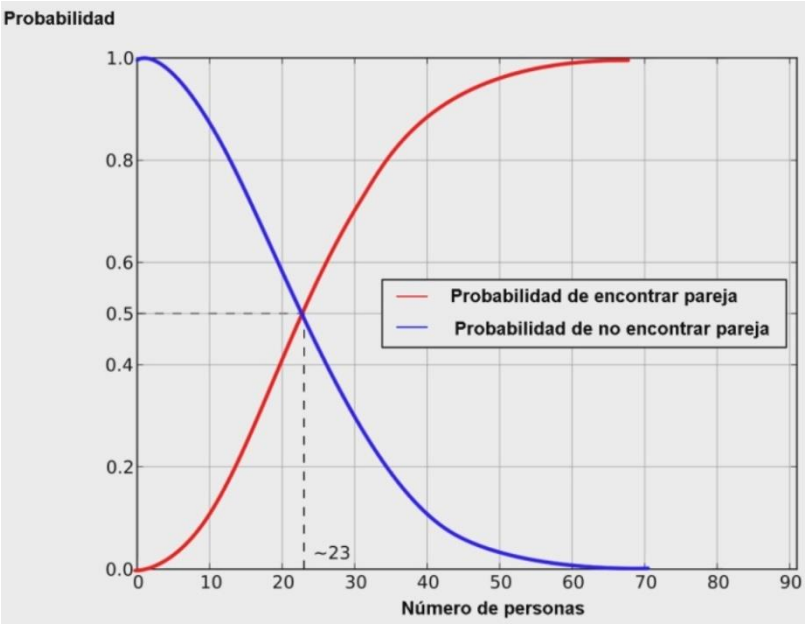


Figura 4.12. Gráfica de la paradoja del cumpleaños.

Volviendo entonces al ejemplo de la lotería presentado anteriormente, en este caso podríamos tener el siguiente escenario de ataque:

1. Bernardo firmará digitalmente con un hash de n bits un documento a Alicia.
2. Alicia escribe $2^{n/2}$ variaciones del mismo documento, todas válidas como ya se ha mostrado.
3. Además, prepara la misma cantidad ($2^{n/2}$) de variaciones de un documento falso o fraudulento.
4. Alicia va comparando el hash entre los dos conjuntos de variaciones del mensaje verdadero y del mensaje falso, hasta encontrar dos mensajes (uno válido y otro fraudulento) cuyos hashes coincidan.
5. Alicia envía a Bernardo el hash de la variante válida del documento legítimo para que Bernardo lo firme.
6. Bernardo firma ese hash y envía a Alicia esa firma.
7. Alicia podría cambiar ahora cuando lo desease el mensaje válido por el fraudulento, para demostrar ante el propio Bernardo o ante terceros que lo que Bernardo ha firmado es el mensaje falso y no el válido.

Observa que Alicia no ha tenido que generar 2^n documentos, sino que, en media, solo ha necesitado generar $2^{n/2}$ documentos.

4.4. Función hash MD5

Diseñada en 1992 por Ron Rivest, esta función resumen está obsoleta desde mediados de 2005 al detectarse graves fallos en su diseño y, además, al ser su resumen de solo 128 bits. No obstante, se ha seguido utilizando hasta bien entrada la década de 2010, especialmente en aplicaciones para comprobar la integridad de archivos, acceso a cuentas de correo, así como en redes P2P, si bien no en e-commerce ni banca online.

MD5 procesa los mensajes en bloques de 512 bits y produce una salida de 128 bits. En primer lugar, expande el mensaje M hasta una longitud exactamente 64 bits inferior a un múltiplo de 512 bits. Para ello, añade como relleno un bit 1 seguido de tantos bits 0 como sean necesarios, reservando los últimos 64 bits para añadir información sobre la longitud del mensaje.

MD5 tiene cuatro vectores iniciales ABCD, palabras de 32 bits cada uno, con los siguientes valores en hexadecimal:

A = 01234567, B = 89ABCDEF, C = FEDCBA98; D = 76543210

A los vectores ABCD, que no son secretos, y al primer bloque de 512 bits se le aplicarán 64 operaciones de 32 bits como se muestra en la figura 4.13.

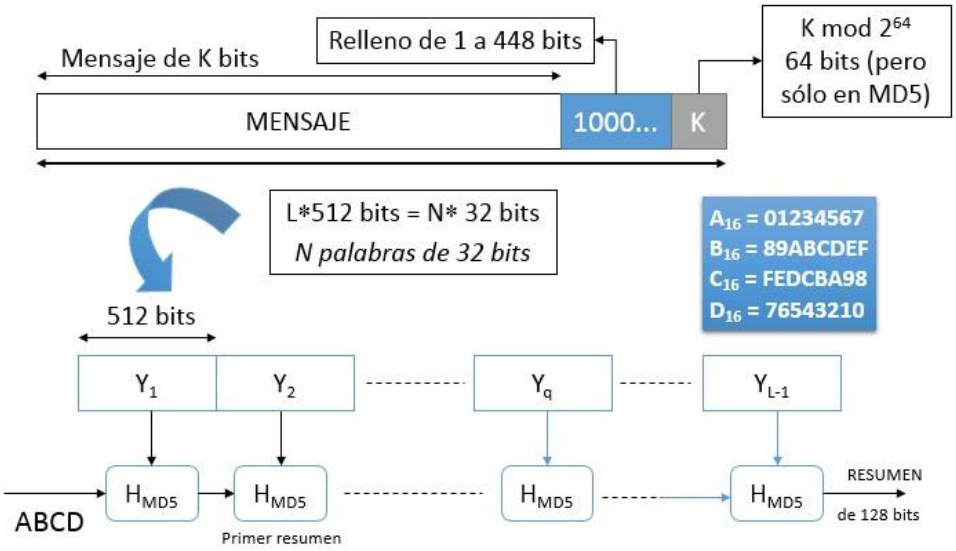


Figura 4.13. Formación de bloques en MD5 y esquema del hash.

Terminadas esas 64 operaciones o vueltas con el primer bloque de texto en claro, se obtienen cuatro nuevos vectores ABCD que serán los vectores de entrada para el siguiente bloque de 512 bits, acción que repite con los restantes bloques del documento, encadenando los sucesivos vectores ABCD. El último bloque ABCD entrega el resumen final h(M) de 128 bits del mensaje M.

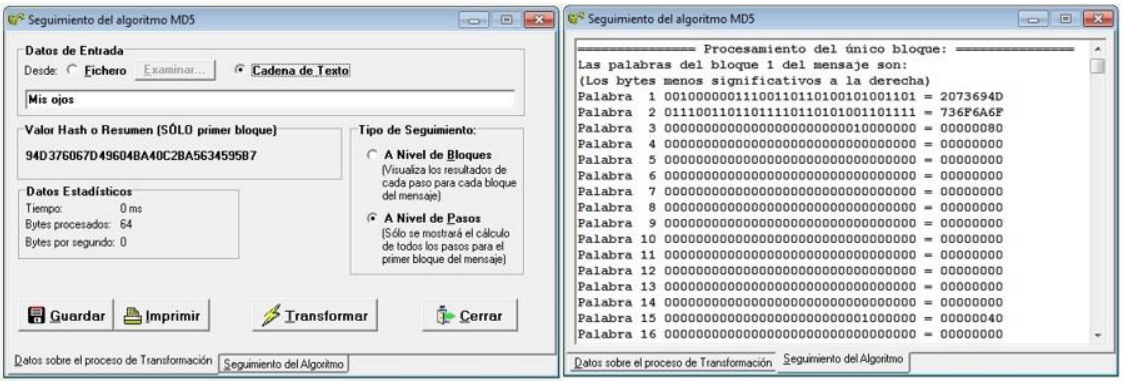


Figura 4.14. Seguimiento con CriptoRes del MD5 del texto Mis ojos, en litlle endian.

Como se muestra en la figura 4.14, (seguimiento del hash con el software CriptoRes), MD5 trata las palabras en formato little endian, es decir, primero se leen los bytes menos significativos de esa palabra de 4 bytes y después los bytes más significativos. De esta manera, los ocho bytes del mensaje Mis ojos, dos palabras de 32 bits cada una (Mis) y (ojos), que en hexadecimal es 4D 69 73 20 6F 6A 6F 73, en little endian se leerán [20 73 69 4D] [73 6F 6A 6F], donde se han separado las dos palabras para una mayor claridad.

MD5 realiza para cada bloque de entrada un total de 64 vueltas, con 16 operaciones en cada una de estas 4 rondas:

- 1ª ronda F = (B AND C) OR (NOT B AND D)
- 2ª ronda G = (B AND D) OR (C AND NOT D)
- 3ª ronda H = (B XOR C XOR D)
- 4ª ronda I = (C XOR (B OR NOT D))

En cada función FGHI se usan las 16 palabras del bloque de 512 bytes (16 x 32 bits), bloques que van desde M_0 hasta M_{15}. Así mismo, en cada vuelta de las 64 vueltas, se usan unas constantes K_i y desplazamientos s que vienen dados en unas tablas. La figura 4.15 muestra las operaciones que se realizan sobre los vectores ABCD, contando con los bloques de texto M_i, las contantes K_i y los desplazamientos s. La función F de la figura corresponde a las funciones FGHI de cada ronda y la suma es mod 2^{32}, no XOR.

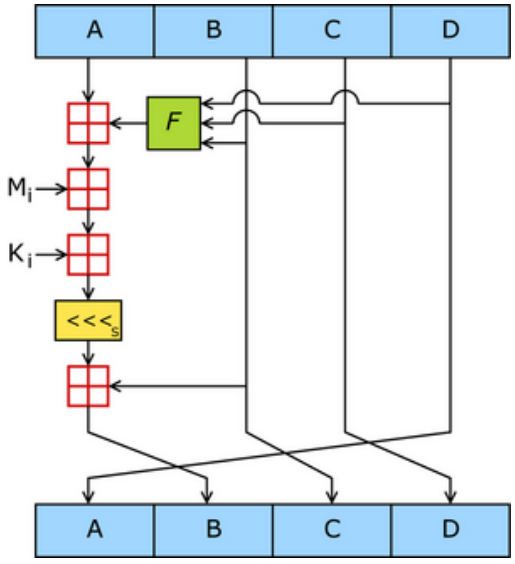


Figura 4.15. Operaciones en MD5. Fuente: Wikipedia.

En agosto de 2004, científicos chinos de la Shandong University presentan en el congreso Crypto 2004 un trabajo en el que se analizan las debilidades de funciones hash como MD5 y SHA-1 frente a colisiones. Le siguen investigaciones de Australia, USA y muchos otros países, demostrando en pocos años que MD5 era muy vulnerable.

Lo que encuentra el grupo de investigación de la Dra. Xiaoyun Wang se corresponde con un escenario en donde pueden generarse mensajes distintos de manera sistemática de forma que sus hashes MD5 colisionen. Son mensajes sin sentido, no texto.

La preocupación sobre las vulnerabilidades encontradas en estas funciones estriba en que muchos servidores Web aún presentan un certificado digital X.509 firmado, en el mejor de los casos, a partir de una función hash SHA-1, muy parecido a MD5 y también hoy en día cuestionado, si bien MD5 no se usa en dichos certificados desde hace varios años. Peor aún, el algoritmo MD5 se sigue usando en la comprobación de la integridad de documentos y archivos, generación de passwords, actualización y parches de *software*, etc. No es malo que se use para comprobar la integridad de archivos, lo peligroso es que aparezca solamente el hash MD5 para comprobar dicha integridad y no junto a otros hashes más seguros como SHA-1 y, mejor aún, SHA-2.

Aunque este tipo de ataques no podía derivar fácilmente en acciones de fraude, como sería la acción de suplantar un hash por otro igual que proviene de un mensaje con sentido pero distinto y falso, y que en recepción se aceptase como válido este último mensaje, tal y como se ha comentado en un ejemplo en este mismo apartado, solo pensar que esto pudiese ser una realidad en un futuro no muy lejano era ya un claro motivo de preocupación. Años antes de finalizar esa década, MD5 deja definitivamente de ser considerado un hash seguro y la opción pasa por usar el estándar del NIST conocido SHA-1. De hecho, existen muchos ejemplos de documentos y archivos de imágenes, que siendo completamente diferentes sus hashes MD5 colisionan. Es más, en páginas web se enseña cómo hacer para que colisionen.

4.5. Función hash SHA-1

SHA-1, acrónimo de Secure Hash Algorithm, propuesto inicialmente en 1993 por la NSA National Security Agency y adoptado como estándar en 1995 por el NIST, sustituye al primer algoritmo de resumen SHA-0 en el que se habían detectado problemas. SHA-1 es muy similar a MD5, ya que también trata bloques de 512 bits de mensaje y sus operaciones lógicas son muy similares, si bien genera un resumen de 160 bits pues cuenta con 5 vectores de 32 bits, ABCDE, en vez de los 4 vectores de MD5. En este caso, para cada bloque se realizan un total de 80 operaciones con palabras de 32 bits, organizadas en 4 rondas de 20 vueltas cada una.

Terminadas las primeras 80 vueltas, los vectores iniciales ABCDE habrán cambiado varias veces de valor, y serán los nuevos 5 vectores ABCDC los que se mezclarán con el segundo bloque de 512 bits de entrada. Esta acción se repite y se va encadenando con los siguientes bloques, hasta que el último valor de los vectores ABCDE es el resumen de 160 bits de todo el documento.

Los vectores en SHA-1 son:

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

La figura muestra 4.16 el seguimiento con el software CriptoRes del hash SHA-1 nuevamente del mensaje de 64 bits Mis ojos. En este caso, el tratamiento de las palabras MD5 es en formato big endian, es decir, primero se leen los bytes más significativos de esa palabra de 4 bytes y después los bytes menos significativos, como en la forma natural de escritura. Así, ahora, los ocho bytes del mensaje Mis ojos, que en hexadecimal era 4D 69 73 20 6F 6A 6F 73, en big endian se leerán [4D 69 73 20] [6F 6A 6F 73], donde se han separado las dos palabras para una mayor claridad.

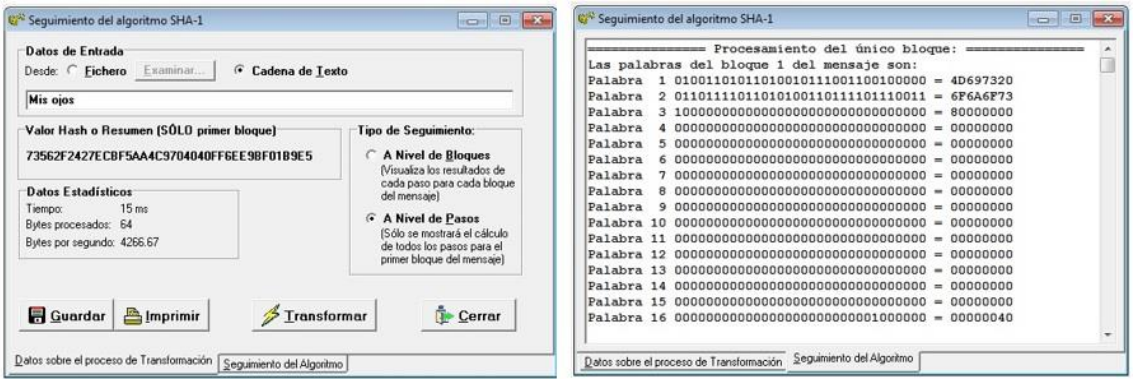


Figura 4.16. Seguimiento con CriptoRes del SHA-1 del texto Mis ojos, en big endian.

La figura 4.17 muestra las operaciones lógicas que realiza SHA-1 con sus cinco vectores ABDCE.

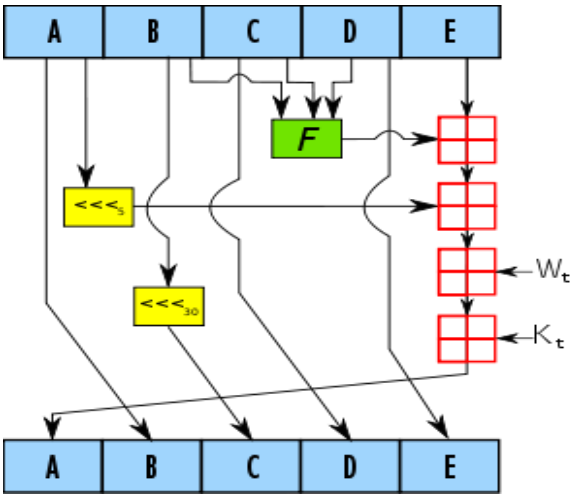


Figura 4.17. Operaciones en SHA-1. Fuente: Wikipedia.

Dependiendo de la ronda, la función F puede ser F, G, H o I, en donde:

- 1ª ronda F = (B AND C) OR (NOT B AND D)
- 2ª ronda G = (B XOR C XOR D)
- 3ª ronda H = (B AND C) OR (B AND D) OR (C AND D)
- 4ª ronda I = (B XOR C XOR D)

La suma es mod 2^{32}, W_t son 80 palabras de 32 bits sacadas del bloque de Y de 512 bits de entrada, K_t es una constante de 32 bits y existirán 4 valores diferentes, uno para cada una de las 4 rondas. Por último, se aplican desplazamientos de 5 bits en el vector A y de 30 bits en el vector B.

A cada bloque del texto de entrada se le aplicarán 20 vueltas con 4 funciones distintas, de forma tal que el número total de vueltas por bloque será igual a 20 x 4 = 80. Para obtener 20 palabras de 32 bits desde una cadena de tan solo 512 bits de texto de entrada, se procede de la siguiente manera: cada bloque de 16 palabras de 32 bits del mensaje se expandirá en 80 palabras, de forma que las palabras 0 a 15 se corresponden a los 512 bits del bloque, y las 64 palabras restantes, de la 16 hasta la 79, se obtienen mediante una operación lógica entre las palabras anteriores.

4.6. Funciones hash SHA-256 y SHA-3

Conocedores que SHA-1 no era del todo seguro, la NSA desarrolla en 2001 una familia de algoritmos hash conocida como SHA-2, que consiste en un conjunto de cuatro funciones hash de 224, 256, 384 y 512 bits. En estos dos últimos se trabaja con palabras de 64 bits. En la figura 4.18 se muestra el esquema de SHA-256.

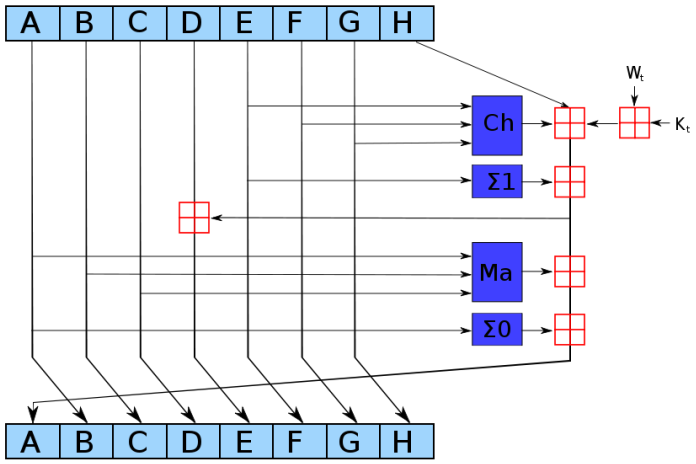


Figura 4.18. Operaciones en SHA-256. Fuente: Wikipedia.

Como se observa en la figura 4.18, tenemos 8 vectores de 32 bits cada uno, por lo que el resumen será de 8x32, es decir 256 bits. Las operaciones en color azul (Ch, $\Sigma 1$, Ma $\Sigma 0$) son similares a las de MD5 y SHA-1, W_t son 64 palabras de 32 bits que se obtienen de una forma similar a como lo hace la expansión en SHA-1, se cuenta con 64 constantes K_t y la suma sigue siendo mod 2^{32}.

$$\begin{aligned}
\mathrm{Ch}(E,F,G) &= (E \wedge F) \oplus (\neg E \wedge G) \\
\mathrm{Ma}(A,B,C) &= (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C) \\
\Sigma_0(A) &= (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22) \\
\Sigma_1(E) &= (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)
\end{aligned}$$

Figura 4.19. Operaciones en SHA-2.

Los algoritmos SHA-224 y SHA-256 usan bloques de entrada de 512 bits, con 64 vueltas. En cambio, para los algoritmos SHA-384 y SHA-512 se trabaja con palabras de 64 bits, la entrada es de 1.024 bits y se realizan 80 vueltas. Los cuatro algoritmos usan las mismas funciones lógicas que SHA-1, a las que se añade la función Shr, que es un desplazamiento a la derecha no circular.

A la vista del carisma que tomaban los ataques a las funciones hash, especialmente contra MD5 y en parte también contra SHA-1, voces reconocidas a nivel mundial pedían en 2005 que el NIST iniciase un concurso similar al del algoritmo AES para sustituir al DES para encontrar un nuevo estándar de funciones resumen. En aquel momento del estándar DES, el NIST estuvo atento y reaccionó rápidamente a los ataques DES Challenge, pero no así en este caso de las funciones hash. Finalmente, el NIST publica una nota de prensa el 23 de enero 2007 presentando un borrador, casi con dos años de retraso, en el que se propone un concurso para encontrar el nuevo estándar hash que se llamaría SHA-3.

En diciembre de 2010 el NIST selecciona en la tercera ronda a cinco finalistas, BLAKE, Ghøstl, JH, Keccak y Skein, y en octubre de 2012 se elige como ganador de SHA-3 al algoritmo Keccak, declarado como estándar mundial en agosto de 2015 por ese mismo organismo, si bien aún no se usa en la práctica, por ejemplo en TLS.

Debe tenerse en cuenta que desde que un algoritmo se declara estándar mundial hasta que su uso puede considerarse masivo, podrían pasan fácilmente 10 años. Por ejemplo, el AES solo comienza a usarse de manera masiva en comunicaciones SSL/TLS en 2010, diez años después de ser declarado por el NIST nuevo estándar mundial de cifra simétrica.

En cuanto al algoritmo SHA-3, comentar que no tiene un esquema como los de la familia SHA y MD5, que todos eran muy similares en su concepción, sino que se trata de un sistema conocido como esponja en el que la entrada es una cadena de bits de cualquier longitud, que son absorbidos por esa esponja, y se obtiene una salida del tamaño de bits que se desee exprimiéndola más o menos.

En su concepción, el algoritmo es bastante más complejo y se interpreta como un array en tres dimensiones, con movimientos similares a los que se realizan sobre un cubo de Rubik.

Para finalizar el capítulo, en la tabla de la figura 4.20 se muestra la fortaleza actual de las funciones hash más conocidas, basadas en el ataque por la paradoja del cumpleaños, sus valores teóricos y los que en realidad hoy se han alcanzado. Observa que MD5 ha podido ser atacado en junio de 2007 con tan solo 2^{24} intentos en un Pentium 4 de 2,6 GHz, encontrando una colisión en pocos segundos, y SHA-1 con 2^{57} intentos en noviembre de 2010 y 2^{52} intentos según unos investigadores australianos en 2009.

Función hash	Fortaleza teórica	Fortaleza real (2016)
MD5	2^{64}	2^{24}
SHA-1	2^{80}	2^{57}
SHA-256	2^{128}	2^{128}
SHA-512	2^{256}	2^{256}
SHA-3 256	2^{128}	2^{128}
SHA-3 512	2^{256}	2^{256}

Figura 4.20. Fortaleza de las funciones hash más conocidas. Fuentes diversas en Internet.

El hash MD5 está definitivamente roto y SHA-1 puede considerarse hoy en día ya inseguro. De hecho, se usó SHA-1 en certificados digitales X.509 hasta el 1 de enero de 2017, fecha en la que pasa el testigo a SHA-256. No se sabe cuándo se hará el cambio práctico a SHA-3.

Test del capítulo 4

1. Las funciones hash por sí solas permiten:

A. Autenticar a un usuario

B. Demostrar la integridad de un documento o texto

C. Firmar digitalmente un documento

D. Son válidas las tres anteriores

2. MD5, SHA-1 y SHA-256 tratan bloques de:

A. 128 bits

B. 256 bits

C. 512 bits

D. 1.024 bits

3. El hash MD5 realiza en cada bloque:

A. 64 vueltas

B. 16 vueltas

C. 80 vueltas

D. 128 vueltas

4. El hash SHA-1 realiza en cada bloque:

A. 64 vueltas

B. 16 vueltas

C. 80 vueltas

D. 128 vueltas

5. Indica cuál de estas afirmaciones es válida sobre el tamaño del hash:

A. MD5 entrega 124 bits y SHA-1 entrega 160 bits

B. MD5 entrega 128 bits y SHA-1 entrega 164 bits

C. MD5 entrega 128 bits y SHA-1 entrega 160 bits

D. MD5 entrega 128 bits y SHA-1 entrega 168 bits

6. MD5 usa un conjunto de vectores públicos de 32 bits, siendo su valor:

A. 3

B. 4

C. 5

D. 6

7. SHA-1 usa un conjunto de vectores públicos de 32 bits, siendo su valor:

A. 3

B. 4

C. 5

D. 6

8. Si el cambio de un bit en el mensaje cambia aproximadamente el 50 % de los bits del hash:

A. Se trata de la resistencia a primera preimagen

B. Se trata de la resistencia a segunda preimagen

C. Se trata de la propiedad de confusión del hash

D. Se trata de la propiedad de difusión del hash

9. La paradoja del cumpleaños reduce los intentos de ataque al hash a:

A. $2^{n/2}$ intentos

B. 2^{n}-1 intentos

C. 2^{n-1} intentos

D. 2^{n-2} intentos

10. El Nuevo estándar de hash desde 2013 se conoce como:

A. SHA-3 o Twofish

B. SHA-3 o Keccak

C. SHA-3 o Snow3G

D. SHA-3 o Rijndael

Tema 5

Algoritmos de cifra asimétrica

5.1. Introducción

5.2. Generalidades de la cifra asimétrica

5.3. Intercambio de clave de Diffie y Hellman

5.4. El algoritmo RSA

5.5. El algoritmo de Elgamal

5.1. Introducción

Para profundizar en el tema lee el capítulo 10 «Introducción a la Cifra Moderna» del siguiente libro.

Ramió, J. (2006). Libro electrónico de seguridad informática y criptografía, versión 4.1.

Disponible en: http://www.criptored.upm.es/guiateoria/gt_m001a.htm

Además, se recomienda leer las siguientes lecciones del MOOC Crypt4you El algoritmo RSA, Ramió, J. (2016).

Lección 0: Introducción al curso

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion0/leccion00.html

Lección 1. Los principios del algoritmo RSA

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion1/leccion01.html

Lección 2. Valores de diseño de las claves

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion2/leccion02.html

Lección 3. Cifrado de números y mensajes

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion3/leccion03.html

Lección 4. Claves privadas y públicas parejas

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion4/leccion04.html

Lección 5. Mensajes no cifrables

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion5/leccion05.html

Lección 6. RSA y el teorema chino del resto

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion6/leccion06.html

Lección 7. Generación de claves con OpenSSL

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion7/leccion07.html

Lección 8. Ataque por factorización

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion8/leccion08.html

Lección 9. Ataque por cifrado cíclico

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion9/leccion09.html

Lección 10. Ataque por paradoja del cumpleaños

http://www.criptored.upm.es/crypt4you/temas/RSA/leccion10/leccion10.html

También se recomienda la visualización de las siguientes píldoras formativas Thoth del mismo autor:

Píldora 38: ¿Qué es el intercambio de clave de Diffie y Hellman?

Píldora 39: ¿Cómo funciona el algoritmo RSA?

Píldora 40: ¿Es vulnerable el algoritmo RSA?

Píldora 41: ¿Cómo podemos atacar al algoritmo RSA?

Píldora 42: ¿Cómo funciona el algoritmo de Elgamal?

Disponible en: https://www.youtube.com/watch?v=7MqTpfEreJ0&list=PL8bSwVy8_IcNNS5QDLjV7gUg8dIeMFSER

En este capítulo veremos cuáles son las características de la cifra asimétrica o por clave pública, estudiaremos el intercambio de clave de Diffie y Hellman, así como los algoritmos RSA y Elgamal, el primero de ellos RSA con más profundidad.

La principal característica de la cifra asimétrica reside en el hecho de que la clave, que hasta ahora era única y secreta, se divide en dos partes, donde una de ellas se mantiene en secreto, pero la otra es pública. Como dichas claves serán inversas entre sí, esto nos permitirá realizar un intercambio de clave computacionalmente seguro y, además, firmar digitalmente un documento, algo que era imposible con los sistemas simétricos. La fortaleza de estos sistemas se basa en que, si no somos los dueños de la clave, y conocemos por tanto tan solo los valores públicos de esa clave, no podremos deducir o adivinar el valor de la clave privada, salvo que nos enfrentemos a un problema de tipo NP de muy difícil resolución para valores grandes como los que se usan en la práctica.

5.2. Generalidades de la cifra asimétrica

La cifra asimétrica, conocida también como cifra por clave pública debido a que una de las dos claves de cada interlocutor es pública y, por lo tanto, no secreta, tiene sus inicios en noviembre de 1976 cuando dos investigadores de la Universidad de Stanford, Whitfield Diffie y Martin Hellman, proponen un nuevo esquema de cifra en su artículo New directions in cryptography.

Este nuevo sistema criptográfico está basado en que la clave deja de ser única, y por tanto secreta, al estar compuesta por dos partes, una pública y conocida por todo el mundo, y la otra secreta, o privada, y conocida solo por su propietario. Mediante propiedades matemáticas de los cuerpos finitos, ambas claves son inversas entre sí y solo puede el dueño de esa clave a partir de una calcular la otra. Cualquier usuario que conozca la clave pública de otro usuario o de un equipo en el sistema, le será computacionalmente difícil descubrir la clave privada correspondiente, porque deberá enfrentarse a un problema de tipo NP que para números grandes sabemos se vuelve intratable.

Como cada usuario posee dos claves, una pública y la otra privada, inversas entre sí dentro de un cuerpo de forma que lo que una cifra la otra lo descifra, podremos realizar las siguientes tres operaciones:

- Cifrar con la clave pública del destino
- Cifrar con la clave privada del emisor
- Cifrar con la clave pública del emisor

De las tres operaciones indicadas anteriores, en la práctica solo se utilizan la primera y la segunda, como veremos a continuación.

Cifra con la clave pública del destinatario: Si se cifra por ejemplo un número N con la clave pública del destino, entonces solamente ese destinatario (y no otro) será capaz de descifrar el criptograma enviado, pues solo él o ella están en posesión de la clave privada inversa de la pública, que deshace la cifra y recupera así el secreto N. Por lo tanto, en este caso el mensaje N se habrá enviado de forma secreta o confidencial. Esto es lo que en la práctica se conoce como intercambio seguro de clave. Resumiendo, si ciframos con la clave pública del destino, se obtiene la confidencialidad.

Cifra con la clave privada del emisor. Si el emisor cifra por ejemplo un número N con su clave privada, podrá descifrar ese criptograma y recuperar el número N todo aquel que conozca la clave pública del emisor. El mensaje N en este caso no es un secreto porque lo pueden recuperar todos los usuarios que conozcan la clave pública del emisor, pero, sin embargo, quienes lo descifran pueden estar seguros de que ese mensaje es de ese emisor y no de otra persona, porque nadie excepto esa persona conoce la clave privada que se ha usado en la cifra. Ahora, en vez de confidencialidad, lo que se obtiene es autenticidad del emisor o, genéricamente, integridad. Esto es así porque si a ello unimos que lo que normalmente se cifra en el extremo emisor es el hash de un documento, entonces todo el proceso se conoce como firma digital y en el extremo receptor, además de esa autenticidad del emisor, se podrá comprobar también la integridad del mensaje debido a las propiedades de las funciones hash.

Cifra con la clave pública del emisor. Observa que se puede cifrar también con la clave pública del emisor, pero en este caso solo podría descifrar el criptograma el mismo emisor usando su propia clave privada, inversa de la anterior. Esto es lo que se conoce como cifrado convencional o cifrado local, característico por ejemplo si se quiere cifrar un documento de forma local en el disco duro o pendrive, pero eso ya lo hacen los algoritmos de cifra simétrica con una tasa de cifra mil veces superior. Por ello, esta opción no se usa, salvo en algunas versiones de malware tipo ransomware.

Obviamente, como emisores que somos, nunca podremos cifrar con la clave privada del destino, pues la desconocemos y además en la práctica será imposible de adivinar debido a su gran tamaño de miles de bits.

Aunque de forma genérica se hable de texto en claro o de mensaje M, lo cierto es que con estos sistemas asimétricos solo se cifran números. Más aún, números de unas pocas centenas de bits, como podrían ser una clave secreta o de sesión de un algoritmo simétrico como AES que se intercambia entre dos interlocutores, cliente y servidor, por ejemplo, o bien el valor de un hash SHA-256 que se va firmar digitalmente. Si se desea, se podrían cifrar textos, documentos o archivos con este sistema asimétrico, formando bloques de x bits siempre menores que el cuerpo de cifra, pero ello solo como un ejercicio de laboratorio, nunca como una cifra real o práctica.

Las figuras 5.1 y 5.2 muestran los esquemas de cifra asimétrica usando la clave pública de destino para lograr confidencialidad y usando la clave privada de emisor para lograr integridad.

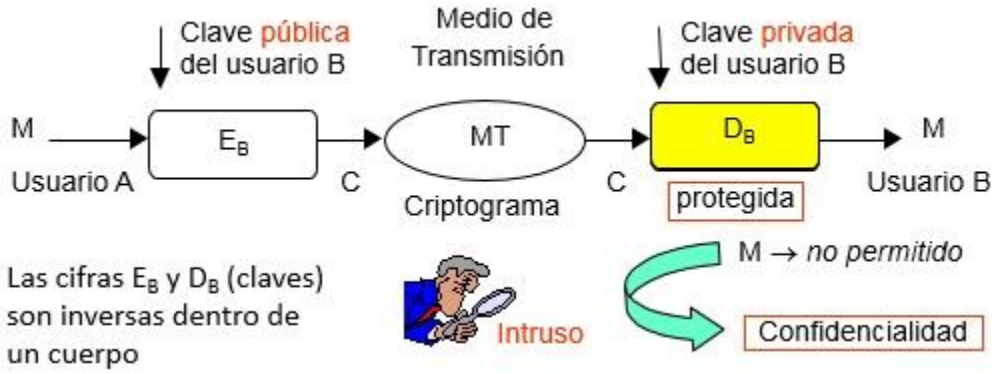


Figura 5.1. Cifra asimétrica con clave pública del destino: confidencialidad.

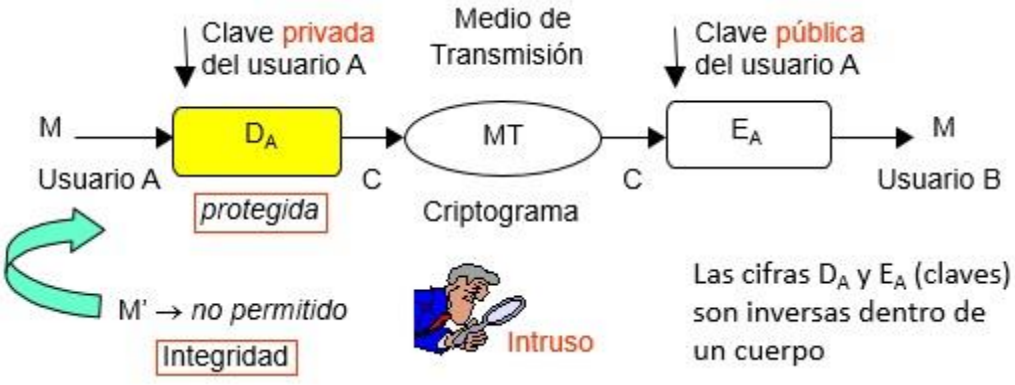


Figura 5.2. Cifra asimétrica con clave privada del emisor: integridad.

De las figuras se deduce que para estos sistemas de cifra asimétrica la confidencialidad y la integridad se logran por separado, al contrario que en los sistemas de cifra simétrica en que la confidencialidad y la integridad se lograban de forma simultánea. Esto último porque si la clave de cifra simétrica K es segura y no está en entredicho, es obvio que lo que A envía a B, y viceversa, es confidencial (nadie más posee la clave) y es, además, íntegro el mensaje y auténtico el emisor, porque no hay más claves que la que comparten A y B.

Lo anterior es muy importante porque nunca antes en criptografía se había podido separar ambos principios de la seguridad en una cifra, confidencialidad e integridad. Por lo tanto, en sistemas de cifra asimétrica podremos enviar un mensaje de forma solo confidencial, enviarlo solo con integridad, o bien enviarlo de forma que cumpla ambos principios, que sea confidencial y que además posea integridad.

Surgen aquí dos preguntas interesantes. ¿Se puede hacer una cifra doble en un sistema asimétrico? En caso afirmativo, si se puede cifrar un mensaje con ambas características, que tenga confidencialidad e integridad, ¿qué cifra hacemos primero? ¿Primero la cifra con la clave pública del destino y después la cifra con la clave privada del emisor, o al revés?

La respuesta a estas dos cuestiones es que sí puede hacerse una cifra doble y que es recomendable primero cifrar con la clave pública del destino y después cifrar con la clave privada de emisión. Obviamente en el primer caso se usará el cuerpo de cifra o módulo del destino y en el segundo caso se usará el cuerpo de cifra o módulo del emisor, ambos públicos. Para recuperar la información en destino, se deshace la cifra de forma inversa; es decir, primero se descifra lo último cifrado en emisión y después se descifra lo primero que se ha cifrado en emisión.

La ecuación que rige para la cifra de un secreto M dirigido al usuario B y su descifrarlo posterior en recepción será: $D_B[E_B(M)] = M$

La ecuación que rige para la firma digital F de un mensaje M por el usuario A y después que en recepción cualquiera pueda comprobar la firma será: $E_A[(D_A(F)] = F$

Así, si A desea enviar a B un mensaje cifrado y además firmado, tendríamos el siguiente escenario:

- A cifra el mensaje M con clave pública de B, para aportar confidencialidad: $E_B(M) = C$
- A cifra ese criptograma C con clave privada de A para aportar integridad: $D_A(C) = C'$

El usuario A envía al usuario B el criptograma dupla {C, C'}

- B descifra el criptograma C' con clave pública de A para comprobar integridad: $E_A(C') = C$
- B descifra el criptograma C con clave privada de B para recuperar el secreto: $D_B(C) = M$

Es decir: $D_B[E_A\{D_A(E_B(M))\}]$

Observa que las operaciones deben hacerse en orden. Si primero se firma y después se cifra, en recepción primero se recuperará el secreto y posteriormente se comprobará la firma. Por el contrario, si primero se cifra y después se añade la firma, en recepción primero se comprobará la firma y después se recuperará el secreto.

Esto se hace así porque como la firma digital es la segunda operación que hace A, será entonces la primera operación que deberá realizar B para comprobar que esa firma es válida, en el caso de que obtenga C. Si no se obtiene ese valor C, esto significaría que o bien el criptograma intermedio C no es íntegro o bien se ha firmado C por otro usuario y otra clave privada, y no con la clave privada de A.

Por lo tanto, ya no merece la pena continuar descifrando C porque lo recibido no es íntegro. Si consideramos además que en la práctica la operación de comprobar una firma digital con la clave pública del firmante es mucho más rápida que la operación de descifrar un criptograma con la clave privada del receptor, más a nuestro favor en seguir este protocolo.

¿Qué pasaría si A envía a B solo el último valor C y no la dupla {C, C'}? En este caso B debería descifrar C' con la clave pública de A (E_A) y el resultado descifrarlo con su clave privada (D_B), pero no tendría ninguna certeza de que en el primer descifrado efectivamente se haya comprobado la firma de A sobre el criptograma C, y que en el segundo descifrado se haya recuperado el mensaje secreto M descifrando C y no otro valor. No obstante, nada impide que podamos realizar esta operación, con la salvedad indicada.

En este nuevo escenario en que se recibe un único valor tras la doble cifra asimétrica, lo que nunca deberá hacer el receptor B es realizar las operaciones en el mismo sentido que las hizo A, siempre deberá ser en sentido inverso. Esto es, si A primero cifra y después firma, B entonces primero deberá comprobar la firma y después descifrar el criptograma.

Otra particularidad de la cifra asimétrica es que en la práctica se cifran siempre números, no mensajes. Lo normal es cifrar números de unas pocas centenas de bits, como puede ser una clave de sesión de cifra simétrica de AES, RC4, 3DES, etc., para un intercambio de clave, o bien el resultado de aplicar una función hash a un documento para una firma digital.

No obstante, sí es posible cifrar mensajes, codificándolos previamente, por ejemplo, mediante su código ASCII decimal, pero no es lo habitual. En ese caso, formaremos bloques de texto de n-1 bits, siendo n el número de bits del cuerpo de cifra, o bien de x-1 bytes, siendo x el número de bytes del cuerpo de cifra. Esto es así porque lógicamente los números a cifrar deben ser parte de los elementos o restos de ese cuerpo de cifra.

5.3. Intercambio de clave de Diffie y Hellman

En noviembre de 1976, dos investigadores de la Universidad de Stanford, Whitfield Diffie y Martin Hellman, proponen un algoritmo para intercambiar una clave secreta de manera computacionalmente segura, usando para ello funciones matemáticas de un solo sentido o unidireccionales. Esto significa que, si bien el sistema puede romperse matemáticamente, para los valores estándar de miles de bits que se usan en la práctica, el problema se vuelve intratable, tanto en recursos de cálculo como en tiempo. Y ahí radica su seguridad.

Diffie y Hellman usaron en su propuesta la función de un solo sentido conocida como el problema del logaritmo discreto PLD, cuyo enunciado recordamos a continuación. Encontrar el resultado de la expresión $\alpha^b \bmod p = x$ conocidos el generador α, el exponente b y el primo p, es sencilla y rápida, incluso para números muy grandes. Sin embargo, conociendo x, α y p, encontrar ahora el valor del exponente $b = \log_\alpha x \bmod p$, el problema del logaritmo discreto, se convierte en un problema de muy difícil solución para números grandes.

Si bien la operación de calcular el logaritmo de x en base α es muy simple, el hecho de aplicar después la reducción módulo p al resultado, aporta una gran complejidad al problema.

El valor α debería ser una raíz primitiva o generador del primo p. Recordemos que una raíz primitiva o generador α dentro de un primo p, es aquel resto de ese primo que genera el conjunto completo de restos, excepto el 0, al hacer la operación $\alpha^r \bmod p$, con r los restos del primo excepto el 0. Es decir, si decimos que α es una raíz primitiva del primo p, entonces $\alpha^1 \bmod p$, $\alpha^2 \bmod p$, $\alpha^3 \bmod p$,... $\alpha^{p-2} \bmod p$ y $\alpha^{p-1} \bmod p$, entregarán resultados distintos desde el 1 hasta p-1 y, por lo tanto, se generarán todos los restos del primo p.

En el primo p = 17, el número 3 es una raíz primitiva del primo porque entrega todos los restos, como se comprueba en la tabla de la figura 5.3.

$3^1 \bmod 17 = 3$	$3^2 \bmod 17 = 9$	$3^3 \bmod 17 = 10$	$3^4 \bmod 17 = 13$
$3^5 \bmod 17 = 5$	$3^6 \bmod 17 = 15$	$3^7 \bmod 17 = 11$	$3^8 \bmod 17 = 16$
$3^9 \bmod 17 = 14$	$3^{10} \bmod 17 = 8$	$3^{11} \bmod 17 = 7$	$3^{12} \bmod 17 = 4$
$3^{13} \bmod 17 = 12$	$3^{14} \bmod 17 = 2$	$3^{15} \bmod 17 = 6$	$3^{16} \bmod 17 = 1$

Figura 5.3. Comprobando que 3 es una raíz primitiva del primo 17.

Como se observa, están todos los restos de ese primo p desde el 1 hasta el 16. Observa, además, que el ciclo se repite cuando el exponente es igual a 0 porque $\alpha^0 \bmod p = 1$. Además, $3^{17} \bmod 17 = 3$, $3^{18} \bmod 17 = 9$, etc.

Si no se elige α como una raíz primitiva, el algoritmo de Diffie y Hellman seguirá funcionando, pero se vuelve menos seguro, en tanto no se dará la condición de que una clave pública (el resultado en la figura anterior) tiene una única clave secreta (el exponente), ya que no se generarán todos los restos como en la tabla 5.3, sino un conjunto de anillos con números que se repiten y de un tamaño evidentemente menor que el cuerpo p, tal y como se muestra en la tabla de la figura 5.4.

$4^1 \bmod 17 = 4$	$4^2 \bmod 17 = 16$	$4^3 \bmod 17 = 13$	$4^4 \bmod 17 = 1$
$4^5 \bmod 17 = 4$	$4^6 \bmod 17 = 16$	$4^7 \bmod 17 = 13$	$4^8 \bmod 17 = 1$
$4^9 \bmod 17 = 4$	$4^{10} \bmod 17 = 16$	$4^{11} \bmod 17 = 13$	$4^{12} \bmod 17 = 1$
$4^{13} \bmod 17 = 4$	$4^{14} \bmod 17 = 16$	$4^{15} \bmod 17 = 13$	$4^{16} \bmod 17 = 1$

Figura 5.4. Comprobando que 4 no es una raíz primitiva del primo 17.

Si la clave pública x fuese valor 13, cuando α = 3 y es una raíz primitiva, tiene como única clave privada (exponente) el valor 4, como se ve en la figura 5.3. Sin embargo, si α = 4 que no es una raíz primitiva, dicho valor público 13 puede conseguirse con los exponentes privados 3, 7, 11 y 15 como se ve en la figura 5.4.

El algoritmo propuesto por Diffie y Hellman es el siguiente. Alicia y Bernardo, que se encuentran distantes, eligen un primo p y un número α dentro de ese primo, conocido como generador del primo. Ambos valores son públicos. Acto seguido, cada uno elige un número secreto que no conoce el otro. Así, Alicia elige el número secreto a y Bernardo elige el número secreto b. A continuación, realizan el siguiente protocolo:

1. Alicia usa su clave secreta a, calcula $\alpha^a \bmod p = x_1$ y se lo envía a Bernardo.
2. Bernardo usa su clave secreta b, calcula $\alpha^b \bmod p = x_2$ y se lo envía a Alicia.
3. Bernardo recibe $x_1 = \alpha^a \bmod p$ y con su clave secreta b calcula: $x_1^b \bmod p = (\alpha^a)^b \bmod p$.
4. Alicia recibe $x_2 = \alpha^b \bmod p$ y con su clave secreta a calcula: $x_2^a \bmod p = (\alpha^b)^a \bmod p$.
5. La clave intercambiada entre Alicia y Bernardo será: $\alpha^{ab} \bmod p = \alpha^{ba} \bmod p = K$.

Como se observa, una vez terminado el protocolo, ambos comparten un mismo valor K que puede usarse como clave de sesión de forma segura. Esto porque cualquier intruso que conozca los datos públicos p y α, así como los valores intercambiados x_1 y x_2 por Alicia y Bernardo, para poder encontrar los valores secretos a de Alicia y b de Bernardo, y por tanto poder calcular $\alpha^{ab} \bmod p$, se enfrentará al problema del logaritmo discreto.

El hecho de que se logre la clave solamente cuando se haya concluido el protocolo y los interlocutores deban ejecutarlo de forma simultánea puede considerarse como una fuerte restricción. Existe, no obstante, una variante del algoritmo que permite intercambiar entre cliente y servidor una clave K, previamente generada por el cliente, y por tanto no requiere que el protocolo se ejecute simultáneamente como en el caso anterior.

Si Alicia desea entonces enviar a Bernardo un mensaje confidencial por correo, cifrando por tanto con un algoritmo simétrico y una clave de sesión K, y las claves asimétricas de ambos son las que se indican, se procede como se indica:

- Claves de Bernardo: Clave pública: $C_{púbB}$, α_B, p_B. Clave privada: b.
- Claves de Alicia: Clave pública: $C_{púbA}$, α_A, p_A. Clave privada: a.

Alicia hace lo siguiente:

1. Calcula una clave: $K_{AB} = (C_{PúbB})^a \bmod p_B$, que obviamente será $(\alpha_B{}^b \bmod p_B)^a \bmod p_B = \alpha_B{}^{ba} \bmod p_B = K_{AB}$.
2. Como $K_{AB} = \alpha_B{}^{ba} \bmod p_B$, de ese valor en el cuerpo de p_B (de al menos 1.024 bits), elige un conjunto de bits para una clave de sesión de la cifra simétrica, por ejemplo, los primeros 256 bits significativos para el AES-256 como clave K.
3. Alicia cifra el mensaje con AES y la clave simétrica K.
4. Alicia envía a Bernardo el mensaje cifrado con la clave K.
5. Alicia envía además a Bernardo el siguiente valor: $\alpha_B{}^a \bmod p_B$.

Bernardo hace lo siguiente:

1. El valor recibido $\alpha_B{}^a \bmod p_B$ lo eleva a su clave privada b.
2. Obtiene $(\alpha_B{}^a \bmod p_B)^b \bmod p_B = \alpha_B{}^{ab} \bmod p_B = \alpha_B{}^{ba} \bmod p_B = K_{AB}$.
3. Selecciona los mismos 256 bits que en origen y obtiene K.
4. Con esa clave simétrica K descifra con AES el mensaje cifrado.

5.4. El algoritmo RSA

Los inicios

Aunque la propuesta de Diffie y Hellman se convierte en un hito que revoluciona el mundo de la criptografía en aquellos años, hasta ese momento solo de tipo simétrica y de clave secreta, el sistema no permitía realizar una cifra real de información y menos aún la firma digital sobre un documento, tan solo un intercambio seguro de una clave.

Es en febrero de 1978, es decir poco más de un año después de la propuesta de Diffie y Hellman, cuando otros tres investigadores norteamericanos, Ron Rivest, Adi Shamir y Leonard Adleman, del Instituto Tecnológico de

Massachusetts MIT, proponen un sistema de cifra que llevará las iniciales de sus apellidos. El algoritmo se patentará como RSA.

A diferencia del intercambio de clave de DH, que basaba su fortaleza en la dificultad computacional de calcular logaritmos discretos en primos muy grandes, RSA basa su fortaleza en la dificultad computacional de factorizar un número compuesto muy grande, producto de dos primos grandes, y encontrar por tanto tales factores primos. Ambos problemas tienen una complejidad algorítmica muy similar y son inabordables hoy en día para la capacidad mundial de cómputo (ordenadores convencionales) cuando el módulo de cifra son números por encima de los mil bits.

La seguridad de RSA

La seguridad del algoritmo RSA se basa en la dificultad computacional que conlleva encontrar los dos factores primos de un número compuesto muy grande, resultado del producto de estos, donde sus primos también son números grandes. Esto es lo que matemáticamente se conoce como el problema de la factorización entera PFE, uno de los problemas no polinomiales o de tipo NP, muy usados en la criptografía.

Se trata de un problema que en un sentido el cálculo es muy fácil y rápido, por ejemplo, multiplicar dos números primos, pero que en sentido contrario, por ejemplo encontrar esos dos factores conocido solo el producto, se vuelve computacionalmente intratable a medida que la entrada es cada vez mayor. Es decir, requiere de unos recursos informáticos excesivos y, por tanto, de un tiempo de cálculo exorbitante.

Aunque no sea matemáticamente exacto, se podría aceptar que la primera operación es lineal en el sentido de que la cantidad de operaciones a realizar es directamente proporcional al tamaño de los operandos. En cambio, la segunda operación es de tipo exponencial, de manera que, por ejemplo, aumentando el tamaño de la entrada, el número de cálculos a realizar no aumenta proporcionalmente, sino mucho más, reflejándose esto en una curva de tipo exponencial del tiempo de cálculo en función del tamaño de la entrada. No obstante, la operación directa de multiplicación de dos números primos muy grandes es sencilla y consume escasos recursos computacionales.

Pero, ¿qué entendemos por un número muy grande aplicado a la criptografía? En el caso de RSA y ya en el año 2010, los valores mínimos de esos dos primos eran de 512 bits (unos 155 dígitos) en certificados digitales X.509 y, por tanto, su producto era un número de 1.024 bits (unos 310 dígitos). Aunque solo se ha llegado a factorizar a fecha actual números algo menores que 800 bits (768 bits en 2009), esto ya recomendaba el aumento de esos primos por ejemplo a 1.024 bits cada uno, con lo que se obtiene un producto o cuerpo de

cifra de 2.048 bits, que es el valor que hoy se usa en diversos programas, algunas aplicaciones, certificados digitales y pasarelas seguras de Internet.

Generando una clave RSA

El protocolo detallado para la generación de una clave RSA para Alicia y Bernardo es el siguiente:

1. Alicia y Bernardo eligen cada uno dos primos p y q aleatorios y diferentes, pero de valores iguales o superiores a 512 bits. Hoy en día es recomendable que sean al menos de 1.024 bits.
2. Los valores de los primos p y q serán un secreto, solo conocido por los propietarios de esas claves.
3. Alicia y Bernardo multiplican esos primos y obtienen el módulo de cifra n = p*q.
4. En el caso de Alicia ese módulo será $n_A = p_A * q_A$ y en el caso de Bernardo será $n_B = p_B * q_B$.
5. Alicia y Bernardo hacen público el módulo de cifra n_A y n_B.
6. Cada usuario calculará el Indicador de Euler ϕ de ese módulo n, en este caso $\phi_n = (p - 1)*(q - 1)$.
7. Así, Alicia calcula $\phi_{nA} = (p_A - 1)*(q_A - 1)$ y Bernardo calcula $\phi_{nB} = (p_B - 1)*(q_B - 1)$.
8. Ese valor va a ser su secreto o trampa, un número que solo conoce su dueño, al depender de los valores secretos p y q, y con el que definirá su clave pública y calculará su clave privada.
9. Cada usuario elegirá un valor de clave pública e en el intervalo $1 < e < \phi_n$.
10. Para asegurarse que exista el inverso multiplicativo, y por tanto la clave privada d, debe cumplirse que mcd [e, ϕ_n] = 1. En la práctica, esto no se hace puesto que todo el mundo usará como clave pública e el número 4 de Fermat $F_4 = 2^{2^4} + 1 = 2^{16} + 1 = 65.537$, un número primo y con ciertas características muy interesantes que ya se analizarán más adelante.
11. Si no se usa F_4, Alicia elegiría como clave pública un valor $1 < e_A < \phi_{nA}$ y Bernardo elegiría como clave pública un valor $1 < e_B < \phi_{nB}$.
12. Ese valor e será la segunda parte de sus claves públicas, además del módulo n.
13. A partir de esa clave pública e y de ϕ_{nA} y ϕ_{nB} Alicia y Bernardo calcularán sus claves privadas d_A y d_B usando para ello el Algoritmo Extendido de Euclides.
14. Alicia calcula $d_A = inv(e_A, \phi_{nA})$ y Bernardo calcula $d_B = inv(e_B, \phi_{nB})$.
15. Luego, cada usuario tiene dos valores que forman su clave pública (n, e) y un valor secreto que es su clave privada (d).
16. Además del valor d, cada usuario guardará también en secreto los primos p y q, que les servirá para acelerar la operación de descifrado mediante el Teorema Chino del Resto.

Por ejemplo, Alicia y Bernardo podrían crear estas claves RSA:

- Alicia elige como primos $p_A = 7$ y $q_A = 17$, con lo que su módulo de cifra será $n_A = 7*17 = 119$ y su indicador de Euler $\phi_{nA} = (7 - 1)*(17 - 1) = 6*16 = 96$.
- Alicia elige como clave pública $e_A = 5$ porque mcd (5, 96) = 1.
- Alicia calcula su clave privada $d_A = inv\ (5, 96) = 77$.
- Bernardo elige como primos $p_B = 11$ y $q_B = 13$, con lo que su módulo de cifra será $n_B = 11*13 = 143$ y su indicador de Euler $\phi_{nB} = (11 - 1)*(13 - 1) = 10*12 = 120$.
- Bernardo elige como clave pública $e_B = 7$ porque mcd (7, 120) = 1.
- Bernardo calcula su clave privada $d_B = inv\ (7, 120) = 103$.

Cifrando y descifrando con RSA

Si Alicia desea enviar un número secreto N (con confidencialidad) a Bernardo, realizará las siguientes operaciones. Como Alicia conoce la clave pública de Bernardo n_B y e_B, calcula $N^{eB} \bmod n_B = C$, enviando a Bernardo este criptograma C. Dado que Bernardo es el único que conoce su clave privada d_B, solo él podrá realizar la siguiente operación: $C^{dB} \bmod n_B$.

Observa que $C^{dB} \bmod n_B = [N^{eB} \bmod n_B]^{dB} \bmod n_B = [N^{eB}]^{dB} \bmod n_B$. Puesto que en la operación $[N^{eB}]^{dB} \bmod n_B$ los exponentes se anulan entre sí porque son inversos en ϕ_{nB}, en recepción Bernardo recuperará el mensaje secreto N enviado por Alicia dentro del cuerpo n_B. En realidad, lo que sucede es lo siguiente:

- Como $d_B = inv\ [e_B, \phi_{nB}]$, entonces $e_B*d_B = k*\phi_{nB} + 1$
- Por tanto $[N^{eB}]^{dB} \bmod n_B = N*k^{\phi nB + 1} \bmod n_B = N*N^{k*\phi nB} \bmod n_B$
- Por el Teorema de Euler tenemos que $N^{k*\phi n} \bmod n = 1$
- Luego $N*N^{k*\phi nB} \bmod n_B = N*1 = N$ y se recupera el mensaje

Por ejemplo, conociendo que los valores públicos de la clave anterior de Bernardo son $n_B = 143$ y $e_B = 7$, Alicia le enviará el valor secreto N = 75. Como la clave privada de Bernardo era $d_B = 103$, se procede entonces así:

- Alicia calcula $75^7 \bmod 143 = 114$, que envía a Bernardo.
- Bernardo calcula $114^{103} \bmod 143 = 75$, y recupera el secreto.

Descifrando con el Teorema Chino del Resto

Ya hemos comentado que los sistemas de cifra de clave pública son muy lentos si los comparamos con los algoritmos de cifra simétrica, por el tipo de operaciones que deben realizar. En el primer caso, normalmente se

trata de una elevación a potencia o exponenciación (DH, RSA, Elgamal) y en el segundo (3DES, IDEA, AES), será un conjunto de operaciones lógicas y desplazamientos de bits en registros, lo que les convierte en unas mil veces más rápidos que los primeros.

Como la operación que se realiza para la cifra RSA es N^{clave} mod n, siendo n un cuerpo actualmente recomendable de 2.048 bits, el talón de Aquiles en cuanto a cómputo estará en los valores que puedan tomar el número N que se cifra y/o la clave que usemos en el exponente en dicha cifra dentro del cuerpo n.

En un intercambio de clave entre un cliente y un servidor, por ejemplo, una clave de sesión K de 256 bits para el algoritmo simétrico AES, la operación que debe realizar el cliente $C = K^{eS} \bmod n_S$, es decir 256 $bits^{17\ bits}$ mod 2.048 bits, y la que debe realizar el servidor para descifrar será $C^{dS} \bmod n_S$, es decir 2.048 $bits^{2.048\ bits}$ mod 2.048 bits, porque tanto C como d_S tendrán valores de ese orden. Esta última operación es muy costosa en términos de cómputo.

Es aquí donde entra en acción el teorema chino del resto (TCR), en el descifrado de un intercambio de clave, puesto que las dimensiones de todos los números que intervienen en la ecuación son muy grandes. El servidor, en vez de realizar el descifrado en el cuerpo n de 2.048 bits, al ser dueño de la clave y poseer los primos p y q, podrá realizar un conjunto de operaciones para el descifrado de ese criptograma, pero en los cuerpos p y q, de la mitad de tamaño en bits que el cuerpo de cifra n, y por tanto reducirá de una manera significativa el tiempo de cómputo. Se trata de un atajo que solo puede hacerlo el dueño de la clave y que optimizará el tiempo de cálculo en unas 4 veces.

Para el sistema RSA podemos usar cualquiera de las dos representaciones del Teorema Chino del Resto que veremos a continuación, la fórmula estándar o bien la fórmula de Garner, que es la más utilizada.

TCR con fórmula estándar

$$M = \{A_p[C_p^{dp} \pmod p] + A_q[C_q^{dq} \pmod q]\} \bmod n$$

En donde:

$A_p = q\ [inv\ (q, p)] = q^{p-1} \bmod n$

$A_q = p\ [inv\ (p, q)] = p^{q-1} \bmod n$

$dp = d \bmod (p-1)$; $dq = d \bmod (q-1)$ (datos precalculados)

$C_p = C \bmod p$; $C_q = C \bmod q$ (datos precalculados)

Vamos a utilizar esta fórmula del TCR y posteriormente la de Garner en el siguiente ejercicio. Se recibe como criptograma el valor decimal 582.819 y debemos descifrarlo usando el TCR, sabiendo que los primos son p = 859, q = 907 (n = 779.113), que e = 7.481 y que d = 17.249.

$A_p = q^{p-1} \bmod n = 907^{859-1} \bmod 779.113 = 470.733$ (dato precalculado)

$A_q = p^{q-1} \bmod n = 859^{907-1} \bmod 779.113 = 308.381$ (dato precalculado)

dp = d mod (p-1) = 17.249 mod (859 - 1) = 89 (dato precalculado)

dq = d mod (q-1) = 17.249 mod (907 - 1) = 35 (dato precalculado)

$C_p = C \bmod p = 582.819 \bmod 859 = 417$

$C_q = C \bmod q = 582.819 \bmod 907 = 525$

Reemplazando:

$M = \{A_p[C_p^{dp} \pmod p] + A_q[C_q^{dq} \pmod q]\} \bmod n$

$M = [470.733 * 417^{89} \bmod 859] + [308.381 * 525^{35} \bmod 907] \bmod 779.113$

$M = [470.733 * 322 + 308.381 * 824] \bmod 779.113 = [151.576.026 + 254.105.944] \bmod 779.113$

$M = 405.681.970 \bmod \bmod 779.113 = 543.210$

Efectivamente:

$582.819^{17.249} \bmod 779.113 = 543.210$

$543.210^{7.481} \bmod 779.113 = 582.819$

TCR optimizado con fórmula de Garner

$$m = m_2 + h * q$$

$$h = q * Inv * [(m_1 - m_2)] \bmod p$$

En donde:

$m_1 = C^{dp} \bmod p$

$m_2 = C^{dq} \bmod q$

dp = d mod (p-1) (dato precalculado)

dq = d mod (q-1) (dato precalculado)

$qInv = q^{-1} \bmod p$ (dato precalculado)

Recordando que: C = 543.210, p = 859, q = 907 (n = 779.113), e = 7.481, d = 17.249:

$dp = d \bmod (p-1) = 7.481 \bmod 858 = 617$ (dato precalculado)

$dq = d \bmod (q-1) = 7.481 \bmod 906 = 233$ (dato precalculado)

$qInv = q^{-1} \bmod p = inv\ (907, 859) = inv\ (48, 859) = 519$ (dato precalculado)

$m_1 = C^{dp} \bmod p = 543.210^{617} \bmod 859 = 417$

$m_2 = C^{dq} \bmod q = 543.210^{233} \bmod 907 = 525$

Reemplazando:

$h = q * Inv * [(m_1 - m_2)] \bmod p$

$h = 519 * [417 - 525] \bmod 859 = -56052 \bmod 859 = 642$

$m = m_2 + h * q$

$m = 525 + 642 * 907 = 582.819$, que es el mismo valor anterior.

Firmando con RSA y comprobando la firma

Si Alicia desea enviar un mensaje M firmado a Bernardo, con su clave privada d_A va a realizar la siguiente operación $M^{dA} \bmod n_A = C$, enviando en ese criptograma C su firma.

Dado que Bernardo conocerá la clave pública de Alicia, n_A y e_A (y todos los que la conozcan) podrá realizar la siguiente operación: $C^{eA} \bmod n_A = M$.

Observa que $C^{eA} \bmod n_A = [M^{dA} \bmod n_A]^{eA} \bmod n_A = [M^{dA}]^{eA} \bmod n_A$. Como en la operación $[M^{dA}]^{eA} \bmod n_A$ los exponentes se anulan entre sí entregando el valor 1 como ya hemos visto, Bernardo recupera el mensaje M firmado por Alicia dentro del cuerpo n_A.

Como ya se ha comentado, en los sistemas de cifra asimétricos como es el caso de RSA, en la práctica solo se cifrarán números y además valores relativamente pequeños, de unas centenas de bits. Por tanto, una firma digital se realizará sobre un número, resultado de aplicar una función hash a un mensaje, ahora sí un archivo ya de cualquier tamaño y tipo.

Los pasos a realizar en la firma digital con RSA a través de un hash serán:

1. Alicia obtiene la función hash h(M) del archivo M que desea firmar.
2. Sobre ese hash h(M), Alicia realizará la firma usando su clave privada d_A.
3. Alicia calcula $h(M)^{dA} \bmod n_A = s$ y le envía a Bernardo la dupla texto en claro y firma (M, s).
4. Bernardo recibe (M', s) pues no se sabe si recibe el mismo mensaje M o un mensaje distinto o falso M'.

5. Bernardo realiza la siguiente operación con la clave pública de Alicia:
6. Calcula $s^{eA} \bmod n_A = [h(M)^{dA} \bmod n_A]^{eA} \bmod n_A = h(M)$.
7. Es decir, recupera el valor h(M) que usó Alicia para su firma en el momento de emisión.
8. Bernardo calcula la función hash h(M') del mensaje M' recibido, que puede ser el verdadero mensaje M u otro distinto M'.
9. Si el hash calculado en recepción coincide con el hash recuperado y firmado en emisión, se acepta la firma como correcta; en caso contrario se rechaza la firma.

Retomando los valores de la clave de Alicia, si desea a firmar un documento M cuyo hash h(M) es el valor 46, usará su clave privada $d_A = 77$ y su cuerpo de cifra $n_A = 119$. En recepción, Bernardo, o quien conozca la clave pública $e_A = 5$ y $n_A = 119$ de Alicia, podrán comprobar dicha firma. Se procede en este caso así:

- Alicia calcula $46^{77} \bmod 119 = 65$, que envía a Bernardo.
- Bernardo calcula $65^5 \bmod 119 = 46$, y comprueba la firma.

Tamaño de los parámetros en RSA

Como se ha comentado, se usará la misma clave pública e para todo el mundo, el número 4 de Fermat. Los motivos para usar un valor de clave pública relativamente pequeña comparada con el módulo n sigue el siguiente razonamiento:

- Como la trampa ϕ_n será igual a (p - 1)*(q - 1) y los primos p y q tienen al menos 1.024 bits, el tamaño de ϕ_n será aproximadamente igual al de n, solo un poco menor, es decir este caso 2.048 bits.
- Como la clave pública e y la clave privada d son inversas en el cuerpo ϕ_n, es decir d = inv [e, ϕ_n], entonces se cumplirá la siguiente relación: $e*d \bmod \phi_n = 1$.
- Para que se cumpla esa igualdad, el producto de e*d debe salir del cuerpo ϕ_n al menos una vez para que la operación en ese módulo nos devuelva el valor 1.
- Es decir, se cumplirá que $e*d = k*\phi_n + 1$, con k = 1, 2, 3, 4, … algún número.
- Para que ese producto salga al menos una vez del cuerpo ϕ_n, es decir con k = 1, como el valor de e es muy pequeño, de tan solo 17 bits, entonces d tendrá que ser un número muy grande, en torno al tamaño de ϕ_n, como mínimo de 2.031 bits, porque 2.031 + 17 = 2.048.
- En la práctica ese valor de k = 1, o un valor bajo de k, será muy poco probable y, por tanto, podemos esperar una clave privada d muy cercana o igual en bits al valor de n, como así sucede en la práctica. Lo típico es que para una clave de 2.048 bits el valor de la clave privada d esté entre 2.040 y 2.048 bits.

- Esto significa que será computacionalmente imposible adivinar el valor de la clave privada d, puesto que encontrar un número dentro de un cuerpo de 2.048 bits lleva un tiempo de cómputo inabordable, significando como media $2^{2.047}$ intentos.
- Luego, forzar a que la clave pública e sea un valor pequeño, menor que 20 bits dentro de un cuerpo de al menos 2.048 bits, asegura que la clave privada d será un valor muy grande y, por tanto, muy segura.

En cuanto a los valores de los primos p y q, estos pueden ser de igual tamaño, por ejemplo, de 1.024 bits cada uno para una clave de 2.048 bits, si bien podrían estar algo separados, por ejemplo un primo de 1.026 bits y el otro primo de 1.022 bits. Las aplicaciones estándar como OpenSSL generan claves RSA siempre con dos primos del mismo tamaño.

Observa que si los primos son muy cercanos, como podría ser en un caso extremo de los denominados primos gemelos o twin primes, en donde p = q + 2 (por ejemplo 1.000.037 y 1.000.039), lógicamente la seguridad del sistema es mínima porque la factorización del módulo n resulta elemental con el método de Fermat (obtener la raíz cuadrada de n y buscar primos cercanos). No obstante, la probabilidad de que al generar dos números primos de 1.024 bits aleatorios cada uno, estos sean primos gemelos es completamente despreciable.

El número 4 de Fermat como clave pública RSA

Aunque la única condición que debe cumplir la clave pública e para que pueda existir la clave privada inversa d es que exista el inverso de e en ϕ_n, es decir d = inv [e, ϕ_n], y por tanto es necesario que mcd [e, ϕ_n] = 1, dado que 1 < e < ϕ_n entonces podríamos elegir cualquier número dentro de este rango que cumpla con esa condición. No obstante, ya hemos visto que es recomendable usar un valor pequeño como clave pública. De hecho, valores típicos suelen ser los números 3, 17 y 65.537 puesto que, además, tienen solo dos bits iguales a 1 en su representación binaria, que como veremos en este apartado tendrá importancia en la eficiencia del cómputo.

Descartando el número 3 porque se han descrito ataques posibles al sistema al ser muy pequeño, la representación binaria del número 17 es 10001 y la representación binaria del número 65.537 es 10000000000000001. Este último número, conocido como número 4 de Fermat, o simplemente F_4, tiene un conjunto de características muy interesantes en criptografía, a saber:

- $65.537 = 2^{2^4} + 1 = 2^{16} + 1$, es un número primo.
- 65.537 es un número de 17 bits, relativamente pequeño.

- 65.537 = 10000000000000001 en binario, con una gran cadena de 0, y solo el bit inicial y final son 1. Al tener muchos ceros, permitirá una operación de cifra muy rápida con el algoritmo de exponenciación rápida AEE.

Si el exponente de una cifra RSA, en este caso la clave pública del receptor e_R, tiene muchos ceros como es el caso de F_4, entonces la operación de intercambio de clave se realiza de forma muy rápida por el algoritmo de exponenciación rápida que ya hemos visto. Por lo tanto, para la clave pública e = 10000000000000001, en el primer paso del algoritmo no se hace operación alguna, porque el resultado es la misma base A con la que se inicia el cálculo, luego tendríamos 15 operaciones de elevar al cuadrado y reducir módulo, que requieren muy poco tiempo de cómputo, y solamente una multiplicación al final y su reducción a módulo.

Como conclusión, el número F_4 es apropiado como clave pública universal porque tiene muchos ceros y es un valor relativamente pequeño comparado con el módulo n, que será siempre en torno a los 2.048 bits. Resumiendo, esto trae consigo las siguientes consecuencias interesantes:

- Forzar a que la clave privada d tenga un tamaño similar al módulo n y, por tanto, sea segura ante un ataque por fuerza bruta.
- Acelerar la operación de exponenciación para el intercambio de clave al tener solo dos valores binarios en 1 y los demás en 0.
- Evitar ataques que se podrían producir si e fuese demasiado pequeño, como sería el caso para e = 3.

Recuerda que el número 65.537 es la clave pública e por defecto de todas las claves RSA comerciales y que podemos ver en certificados digitales. Lo que será distinto y propio de cada clave, son los primos p y q, y lógicamente el módulo n.

Cifrando mensajes con RSA

Puesto que los sistemas de cifra con clave pública tienen una velocidad de cifra en torno a los cientos de Kilobytes por segundo, muy lenta si la comparamos con la velocidad de los sistemas de cifra con clave secreta que está en los cientos de Megabytes por segundo (mil veces más rápidos), los primeros se usarán preferentemente para cifrar números de solo unas pocas centenas de bits. Por lo tanto, aunque podemos codificar el mensaje M con la codificación que se nos ocurra (código ASCII decimal, por ejemplo) para transformar esa información a un conjunto a números y usar luego RSA para cifrar esa cadena de valores, en la práctica no tiene sentido hacerlo por esa velocidad tan baja.

No obstante, si deseamos cifrar con RSA mensajes de texto, es menester que previamente codifiquemos el texto a cifrar, por ejemplo según los valores en decimal de tres dígitos de los caracteres en código ASCII extendido. Como se ha comentado, pueden existir infinitas maneras de codificar el texto en claro, como por ejemplo dando un peso o valor a cada carácter según un alfabeto dado, modificando estos pesos, etc.

Para la siguiente clave RSA, en donde p = 5.923, q = 6.689, n = 39.618.947, con e = 31 y d = 5.110.495, vamos a cifrar el mensaje de texto CALCULANDO BLOQUES de 18 caracteres.

Como n tiene un tamaño de 26 bits y la cifra será por bytes, vamos a cifrar bloques de 3 bytes, es decir 24 bits. Por lo tanto, como el mensaje CALCULANDO BLOQUES tiene 18 bytes, vamos a cifrar estos 6 bloques de 24 bits cada uno, con sus valores binarios según el código ASCII:

M_1 = CAL 01000011 01000001 01001100 = 4.407.628
M_2 = CUL 01000011 01010101 01001100 = 4.412.748
M_3 = AND 01000001 01001110 01000100 = 4.279.876
M_4 = O B 01001111 00100000 01000010 = 5.185.602
M_5 = LOQ 01001100 01001111 01010001 = 5.001.041
M_6 = UES 01010101 01000101 01010011 = 5.588.307

Como la clave e = 31 y n = 39.618.947 los subcriptogramas C_1, C_2, C_3, C_4, C_5 y C_6 serán:

$C_1 = 4.407.628^{31} \bmod 39.618.947 = 6\,734.280$
$C_2 = 4.412.748^{31} \bmod 39.618.947 = 21.898.080$
$C_3 = 4.279.876^{31} \bmod 39.618.947 = 33.215.194$
$C_4 = 5.185.602^{31} \bmod 39.618.947 = 30.956.224$
$C_5 = 5.001.041^{31} \bmod 39.618.947 = 8.150.418$
$C_6 = 5.588.307^{31} \bmod 39.618.947 = 22.796.302$

Y como la clave privada es d = 5.110.495, desciframos esos seis bloques como sigue:

$6.734.280^{5.110.495} \bmod 39.618.947 = 4.407.628 = M_1$
$21.898.080^{5.110.495} \bmod 39.618.947 = 4.412.748 = M_2$
$33.215.194^{5.110.495} \bmod 39.618.947 = 4.279.876 = M_3$
$30.956.224^{5.110.495} \bmod 39.618.947 = 5.185.602 = M_4$
$8.150.418^{5.110.495} \bmod 39.618.947 = 5.001.041 = M_5$
$22.796.302^{5.110.495} \bmod 39.618.947 = 5.588.307 = M_6$

Como los valores recuperados en el descifrado M_1, M_2, M_3, M_4, M_5, M_6 son los mismos que los bloques a cifrar, hemos recuperado el mensaje M = CALCULANDO BLOQUES. Simplemente pasamos a binario el número recuperado en el descifrado en bloques de 24 bits, añadiendo ceros a la izquierda en cada bloque si hace falta, y vamos leyendo ahora esos 24 bits de derecha a izquierda en bloques de 8 bits, un byte, y convirtiéndolo a carácter ASCII.

En este caso, la longitud del mensaje era congruente con la longitud del bloque. Si no fuese así, por ejemplo, si el texto en claro terminase con un punto y aparte (.), no se añadiría aquí relleno.

Simplemente habría un séptimo bloque a cifrar con un único carácter, el punto, cuyo ASCII binario es 0010 1110 y en decimal es 46. Es decir, se añadiría:

$M_7 = 46$
$C_7 = 46^{31} \bmod 39.618.947 = 5.740.401$

Que se descifrará así:

$5.740.401^{5.110.495} \bmod 39.618.947 = 46 = M_7$

Claves privadas y públicas parejas

Cuando generamos una clave RSA, usamos como trampa el indicador de Euler ϕ_n para calcular la clave privada d a partir del conocimiento de la clave pública e. Puesto que mcd [e, ϕ_n] = 1, se asegura que el inverso d existe y que, además, es el único inverso de la clave pública e en ese cuerpo ϕ_n.

No obstante, como es lógico, la cifra se hace posteriormente en el cuerpo público n (no en ϕ_n) para que todo el mundo pueda utilizarlo. Y en dicho cuerpo n ya no se cumple que el único inverso de la clave pública e sea la clave privada d. Hay al menos otro valor diferente a la clave privada d que cumple las mismas funciones y que, por lo tanto, permite descifrar algo enviado de forma secreta que se ha cifrado con la clave pública inversa, o bien firmar digitalmente con esa otra clave privada. A estas claves se les llama claves privadas parejas (CPP).

Esto es algo en principio inesperado porque normalmente siempre se ha dicho que un sistema de cifra asimétrico, como lo es RSA, tiene una única clave pública y, por lo tanto, también una única clave privada. Esto

no será verdadero. Vamos a explicarlo con un sencillo ejemplo, que puede comprobar fácilmente usando calculadora de Windows.

Sea una clave RSA con p = 37, q = 41, n = 1.517, ϕ_n = 1.440, e = 13, d = 997. Vamos a cifrar el valor 1.001 con la clave pública e = 13 y luego descifrar el criptograma con la clave privada d = 997.

- Cifrando con e = 13 $1.001^{13} \bmod 1.517 = 1.088$
- Descifrando con d = 997 $1.088^{997} \bmod 1.517 = 1.001$

Y se ha recuperado el secreto.

Pero si usamos los números 277, 637 y 1.357 como si fuesen la clave privada d, obtenemos lo siguiente:

- Descifrando con d' = 277 $1.088^{277} \bmod 1.517 = 1.001$
- Descifrando con d' = 637 $1.088^{637} \bmod 1.517 = 1.001$
- Descifrando con d' = 1.357 $1.088^{1.357} \bmod 1.517 = 1.001$

Y se ha recuperado también el texto en claro o secreto.

Lo que ha sucedido en el ejemplo anterior es que en el cuerpo de cifra n = 1.517 con una clave pública e = 13, los valores 277, 637 y 1.357 son claves privadas parejas (CPP) denominadas d' y que cumplen la misma función que la clave privada d = 997.

Toda clave RSA tendrá como mínimo una clave privada pareja. La cantidad de claves privadas parejas dependerá fuertemente de los primos p y q, siendo las ecuaciones que nos entregan este valor de CPP las siguientes.

$$\gamma = mcm\ [(p-1), (q-1)]$$
$$d_\gamma = e^{-1} \bmod \gamma = inv\ (e, \gamma)$$

Entonces la clave privada d tendrá λ claves parejas d_i de la forma:

$$d_i = d_\gamma + i\,\gamma \quad con\ 1 < d_i < n \quad Con\ i = 0, 1, \ldots \lambda$$
$$\lambda = \lfloor (n - d_\gamma)/\gamma \rfloor$$

Así, para la clave anterior con p = 37, q = 41, n = 1.517, ϕ_n = 1.440, e = 13, d = 997, tenemos:

$$\gamma = \text{mcm}\,[(p-1), (q-1)] = \text{mcm}\,(36, 40) = 360$$

$$d_\gamma = \text{inv}\,(13, 360) = 277$$

$$d_i = d_\gamma + i\,\gamma = 277 + i*360\ (i = 0, 1, 2, 3)$$

$$d_i = 277, 637, 997, 1.357$$

$$\lambda = \lfloor (n - d_\gamma)/\gamma \rfloor = \lfloor (1.517 - 277)/360) \rfloor = 3{,}44 = 3\ \text{CPP}$$

Observa que las claves están separadas por un valor constante γ = 360. Observa además que una de las claves que nos entrega la Ecuación $d_i = d_\gamma + i\,\gamma$ es precisamente la clave privada d; en este caso 997.

Al igual que existe una cantidad λ de claves privadas parejas, existirá un número similar λ de claves públicas parejas. En este caso, las ecuaciones serán muy similares, cambiando solo e por d:

Si γ = mcm [(p-1), (q-1)] y además $e_\gamma = d^{-1} \bmod \gamma = \text{inv}\,(d, \gamma)$, entonces la clave pública e tendrá λ claves parejas e_i de la forma $e_i = e_\gamma + i\,\gamma$, con $1 < e_i < n$, con $i = 0, 1, \ldots \lambda$, siendo:

$$\lambda = \lfloor (n - e_\gamma)/\gamma \rfloor$$

Por ejemplo, para la misma clave anterior con p = 37, q = 41 y 13, d = 997, tendremos:

$$\gamma = \text{mcm}\,[(p-1), (q-1)] = \text{mcm}\,(36, 40) = 360$$

Luego:

$$e_\gamma = \text{inv}\,(997, 360) = \text{inv}\,(277, 360) = 13$$

$$e_i = e_\gamma + i\,\gamma = 13 + i*360\ (i = 0, 1, 2, 3, 4)$$

Es decir, d_i = 13, 373, 733, 1 093, 1.453

Además, $\lambda = \lfloor (n - e_\gamma)/\gamma \rfloor = \lfloor (1.517 - 13)/360) \rfloor = 4{,}17 = 4\ \text{CP}_{\text{úb}}\text{P}_{\text{arejas}}$

A primera vista, no resulta agradable saber que hay otras claves en RSA y que realizan la misma función que una clave privada o una clave pública, más aún si se tiene en mente lo que se dice habitualmente de este sistema de cifra asimétrico: que existe una única clave pública (e, n) y una única clave privada (d), y lo que se

cifra con una de ellas solo se descifrará con la otra. De hecho, todos los certificados digitales X.509 que trabajan con RSA tienen este tipo de claves.

Hemos visto además que el cálculo de estas claves privadas parejas es muy sencillo, que están separadas por un valor constante y que incluso para claves con valores reales de miles de bits se obtienen sus valores muy rápidamente porque su cálculo es elemental.

La verdad es que no hay de qué alarmarse puesto que debido a los valores estándar que se usan de p, q y e en la generación de claves reales, aunque en ciertos casos existan muchas CPP, sus valores serán todos muy próximos al valor del módulo n y, por lo tanto, será imposible adivinarlos o intentar romperlos por fuerza bruta.

¿Qué pasará ahora con las claves públicas parejas?

La situación en este caso es muy similar, las claves públicas parejas tendrán valores muy altos. Además, la clave es por definición pública y no tiene ningún secreto. No obstante, existe un escenario de firma digital como el que se comenta a continuación que nos puede dejar ciertas dudas. Supongamos que un usuario firma el hash de un mensaje con su clave privada, lo que envía a destino junto con el mensaje original pues no es necesario agregarle confidencialidad. En destino se descifrará el criptograma con la clave pública del emisor, recuperando por lo tanto el hash firmado en emisión. Se calcula el hash del mensaje recibido y se comparan los dos hashes, el de emisión y el recuperado en recepción, para dar validez a dicha firma.

Hasta aquí todo es correcto, pero, ¿qué pasaría ahora si mediante algún artilugio alguien puede comprobar la firma usando una clave pública pareja distinta a la clave pública e conocida por todo el mundo? Esto podría interpretarse como una muestra de debilidad del sistema, en tanto nadie conoce la existencia de estas claves públicas parejas y consecuentemente podrían, incluso, hacer dudar de las bondades del algoritmo. Pero será igual de difícil adivinar una clave pública pareja al ser por lo general todas (excepto el valor de e) muy grandes y muy cercanas al módulo n.

Para minimizar las claves parejas y reducirlas a la unidad o muy cerca de la unidad, habrá que usar primos seguros. Se dice que p es un primo seguro si p = 2*q +1 siendo q primo. Por ejemplo, el 83 es un primo seguro porque 83 = 2*41 + 1, donde 41 también es un primo. No obstante, aunque no se usen primos seguros y la clave en cuestión presente varias CPP, se concluye que las claves privadas y públicas parejas no son una vulnerabilidad del sistema RSA, es tan solo una curiosidad. Por esto mismo, programas tan populares como OpenSSL no tienen en cuenta esto y no usan primos seguros al generar claves RSA.

Pero no todo se queda aquí. En algunos casos, en una clave RSA aparecerán además otros números que, ni siendo la clave privada d ni ninguna de las posibles claves privadas parejas d', también descifran lo que se ha cifrado con la clave pública e.

No profundizaremos en este tema (se hará en el libro a publicarse en 2019) porque habría que abordar con mayor profundidad el concepto de los anillos en un cuerpo, que no obstante aparecerán más adelante en los ataques a RSA.

Solo como ejemplo ilustrativo y para que pienses en este interesante tema, considera la siguiente clave RSA con los estos datos:

$$p = 7,\ q = 11,\ e = 17,\ \phi_n = 60,\ d = 53$$

Aplicando las ecuaciones anteriores se tiene:

$$\gamma = \text{mcm}\ [(p-1), (q-1)] = \text{mcm}\ (6, 10) = 30$$

$$d_\gamma = \text{inv}\ (17, 30) = 23$$

$$d_i = d_\gamma + i\,\gamma = 23 + i*30\ (i = 0, 1)$$

$$d_i = 23, 53$$

$$\lambda = \lfloor (n - d_\gamma)/\gamma \rfloor = \lfloor (77 - 23)/30) \rfloor = 1{,}8 = 1\ \text{CPP}$$

Si ciframos N = 13 obtenemos $13^{17} \bmod 77 = 62$, y desciframos con d y d':

$$62^{53} \bmod 77 = 62^{23} \bmod 77 = 13.$$

Hasta aquí todo correcto. Pero si ahora desciframos el criptograma 62 con los números 3, 13, 33, 43, 63 y 73 recuperamos también el secreto 13:

$$62^3 \bmod 77 = 62^{13} \bmod 77 = 62^{33} \bmod 77 = 62^{43} \bmod 77 = 62^{63} \bmod 77 = 62^{73} \bmod 77 = 13.$$

No ocurrirá lo mismo si ahora ciframos otro secreto, por ejemplo el 31.

¿Tienen algo que ver entre sí los números que han entrado en juego en ejemplo anterior, el primo p = 7, el primo q = 11, el módulo de cifra n = 77 y la clave pública e = 17, para que hubiese 8 claves válidas de descifrado separadas exactamente por 10 espacios?

Números no cifrables

Puesto que la operación de cifrado RSA de un número N es N^{clave} mod n, donde clave puede ser la clave pública de destino o la clave privada de emisión, según el tipo de cifra que se desee realizar, y siendo los elementos N a cifrar todos los valores del cuerpo n, que van desde 0 hasta n-1, existirán algunos números N_i que aunque se cifren irán en claro y se denominarán Números No Cifrables NNC. Como es evidente, el valor de la clave en ese exponente puede ser cualquier número que cumpla con los requisitos de diseño RSA.

Dos casos muy obvios de números que si se cifran van en claro serán el 0 y 1, puesto que tanto 0^{clave} mod n = 0 como 1^{clave} mod n = 1. Otro valor, ahora no tan obvio pero que puede demostrarse matemáticamente que también se transmite en claro o no se cifra, es el número n-1 puesto que $n\text{-}1^{clave}$ mod n = n-1. Por ejemplo, para la clave RSA con n = 323 (p = 17, q = 19) y e = 11, se obtiene:

$$0^{11} \bmod 323 = 0$$
$$1^{11} \bmod 323 = 1$$
$$322^{11} \bmod 323 = 322$$

Además de esos tres valores, en RSA existirán siempre como mínimo otros 6 valores que irán en claro. Por tanto, cualquier clave RSA tendrá como mínimo 9 números no cifrables.

En el ejemplo anterior, los otros seis valores que se transmiten en claro son:

$$18^{11} \bmod 323 = 18$$
$$152^{11} \bmod 323 = 152$$
$$153^{11} \bmod 323 = 153$$
$$170^{11} \bmod 323 = 170$$
$$171^{11} \bmod 323 = 171$$
$$305^{11} \bmod 323 = 305$$

A diferencia de las claves privadas y públicas parejas, cuyos valores se obtenían de forma inmediata mediante una ecuación en la que existía una separación constante entre dichas claves, en este caso el cálculo de los números no cifrables NNC es más laborioso porque habrá que hacer un ataque por fuerza bruta en el espacio de los primos p y q, para comprobar qué valores de N nos entregan las siguientes igualdades:

$$N^e \bmod p = N \text{ para } 1 < N < p\text{-}1$$
$$N^e \bmod q = N \text{ para } 1 < N < q\text{-}1$$

Como es de esperar, para claves reales de 1.024 o más bits, resulta computacionalmente imposible realizar esos cálculos dentro de los primos p y q de 512 bits cada uno, pues habrá que hacer 2^{512} operaciones en p y otras 2^{512} operaciones en para recorrer todos esos posibles valores de N. Por lo tanto, para claves reales, excepto los valores 0, 1 y n-1, no será posible conocer los demás números no cifrables.

A continuación, se indican las ecuaciones que intervienen en el cálculo de estos números no cifrables NNC. La única dificultad de cálculo se encuentra en las dos últimas ecuaciones, que significan aplicar fuerza bruta con todos los valores de N candidatos a ser número no cifrable en p y en q.

La cantidad de números no cifrables dentro de un cuerpo de cifra n será:

$$\sigma_n = [1 + mcd\ (e-1, p-1)][1 + mcd\ (e-1, q-1)]$$

Y los valores de esos números no cifrables serán:

$$N = [q\{inv\ (q, p)\}N_p + p\{inv\ (p, q)\}N_q] \bmod n$$

Con: N_p las soluciones de $N^e \bmod p = N$

N_q las soluciones de $N^e \bmod q = N$

Por ejemplo, sea p = 13; q = 17; n = 221; $\phi_n = 192$, e = 7; d = inv (7, 192) = 55. Vamos a encontrar los números no cifrables de esta clave RSA. La cantidad de números no cifrables σ_n de esta clave será igual a 21 y así se encuentran sus valores:

$$\sigma_n = [1 + mcd\ (e-1, p-1)][1 + mcd\ (e-1, q-1)]$$

$$\sigma_{221} = [1 + mcd\ (6, 12)][1 + mcd\ (6, 16)] = (1+6)(1+2) = 21$$

Los números no cifrables serán:

$$NNC = [q\{inv\ (q, p)\}Np + p\{inv\ (p, q)\}Nq] \bmod n$$

$$NNC = [17\{inv\ (17, 13)\}Np + 13\{inv\ (13, 17)\}Nq] \bmod 221$$

$$inv\ (17, 13) = inv\ (4, 13) = 10$$

$$inv\ (13, 17) = 4$$

$$NNC = [\{17*10\}Np + \{13*4\}Nq] \bmod 221$$

$$NNC = [170*Np + 52*Nq] \bmod 221$$

Por fuerza bruta encontramos las soluciones de $Np = N^7 \bmod 13 = N$ para $1 < N < 12$: Np = {0, 1, 3, 4, 9, 10, 12}

Por fuerza bruta encontramos las soluciones de $Nq = N^7 \bmod 17 = N$ para $1 < N < 16$: Nq = {0, 1, 16}

Luego:

NNC = [170*{0, 1, 3, 4, 9, 10, 12} + 52*{0, 1, 16}] mod 221
NNC = [{0, 170, 510, 680, 1 530, 1.700, 2 040} + {0, 52, 832}] mod 221
NNC = [0+0, 0+52, 0+832, 170+0, 170+52, 170+832,.1 700+0, 1 700+52, 1 700+832, 2 040+0, 2 040+52, 2 040+832] mod 221
NNC = [0, 52, 832, 170, 222, 1 002, 510, 562, 1 342, 680, 732, 1 512, 1 530, 1 582, 2 362, 1 700, 1 752, 2 531, 2 040, 2 092, 2 872] mod 221
Reduciendo cada uno de esos 21 números módulo 221 obtenemos:
NNC = [0, 52, 169, 170, 1, 118, 68, 120, 16, 17, 69, 186, 204, 35, 152, 153, 205, 101, 51, 103, 220]
Y ordenando ahora de menor a mayor se obtiene:
NNC = [0, 1, 16, 17, 35, 51, 52, 68, 69, 101, 103, 118, 120, 152, 153, 169, 170, 186, 204, 205, 220]

De los resultados del ejercicio anterior podemos observar dos características muy interesantes de los números no cifrables en RSA.

- Que aparecen varios números consecutivos, ej. (0, 1), (16, 17), (51, 52), (68, 69), (152, 153), (169, 170), (204, 205), una característica que se repite con todas las claves RSA.
- Que excepto el valor 0, los valores de los extremos siempre sumarán el valor del módulo n, en este caso: n = 221 = 1+220 = 16+205 = 17+204 = 35+186 = 51+170 = 52+169 = 68+169 = 69+153 = 101+120 = 103+118.

A pesar de este comportamiento tan peculiar, ninguna de estas características puede interpretarse, al menos en principio, como una vulnerabilidad.

Para minimizar la cantidad de números no cifrables y reducirlos a su valor mínimo 9, también habrá que usar primos seguros. Así, una clave RSA que tenga una única clave privada pareja y nueve números no cifrables, se conocerá como clave óptima.

Una vez conocidas las condiciones de diseño para que una clave RSA tenga la menor cantidad posible de números no cifrables, la pregunta que nos acecha es, ¿cuál será la cantidad mayor de estos NNC?

El peor de los escenarios será cuando:

$$mcd\ (e - 1, p - 1) = p - 1$$
$$mcd\ (e - 1, q - 1) = q - 1$$

En estas condiciones $\sigma_n = [1 + mcd\ (e - 1, p - 1)]\ [1 + mcd\ (e - 1, q - 1)] = [1 + (p -1)]\ [1 + (q - 1)] = p*q = n$. Esto quiere decir que todos los elementos del cuerpo n = p*q irían en claro. Por lo tanto, una vez elegidos p y q, hay que tener especial cuidado en la elección de la clave pública e, como lo veremos a continuación.

Si en la clave del ejemplo anterior (p = 13, q = 17, ϕn = 192, n = 221) cambiamos la clave púbica e = 7 por la clave pública e = 11, obtenemos una clave con 9 números no cifrables, el mínimo. Pero si, por el contrario, hubiésemos elegido como clave pública e = 49, igual de válida que 11 y que 7, observamos ahora lo siguiente:

$$mcd\ (e - 1, p - 1) = mcd\ (48, 12) = 12$$
$$mcd\ (e - 1, q - 1) = mcd\ (48, 16) = 16$$
$$\sigma_{221} = [1 + 12][1 + 16] = 13*17 = 221 = n$$

Es decir, cualquier número que se cifre irá en claro.

¿Por qué ha pasado esto? En el ejemplo anterior, observamos que e = 49 = 192/4 +1 = $\phi n/4 + 1$. En general, si elegimos un valor de clave pública e = $\phi n/k + 1$ para valores de k igual a 2, 3, 4, etc., siempre con k pequeño, la cantidad de NNC puede ser muy alta, incluso todo el cuerpo como acaba de suceder para e = 49, una situación que se va a repetir para esta clave RSA con e = 97.

Si $\sigma_n = n$, como todo el cuerpo de cifra irá en claro, entonces el valor de la clave pública se repetirá en la clave privada o en una clave privada pareja. Esto sucede cuando e = $\phi n/2 + 1$.

Por ejemplo, la clave RSA con p = 199, q = 239, n = 47.561, ϕn = 47.124, e = $\phi n/2 + 1$ = 23.563, tiene todos sus elementos no cifrables y la clave privada d = 23.563, igual que la pública. Y si p = 60.589, q = 51.829, con lo que n = 3.140.267.281, ϕn = 3.140.154.864, con una clave pública e = $\phi n/2 + 1$ = 1.570.077.433, sucede lo mismo, que $\sigma_n = n$ y que e = d.

En condiciones reales, en las que la clave pública e es el número 4 de Fermat, como n y ϕ_n serán números de al menos 1.024 bits, entonces en el supuesto caso de que se cumpliese la relación e = ϕ(n)/k + 1, como el valor de k sería tan grande, la cantidad de números no cifrables podría ser alta pero nunca se llegará a estos extremos.

El caso que más nos preocupa en este apartado de valores no cifrables es que un número secreto que cifremos con RSA para un intercambio de clave, al final vaya en claro. Pero los números que cifraremos en la práctica pueden ser de estos dos tipos:

a) Una clave de sesión para su uso en un algoritmo simétrico y que estará entre los 128 y 256 bits, en lo que se conoce como intercambio de clave con RSA usando la clave pública del receptor.

b) Un número resultado de aplicar a un documento una función hash de 160 a 256 bits, en lo que se conoce como firma digital RSA usando la clave privada del emisor.

En ambos casos son números de tan solo unas centenas de bits dentro de un cuerpo de cifra que actualmente es de 2.048 bits. Por lo tanto, la probabilidad de que dentro de un cuerpo n de 2.048 bits, un conjunto de números secretos caiga precisamente en la zona de centenas de bits, es decir, sea un número binario con una inmensa cantidad de ceros a la izquierda y con solo 100 o 200 bits significativos, es totalmente despreciable. Pero, así y todo, si aceptamos que hay algunos números de ese conjunto no cifrable de la clave RSA que tienen solo una o dos centenas de bits significativos, la probabilidad ahora de que, además, alguno de ellos sea precisamente el número secreto de n bits que hemos elegido para ese intercambio de clave K, es solo de $1/2^n$.

Como conclusión, vemos también que para valores de diseño estándar de una clave RSA con módulo sobre los mil bits, tampoco será un motivo de preocupación la existencia de una gran cantidad de números no cifrables, Es decir, al igual que sucedía con las claves privadas parejas, tampoco estamos frente a una vulnerabilidad de RSA.

Ataques a RSA

La seguridad del algoritmo RSA puede atacarse de varias maneras. En este libro veremos estas tres:

- Ataque por la factorización del módulo o cuerpo de cifra.
- Ataque basado en la paradoja del cumpleaños.
- Ataque basado en el cifrado cíclico.

Los ataques por factorización y paradoja del cumpleaños permitirían romper RSA al encontrar la clave privada, no aplicando fuerza bruta.

En el primer caso, al factorizar el módulo n y conocer así los primos p y q, se podrá calcular la clave privada d simplemente usando el algoritmo extendido de Euclides con la clave pública e y el indicador de Euler ϕ_n ahora ya conocido por el atacante.

En el caso del ataque basado en la paradoja del cumpleaños, se encontrará directamente el valor de la clave privada. Lo interesante de este ataque es que permite paralelizar la acción y aplicar así el principio de "divide y vencerás", igual como sucedía con los algoritmos de cifra simétrica.

Por su parte, un ataque basado en el cifrado cíclico, permitirá encontrar el mensaje secreto cifrado con RSA, sin necesidad de conocer la clave privada del destinatario o víctima, pero a diferencia de los otros dos no se romperá el sistema de cifra.

Hay que tener en cuenta que para las claves RSA actuales, con un cuerpo de cifra de al menos 1.024 bits, ninguno de estos tres ataques son viables con los computadores que hoy conocemos y usamos.

Ataque por factorización

Los algoritmos de factorización pueden dividirse en dos grupos, los denominados de propósito general y los de propósito específico. Independientemente de ello, entre los más conocidos se encuentran:

- Método de factorización directa o criba de Eratóstenes
- Método de Fermat
- Método de Euler
- Método de Dixon
- Método de Williams p+1
- Método de Pollar rho
- Método de Pollar p-1
- Método de las fracciones continuas
- Método de las curvas elípticas
- Método de la criba numérica

El mejor algoritmo conocido a la fecha es el de la criba numérica o *general number field sieve* (GNFS), que tiene asociada una complejidad representada en la siguiente expresión para un número de b bits:

$$O\left(\exp\left(\left(\tfrac{64}{9}b\right)^{\frac{1}{3}}(\log b)^{\frac{2}{3}}\right)\right)$$

En donde, el factor b que representa en bits el tamaño del número a factorizar, se encuentra como un elemento del exponente y por ello el carácter exponencial de este problema.

El problema de la factorización entera es uno de esos problemas en matemáticas clasificados como abiertos; es decir, siempre puede aparecer un nuevo algoritmo que mejore las prestaciones de los anteriores o bien que presente ciertas ventajas frente a aquellos en casos especiales. Lo que es cierto es que a fecha de hoy (2018) nadie ha logrado quitarle esa característica de problema NP o comportamiento exponencial.

Desde 1991 RSA Laboratories propuso una serie de desafíos de factorización de diferentes módulos RSA con premios en metálico, pero en el año 2007 da por terminado ese desafío, poco después de que en el año 2005 se factorizasen números de 193 dígitos (640 bits) y 200 dígitos (663 bits). Ya fuera del concurso con estos premios, el 12 diciembre de 2009 un equipo de seis instituciones de investigación lideradas por Thorsten Kleinjung logran factorizar el mayor número RSA hasta el momento: 768 bits.

Ataque por paradoja del cumpleaños

En el año 1981 Ralph Merkle y Martin Hellman proponen un método para atacar al algoritmo DES, *Data Encryption Standard*, en su modalidad de doble cifrado con texto en claro y criptograma conocidos, basado en la paradoja del cumpleaños y que deriva finalmente en lo que conocemos como ataque por encuentro a medio camino, *meet in the middle*, que no hace aconsejable este tipo de cifrado doble.

Este mismo principio permitirá encontrar colisiones en las cifras realizadas con RSA. El ataque tiene la particularidad de que al prosperar tras observarse una colisión, nos entregará la clave privada de la víctima, una clave privada pareja, y en muy pocos casos un falso positivo.

El ataque sigue estos pasos:

1. Se conoce el módulo n y la clave pública e de la víctima.
2. Se elige un valor o número N de ataque aleatorio, siendo recomendable que sea el número 2 por la rapidez en el cálculo.

3. Se divide el número del módulo n en dos mitades, una parte izquierda que denominaremos i y otra parte derecha que denominaremos j.
4. La zona izquierda realizará cifrados del valor N para exponentes i en el intervalo $0 \leq i \leq n/2$ en módulo n, guardando sus resultados.
5. La zona derecha realizará cifrados del valor N para exponentes j en el intervalo $n/2 + 1 \leq j \leq n-1$ en módulo n, guardando sus resultados.
6. Para i = i+1 y para j = j+1 se calcula $N^i \bmod n$ y $N^j \bmod n$.
7. Se repite esta operación hasta que un resultado de N_j colisiona con el primer resultado de N_i o bien un resultado de N_i colisiona con el primer resultado de N_j.
8. Encontrada la colisión, el algoritmo llega a su fin y puede encontrarse la clave buscada mediante la resolución de un conjunto simple de ecuaciones.

Por ejemplo, sea p = 7; q = 17, n = 119, e = 11. Aunque la clave privada sea d = 35 y haya una clave privada pareja 83, el atacante solo conoce los valores públicos de la víctima, es decir n = 119 y e = 11. El atacante usará N = 2 y los valores iniciales de los contadores serán i = 0 y j = 59. Realizando las operaciones, se tiene:

i = 0 $2^0 \bmod 119 = 1$	j = 59 $2^{59} \bmod 119 =$ **25**
i = 1 $2^1 \bmod 119 = 2$	j = 60 $2^{60} \bmod 119 = 50$
i = 2 $2^2 \bmod 119 = 4$	j = 61 $2^{61} \bmod 119 = 100$
i = 3 $2^3 \bmod 119 = 8$	j = 62 $2^{62} \bmod 119 = 81$
i = 4 $2^4 \bmod 119 = 16$	j = 63 $2^{63} \bmod 119 = 43$
i = 5 $2^5 \bmod 119 = 32$	j = 64 $2^{64} \bmod 119 = 86$
i = 6 $2^6 \bmod 119 = 64$	j = 65 $2^{65} \bmod 119 = 53$
i = 7 $2^7 \bmod 119 = 9$	j = 66 $2^{66} \bmod 119 = 106$
i = 8 $2^8 \bmod 119 = 18$	j = 67 $2^{67} \bmod 119 = 93$
i = 9 $2^9 \bmod 119 = 36$	j = 68 $2^{68} \bmod 119 = 67$
i = 10 $2^{10} \bmod 119 = 72$	j = 69 $2^{69} \bmod 119 = 15$
i = 11 $2^{11} \bmod 119 =$ **25**	j = 70 $2^{70} \bmod 119 = 30$

Figura 5.5. Ataque a RSA por paradoja del cumpleaños: colisión $N_{j=59} = N_{i=11} = 25$.

Como colisiona el último cifrado en i=11 con el primer cifrado en j=59 con el valor 25, el atacante calcula:

$$w = (i - j) / mcd(e, |i-j|)$$

Entonces deberán existir valores s, t de forma que se cumpla lo siguiente:

$$w*s + e*t = 1$$

Las posibles soluciones a estas ecuaciones son:

$$w*s \bmod e = 1$$
$$e*t \bmod w = 1$$

En nuestro ejemplo, i = 11, j = 59, e = 11, por lo tanto:

$$w = (11-59) / mcd\ (11, |11-59|) = -48 / mcd\ (11, 48) = -\ 48$$
$$-48*s + 11*t = 1$$

Por lo tanto:

$$-48*s \bmod 11 = 1, \text{ por lo tanto } s = inv\ (-48, 11) = inv\ (7, 11) = 8$$
$$11*t \bmod 48 = 1, \text{ por lo tanto } t = inv\ (11, 48) = 35$$

El valor 35 es la clave privada buscada.

A igual resultado se llega, pero con una colisión distinta, ahora el último cifrado en j colisiona con el primer cifrado en i, si los valores de los contadores son i = 0 y j = 47. Además, la cantidad de pasos realizados en el ataque han sido solo dos como se muestra en la figura 5.6. Más aún, si hubiésemos elegido j = 48, habríamos dado con la solución en el primer intento.

i = 0 $2^0 \bmod 119 = \mathbf{1}$	j = 47 $2^{47} \bmod 119 = 60$
i = 1 $2^1 \bmod 119 = 2$	j = 48 $2^{48} \bmod 119 = \mathbf{1}$

Figura 5.6. Ataque a RSA por paradoja del cumpleaños: colisión $N_{i=0} = N_{j=48}$.

Como la coincidencia del resultado 1 se encuentra para i = 0 y para j = 48, el atacante calcula w = (0-48) / mcd (11, |0-48|) = -48 / mcd (11, 48) = - 48 y las ecuaciones son las mismas.

Ataque por cifrado cíclico

Se supone que un mensaje secreto cifrado con la clave pública del destino, típica operación en un intercambio de clave, solo podía descifrarse y recuperar ese secreto usando la clave privada de ese destino, es decir la clave inversa de la usada en el proceso de cifra, o bien con alguna clave privada pareja como ya sabemos.

Pero también será posible romper ese secreto usando solamente las claves públicas de la víctima, algo que estaría en contra de toda la lógica de los sistemas de cifra asimétrica o de clave pública. El problema que se nos presenta es que entonces cualquiera que conozca las claves públicas usadas en el proceso de intercambio de clave, podría en teoría recuperar el secreto.

Se trata de lo siguiente. Como $C = N^e \bmod n$, con N un valor secreto, vamos a realizar cifrados sucesivos de los criptogramas C_i resultantes con la misma clave pública e y en el mismo módulo. Si en uno de estos cifrados obtenemos nuevamente el cifrado C original con el que se ha iniciado el ataque, resulta obvio que el valor del paso anterior será el secreto N buscado. Esto se debe a que en RSA se generan anillos, con longitudes distintas y en los que se encuentran un conjunto de números de ese módulo, sin repetirse. En el cifrado cíclico los números de todos esos anillos forman el cuerpo de cifra o módulo n.

Por tanto, conocido o capturado el criptograma C, realizaremos los siguientes cifrados:

$$C_i = C^{ei-1} \bmod n; \text{ para } i = 1, 2,... \text{ con } C_0 = C$$

Si en el cifrado iésimo se encuentra el criptograma C inicial, entonces el cifrado anterior (i-1) será el número secreto, como se comprueba en el siguiente ejercicio.

Sea p = 37, q = 61, n = 2.257, e = 7 (d = 1.543). El valor secreto es N = 50, que se cifra como $50^7 \bmod 2.257$, dando como criptograma C = 1.475. Así, el atacante, que solo conoce la clave pública n = 2.257 y e = 7, además del criptograma capturado C = 1.475, hará estos cálculos:

$$\mathbf{1.475}^7 \bmod 2.257 = 1.758$$

$$1.758^7 \bmod 2.257 = 2.207$$

$$2.207^7 \bmod 2.257 = 782$$

$$782^7 \bmod 2.257 = 499$$

$$499^7 \bmod 2.257 = 50$$

El atacante todavía no conoce el secreto

$$50^7 \bmod 2.257 = \mathbf{1.475}$$

El atacante deduce que el secreto era 50

En este tipo de ataques por cifrado cíclico se manifiestan como hemos visto los anillos en un cuerpo, no de la misma manera que hemos visto al calcular las claves privadas y públicas parejas. Su estudio así como ejemplos prácticos con un software adecuado, se dejan para el libro de próxima publicación

5.5. El algoritmo de Elgamal

En 1985 el investigador egipcio Taher Elgamal propone sendos algoritmos de cifra y firma digital que llevan su nombre. Una de sus características es que se envían dos valores dentro del cuerpo de cifra, un primo muy grande. Si bien el algoritmo es seguro, no logra desbancar a RSA como estándar de cifra, aunque una variación de su algoritmo de firma conocido como DSS es actualmente un estándar de firma digital por el NIST.

Como el algoritmo de Elgamal se basa en el problema del logaritmo discreto, cada usuario elegirá como cuerpo de cifra un número primo p grande y una raíz primitiva α de ese primo, ambos valores públicos.

Hecho esto, se elige como clave privada un número aleatorio λ dentro de p, y se calcula la clave pública como $\beta = \alpha^{\lambda} \bmod p$; es decir:

$$\text{Clave privada: } \lambda$$
$$\text{Clave pública: } \beta = \alpha^{\lambda} \bmod p$$

Al igual que en Diffie y Hellman, la seguridad del sistema reside en que para descubrir la clave privada λ, conocidos los valores públicos p, α y β, el atacante deberá enfrentarse al problema del logaritmo discreto en un primo grande, algo computacionalmente intratable para valores cercanos a los mil bits con la capacidad de cómputo actual. En lo que sigue, el valor genérico λ de la clave privada se representará con la letra a para Alicia, con la letra b para Bernardo, etc.

Para cifrar o enviar un número N con confidencialidad de Alicia a Bernardo, se procede de la siguiente manera:

Emisión: Cifrado de Alicia

1. Alicia conoce la clave pública de Bernardo: p_B, α_B, β_B
2. Alicia genera un número aleatorio ν de sesión y calcula: $\alpha_B^{\nu} \bmod p_B$
3. Con la clave pública β_B de Bernardo Alicia calcula: $N*\beta_B^{\nu} \bmod p_B$
4. Alicia envía a Bernardo el par de números $\{\alpha_B^{\nu} \bmod p_B, N*\beta_B^{\nu} \bmod p_B\}$

Recepción: Descifrado de Bernardo

1. Bernardo con su clave privaba b calcula: $(\alpha_B^{\nu})^b \bmod p_B$ (Ec.1)
2. Como $\beta_B = \alpha_B^b \bmod p_B$, entonces: $N*\beta_B^{\nu} \bmod p_B = N*(\alpha_B^b)^{\nu} \bmod p_B$ (Ec.2)
3. Como $(\alpha_B^{\nu})^b \bmod p_B = (\alpha_B^b)^{\nu} \bmod p_B$ (Ec.3)
4. Bernardo debería calcular (Ec.2) / (Ec.3) para encontrar N

5. Como eso está prohibido, Bernardo calcula $N*(\alpha_B{}^b)^\nu * inv[(\alpha_B{}^\nu)^b, p_B] = N$
6. Es decir, calcula (Ec.2) * inv [(Ec.1), p_B] = N y recupera el texto en claro

Por ejemplo, Bernardo elige como módulo $p_B = 1.277$ y como raíz primitiva $\alpha_B = 5$. Si su clave privada es b = 61, su clave pública será $\beta_B = \alpha_B{}^b \bmod p_B = 5^{61} \bmod 1.277 = 629$. Si Alicia desea enviarle el valor secreto N = 100, se realizan los siguientes cálculos:

1. Alicia usa $\nu = 583$ y calcula: $\alpha_B{}^\nu \bmod p_B = 5^{583} \bmod 1.277 = 771$
2. Alicia calcula $N*\beta_B{}^\nu \bmod p_B = 100*629^{583} \bmod 1.277 = 1.200$
3. Alicia envía a Bernardo ese par de números {771, 1.200}
4. Bernardo con su clave privaba b calcula $(\alpha_B{}^\nu)^b \bmod p_B = 771^{61} \bmod 1.277 = 12$
5. Bernardo calcula el $inv[(\alpha_B{}^\nu)^b, p_B] = inv [12, 1.277] = 745$
6. Bernardo calcula $[N*(\alpha_B{}^b)^\nu] * inv[(\alpha_B{}^\nu)^b, p_B] = 1.200 * 745 \bmod 1.277 = 100$
7. Bernardo recupera el texto en claro N = 100

Si lo que se desea es firmar digitalmente un mensaje de Alicia para Bernardo, se procede de la siguiente manera:

Emisión: Firma de Alicia

1. Alicia tiene como clave pública p_A, α_A, β_A, y como clave privada a
2. Alicia genera un número H que será primo relativo con ϕ_{pA}. Como p_A es primo, entonces $\phi_{pA} = p_A - 1$. Es decir, se cumplirá que mcd {H, (p_A - 1)} = 1
3. Alicia calcula: inv {H, ϕ_{pA}}
4. Alicia calcula la primera parte de su firma: $r = \alpha_A{}^H \bmod p_A$
5. Alicia resuelve la congruencia $M = a*r + H*s \bmod \phi_{pA}$, es decir:
6. $s = (M - a*r) * inv (H, \phi_{pA}) \bmod \phi_{pA}$
7. Alicia envía a Bernardo el par de números {r, s}, además del mensaje M

Recepción: Comprobación de la firma por Bernardo

1. Usando la clave pública de Alicia (p_A, α_A, β_A), Bernardo calculará:

   $r^s \bmod p_A$

   $\beta_A{}^r \bmod p_A$

$k = [\beta_A^r * r^s] \bmod p_A$

2. Como $r = \alpha_A^H \bmod p_A$ y además $\beta_A = \alpha_A^a \bmod p_A$, entonces:

   $k = [\alpha_A^{ar} * \alpha_A^{Hs}] \bmod p_A = \alpha_A^{(ar + Hs)} \bmod p_A$

3. Como $M = (a*r + H*s) \bmod \phi_{pA}$ y α_A es una raíz primitiva de p_A, se cumple que:

   si $k = \alpha_A^{M'} \bmod p_A$, siendo M' el mensaje recibido, entonces la firma es válida

En la práctica lógicamente no se firmará un mensaje M sino su función hash h(M).

Por ejemplo, Alicia va a firmar el mensaje M = 857, que es el resultado de h(M), y sus claves son $p_A = 1.511$, con $\alpha_A = 11$, a = 122, $\beta_A = 11^{122} \bmod 1.511 = 1.029$.

1. Alicia elige H = 359 (que no tiene factores en común con p - 1 = 1.510)
2. Alicia encuentra el inv $\{H, \phi_{pA}\}$ = inv (359, 1.510) = 959
3. Alicia calcula $r = \alpha_A^H \bmod p_A = 11^{359} \bmod 1.511 = 1.187$
4. Alicia calcula $s = (M - a*r)*inv (H, \phi_{pA}) \bmod \phi_{pA}$
5. $s = (857 – 122*1.187)*959 \bmod 1.510 = -143.957\ 959 \bmod 1.510 = 7$
6. Bernardo recibe {r = 1.187, s = 7} y calcula:
7. $r^s \bmod p_A = 1.187^7 \bmod 1.511 = 235$
8. $\beta_A^r \bmod p_A = 1.029^{1.187} \bmod 1.511 = 1.138$
9. $k = [\beta_A^r * r^s] \bmod p_A = 1.138*235 \bmod 1.511 = 1.494$
10. Suponiendo que Bernardo recibe el mensaje M' que es el correcto 857, calcula:

    $\alpha_A^{M'} \bmod p_A = 11^{857} \bmod 1.511 = 1.494$

    Como este valor es igual al k encontrado, se acepta la firma como válida.

Test del capítulo 5

1. La criptografía asimétrica nace a partir de:

A. Los trabajos de Rivest, Shamir y Adleman en 1978

B. Los trabajos de Diffie y Hellman en 1986

C. Los trabajos de Diffie y Hellman en 1976

D. Los trabajos de Rivest, Shamir y Adleman en 1976

2. Los algoritmos de Diffie-Hellman y RSA:

A. Basan su seguridad en el problema del logaritmo discreto

B. Son algoritmos de cifra asimétrica

C. Son algoritmos de cifra simétrica

D. Son algoritmos donde los interlocutores no guardan valores secretos

3. Sea una clave RSA con las claves públicas e = 3 y n = 33.

A. La clave privada será d = 5

B. La clave privada será d = 6

C. La clave privada será d = 7

D. La clave privada será d = 8

4. Supuesto el sistema RSA con e = 5 y n = 35, el resultado de cifrar el número 3 será:

A. 31

B. 32

C. 33

D. 34

5. Una clave RSA se dice que es óptima si se cumple que:

A. Tiene 9 números no cifrables y 1 clave privada pareja

B. Tiene 1 número no cifrable y 9 claves privadas parejas

C. Tiene 1 número no cifrable y 1 clave privada pareja

D. No tiene números no cifrables ni claves privadas parejas

6. La seguridad de RSA se basa en:

A. La dificultad de factorización de números grandes compuestos

B. La dificultad de encontrar el logaritmo discreto en primos grandes

C. La dificultad de factorización de números grandes primos

D. La dificultad de factorización de números grandes decimales

7. El algoritmo RSA puede atacarse:

A. Por factorización, paradoja del cumpleaños y logaritmo discreto

B. Por factorización, paradoja del cumpleaños y redundancia del lenguaje

C. Por factorización, logaritmo discreto y cifrado cíclico

D. Por factorización, paradoja del cumpleaños y cifrado cíclico

8. Los algoritmos de Diffie y Hellman y Elgamal tienen en común que:

A. Su fortaleza está basada en el problema del logaritmo discreto en n = p*q

B. Su fortaleza está basada en el problema del logaritmo discreto en p

C. La cifra se realiza en módulo n = p*q

D. La cifra se realiza en módulo p-1

9. ¿Cuál de los siguientes elementos no debe ir en el contenido de un certificado de clave pública?

A. El algoritmo asimétrico empleado

B. La función hash empleada

C. La clave pública del usuario

D. La clave privada del usuario

10. La firma digital estándar DSS es:

A. Una modificación del algoritmo de Diffie y Hellman

B. Una modificación del algoritmo RSA

C. Una modificación del algoritmo AES

D. Una modificación del algoritmo de Elgamal

Bibliografía complementaria

- Caballero, P. (1996). Introducción a la criptografía. Madrid: Ra-Ma.
- Lucena, M. J. (2010). Criptografía y seguridad en computadores. http://criptografiayseguridad.blogspot.com.es/p/criptografia-y-seguridad-en.html
- Fúster, A. et al. (2004). Técnicas criptográficas de protección de datos. Madrid: Ra-Ma.
- Pastor, J. (1998). Criptografía digital: fundamentos y aplicaciones. Zaragoza: Prensas Universitarias de Zaragoza.
- Ramió, J. (2006). Libro Electrónico de Seguridad Informática y Criptografía, Versión 4.1. http://www.criptored.upm.es/guiateoria/gt_m001a.htm
- Ramió, J. (2009) Necesidad y Urgencia de un Nuevo Estándar en Funciones Hash. http://www.criptored.upm.es/guiateoria/gt_m001n.htm