

Capítulo 15

Funciones Hash en Criptografía

Seguridad Informática y Criptografía



Material Docente de
Libre Distribución

Ultima actualización del archivo: 01/03/06
Este archivo tiene: 34 diapositivas

Dr. Jorge Ramió Aguirre
Universidad Politécnica de Madrid

Este archivo forma parte de un curso completo sobre Seguridad Informática y Criptografía. Se autoriza el uso, reproducción en computador y su impresión en papel, sólo con fines docentes y/o personales, respetando los créditos del autor. Queda prohibida su comercialización, excepto la edición en venta en el Departamento de Publicaciones de la Escuela Universitaria de Informática de la Universidad Politécnica de Madrid, España.

Uso de las funciones hash en criptografía

- Una de las aplicaciones más interesantes de la actual criptografía es la posibilidad real de añadir en un mensaje una firma digital: la autenticación completa.
- Todo esto comienza en el año 1976 cuando Diffie y Hellman presentan un modelo de cifrado asimétrico con clave pública. Con los antiguos sistemas de cifra de clave simétrica esto era inviable o bien muy complejo.
- No obstante, dado que los sistemas de clave pública son muy lentos, en vez de firmar digitalmente el mensaje completo, en un sistema criptográfico se incluirá como firma digital una operación de cifra con la clave privada del emisor sobre un resumen o hash de dicho mensaje, representado por sólo una centena de bits.

Funciones hash

Mensaje = $M \Rightarrow$ Función Resumen = $h(M)$

Firma (rúbrica): $r = E_{d_E}\{h(M)\}$

d_E es la clave privada del emisor que firmará $h(M)$

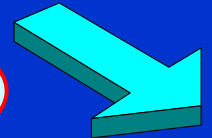
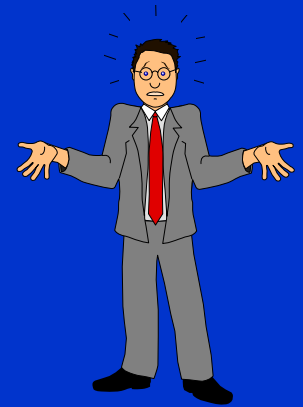
¿Cómo se comprueba la identidad en destino?

Se descifra la rúbrica r con la clave pública del emisor e_E . Al mensaje en claro recibido M' (si viniese cifrado, se descifra) se le aplica la misma función hash que en emisión. Si los valores son iguales, la firma es auténtica y el mensaje íntegro:

Calcula: $E_{e_E}(r) = h(M)$

Compara: ¿ $h(M') = h(M)$?

¿Qué seguridad nos da un resumen de k bits?



Seguridad asociada a una función hash

Suponga que hemos creado una función hash de forma que el resumen es sólo de 4 bits, independientemente del tamaño del mensaje de entrada.

La pregunta es: ¿cuál es la probabilidad de que dos mensajes distintos tengan igual función hash? Si esta probabilidad fuese muy baja (en este caso 1/16: hash desde 0000 hasta 1111) podría darse el siguiente caso: alguien modifica nuestro mensaje firmado, y envía ese mensaje falso con la firma del primero ya que en ambos casos son los mismos 4 bits...

Mensaje 1: “Rechazamos el contrato por no interesarnos nada” hash: 1101

Mensaje 2: “Firma todo lo que te pongan porque nos interesa” hash: 1101

Observe que ambos mensajes tienen 47 caracteres. Así, podríamos crear una gran cantidad de mensajes diferentes que digan cosas distintas incluso con igual número de caracteres... ¡hasta que los dos hash coincidan!

Por este motivo, para que las funciones hash sean interesantes en criptografía deben cumplir un conjunto de propiedades. 

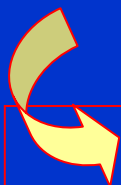
Propiedades de las funciones hash

$h(M)$ será segura si tiene las siguientes características:

1. **Unidireccionalidad:** conocido un resumen $h(M)$, debe ser computacionalmente imposible encontrar M a partir de dicho resumen.
2. **Compresión:** a partir de un mensaje de cualquier longitud, el resumen $h(M)$ debe tener una longitud fija. Lo normal es que la longitud de $h(M)$ sea menor que el mensaje M .
3. **Facilidad de cálculo:** debe ser fácil calcular $h(M)$ a partir de un mensaje M .
4. **Difusión:** el resumen $h(M)$ debe ser una función compleja de todos los bits del mensaje M : si se modifica un solo bit del mensaje M , el hash $h(M)$ debería cambiar la mitad de sus bits aproximadamente.

Colisiones: resistencia débil y fuerte

5. **Colisión simple:** será computacionalmente imposible conocido M , encontrar otro M' tal que $h(M) = h(M')$. Esto se conoce como *resistencia débil a las colisiones*.
6. **Colisión fuerte:** será computacionalmente difícil encontrar un par (M, M') de forma que $h(M) = h(M')$. Esto se conoce como *resistencia fuerte a las colisiones*.



Ataque por la paradoja del cumpleaños

Para tener *confianza* en encontrar dos mensajes con el mismo resumen $h(M)$ -probabilidad $\geq 50\%$ - no habrá que buscar en 2^n (p.e. 2^{128}), bastará una búsqueda en el espacio $2^{n/2}$ (2^{64}).
¡La complejidad algorítmica se reduce de forma drástica!

Repaso de la paradoja del cumpleaños

En el capítulo de criptosistemas asimétricos ya se presentaba este problema matemático, consistente en que exista **confianza**, es decir una probabilidad mayor que el 50%, de que en un aula con 365 personas, dos de ellas al azar cumplan años en la misma fecha.

En realidad no se trata de una paradoja, pero parece serlo dado que ese número que buscamos es sólo de 23 personas.

Explicación: si éstos entran en el aula de uno en uno y van borrando de una pizarra su cumpleaños, el primero tendrá una probabilidad de que su número **no** esté borrado igual a $n/n = 1$, el segundo de $(n-1)/n$, etc. La probabilidad de **no** coincidencia es $p_{NC} = n!/(n-k)!n^k$. Si $k = 23$, se tiene $p_{NC} = 0,493$ y la probabilidad de coincidencia es $p_C = 0,507$.

👉 En un hash, esto sería equivalente a sacar dos mensajes de dos conjuntos disjuntos y comparar sus valores; si los hash son distintos sacamos un nuevo mensaje y lo comparamos con los anteriores,... hasta que haya una colisión.

Algoritmos de resumen en criptografía

- Los más usados actualmente
- ✓ • **MD5**: Ron Rivest 1992. Mejoras al MD4 y MD2 (1990), es más lento pero con mayor nivel de seguridad. Resumen de 128 bits.
 - ✓ • **SHA-1**: Del NIST, National Institute of Standards and Technology, 1994. Similar a MD5 pero con resumen de 160 bits. Existen otras propuestas conocidas como **SHA-256** y **SHA-512**, posibles estándares.
 - **RIPEMD**: Comunidad Europea, RACE, 1992. Resumen de 160 bits.
 - **N-Hash**: Nippon Telephone and Telegraph, 1990. Resumen: 128 bits.
 - **Snefru**: Ralph Merkle, 1990. Resúmenes entre 128 y 256 bits. Ha sido criptoanalizado y es lento.
 - **Tiger**: Ross Anderson, Eli Biham, 1996. Resúmenes de hasta 192 bits. Optimizado para máquinas de 64 bits (Alpha).
 - **Panama**: John Daemen, Craig Clapp, 1998. Resúmenes de 256 bits de longitud. Trabaja en modo función hash o como cifrador de flujo.
 - **Haval**: Yuliang Zheng, Josef Pieprzyk y Jennifer Seberry, 1992. Admite 15 configuraciones diferentes. Hasta 256 bits.

Función resumen MD5

Esta función ya está obsoleta desde mediados de 2005. No obstante, es interesante su estudio dada la sencillez del algoritmo y su generalidad.

Algoritmo básico de Message Digest 5

- a) Un mensaje M se convierte en un bloque múltiplo de 512 bits, añadiendo bits si es necesario al final del mismo.
- b) Con los 128 bits de cuatro vectores iniciales $ABCD$ de 32 bits cada uno y el primer bloque del mensaje de 512 bits, se realizan diversas operaciones lógicas entre ambos bloques.
- c) La salida de esta operación (128 bits) se convierte en el nuevo conjunto de 4 vectores $A'B'C'D'$ y se realiza la misma función con el segundo bloque de 512 bits del mensaje, y así hasta el último bloque del mensaje. Al terminar, el algoritmo entrega un resumen que corresponde a los últimos 128 bits de estas operaciones.

<http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html>



Etapas de MD5

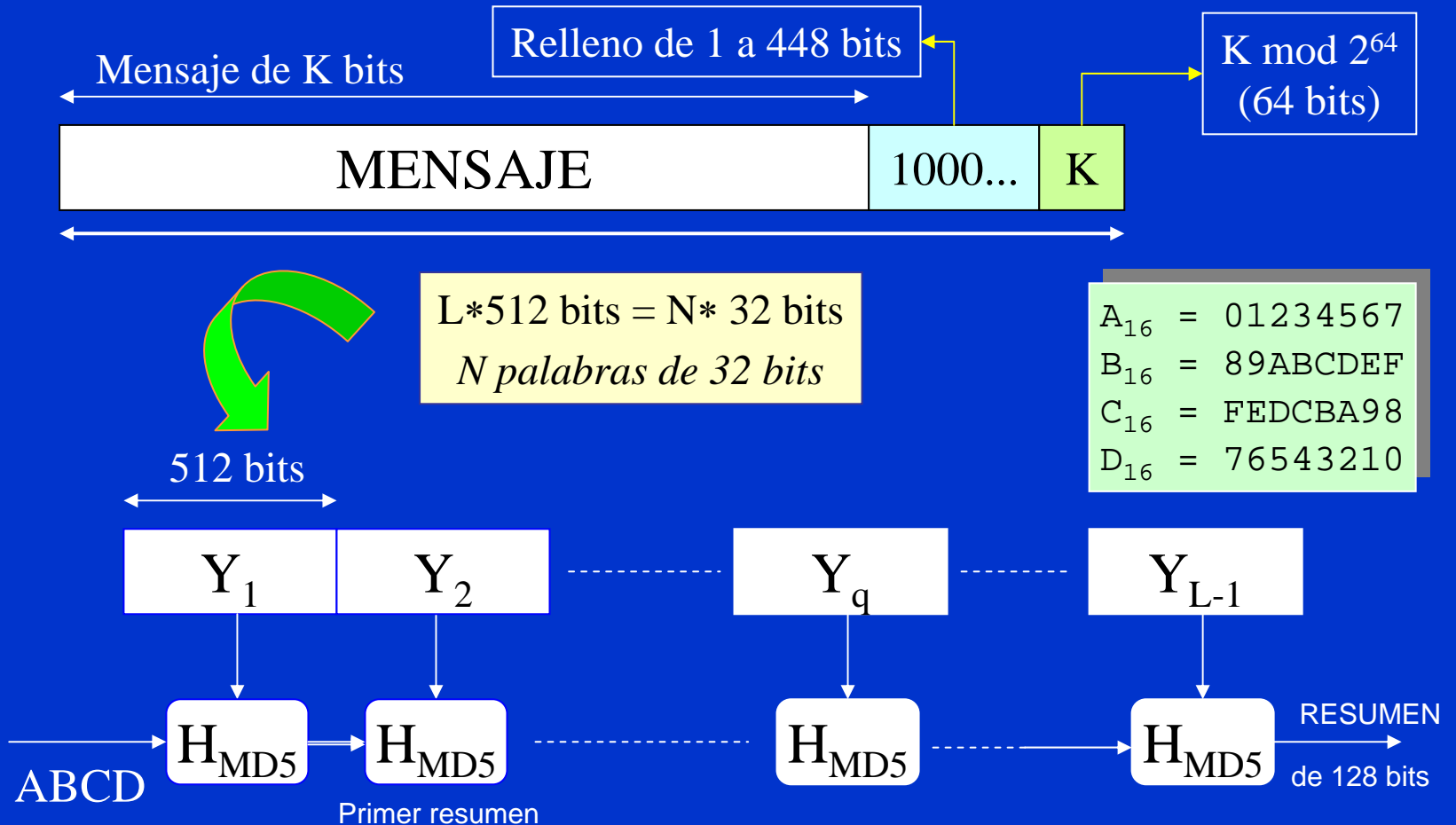
Bloques funcionales de MD5

- a) Añadir bits para congruencia módulo 512, reservando los últimos 64 bits para un indicador de longitud.
- b) Añadir indicación de la longitud del mensaje en los 64 bits reservados para ello.
- c) Inicializar el vector ABCD de claves con un valor que no es secreto.
- d) Procesar bloques de 512 bits, entregando una salida de 128 bits que formarán nuevamente el vector ABCD.
- e) Obtener el resumen de los últimos 128 bits.

<http://www.faqs.org/rfcs/rfc1321.html>



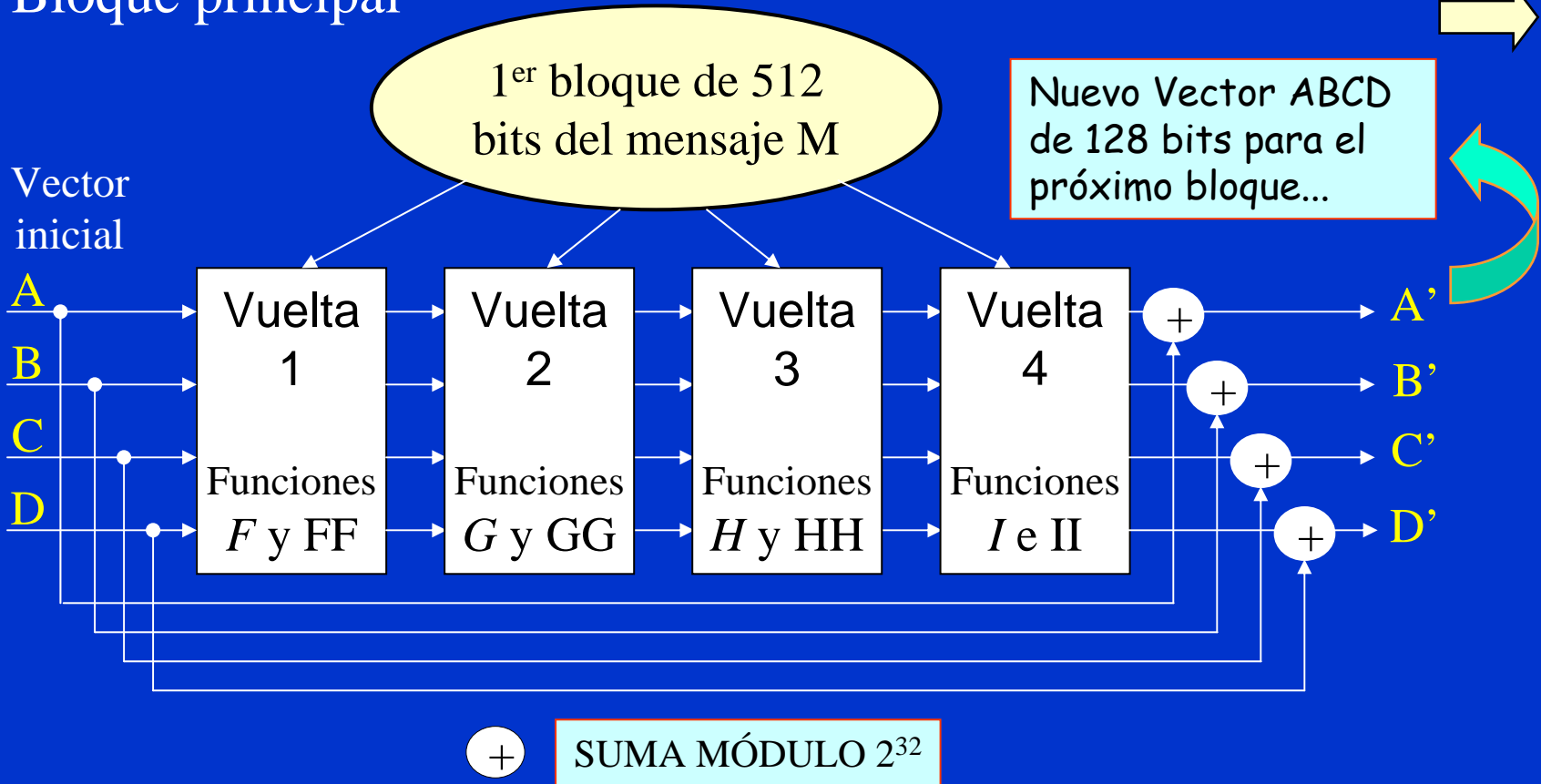
Esquema de la función MD5



Bloque principal de MD5

Bloque principal

¿Qué hacen las funciones F y FF ...?



Esquema funciones F, G, H, I en MD5

Vector inicial ABCD

$A_{16} = 01234567$
 $B_{16} = 89ABCDEF$
 $C_{16} = FEDCBA98$
 $D_{16} = 76543210$



128 bits

función no lineal

$x, y, z \Rightarrow b, c, d$



$F(x, y, z)$
 $(x \text{ AND } y) \text{ OR } (\text{NOT } x \text{ AND } z)$
 $G(x, y, z)$
 $(x \text{ AND } z) \text{ OR } (y \text{ AND } \text{NOT } z)$
 $H(x, y, z)$
 $x \text{ XOR } y \text{ XOR } z$
 $I(x, y, z)$
 $y \text{ XOR } (x \text{ OR } \text{NOT } z)$

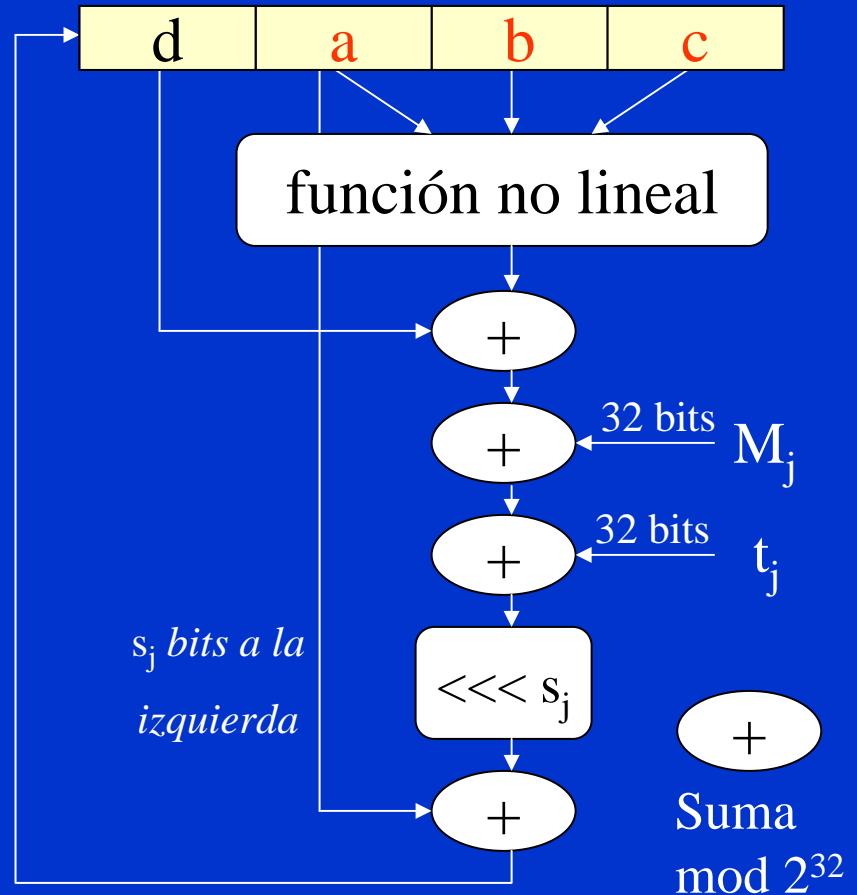
$F(b, c, d)$
 $(b \text{ AND } c) \text{ OR } (\text{NOT } b \text{ AND } d)$
 $G(b, c, d)$
 $(b \text{ AND } d) \text{ OR } (c \text{ AND } \text{NOT } d)$
 $H(b, c, d)$
 $b \text{ XOR } c \text{ XOR } d$
 $I(b, c, d)$
 $c \text{ XOR } (b \text{ OR } \text{NOT } d)$

Algoritmo de las funciones en MD5

→ Desplazamiento del registro
Situación luego del desplazamiento

Se repite el proceso para M_{j+1} hasta 16 bloques del texto. En las vueltas 2, 3 y 4 se repite el proceso ahora con funciones G, H e I.

El algoritmo realiza $4 * 16 = 64$ vueltas para cada uno de los bloques de 512 bits



Funciones no lineales en MD5

Funciones no lineales
en cada vuelta

Vector de 128 bits

a	b	c	d
---	---	---	---

1ª Vuelta:

$$FF(a,b,c,d,M_j,t_j,s) \Rightarrow a = b + ((a + F(b,c,d) + M_j + t_j) \lll s)$$

2ª Vuelta:

$$GG(a,b,c,d,M_j,t_j,s) \Rightarrow a = b + ((a + G(b,c,d) + M_j + t_j) \lll s)$$

3ª Vuelta:

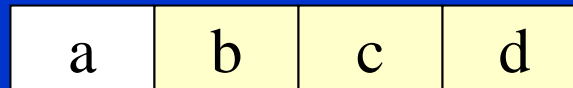
$$HH(a,b,c,d,M_j,t_j,s) \Rightarrow a = b + ((a + H(b,c,d) + M_j + t_j) \lll s)$$

4ª Vuelta:

$$\Pi(a,b,c,d,M_j,t_j,s) \Rightarrow a = b + ((a + I(b,c,d) + M_j + t_j) \lll s)$$

Algoritmo y desplazamiento en MD5

Vector de 128 bits



Sea f la función F, G, H o I según la vuelta.
El algoritmo será:

Para $j = 0$ hasta 15 hacer:

$$\text{TEMP} = [(a + f(b,c,d) + M_j + t_j) \lll s_j]$$

$$a = d$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = \text{TEMP}$$

Operaciones en 1ª y 2ª vueltas en MD5

FF (a, b, c, d, M_j, t_j, s)

FF(a, b, c, d, M₀, D76AA478, 7)
 FF(d, a, b, c, M₁, E8C7B756, 12)
 FF(c, d, a, b, M₂, 242070DB, 17)
 FF(b, c, d, a, M₃, C1BDCEEE, 22)
 FF(a, b, c, d, M₄, F57C0FAF, 7)
 FF(d, a, b, c, M₅, 4787C62A, 12)
 FF(c, d, a, b, M₆, A8304613, 17)
 FF(b, c, d, a, M₇, FD469501, 22)
 FF(a, b, c, d, M₈, 698098D8, 7)
 FF(d, a, b, c, M₉, 8B44F7AF, 12)
 FF(c, d, a, b, M₁₀, FFFF5BB1, 17)
 FF(b, c, d, a, M₁₁, 895CD7BE, 22)
 FF(a, b, c, d, M₁₂, 6B901122, 7)
 FF(d, a, b, c, M₁₃, FD987193, 12)
 FF(c, d, a, b, M₁₄, A679438E, 17)
 FF(b, c, d, a, M₁₅, 49B40821, 22)

Primera vuelta

GG (a, b, c, d, M_j, t_j, s)

GG(a, b, c, d, M₁, F61E2562, 5)
 GG(d, a, b, c, M₆, C040B340, 9)
 GG(c, d, a, b, M₁₁, 265E5A51, 14)
 GG(b, c, d, a, M₀, E9B6C7AA, 20)
 GG(a, b, c, d, M₅, D62F105D, 5)
 GG(d, a, b, c, M₁₀, 02441453, 9)
 GG(c, d, a, b, M₁₅, D8A1E681, 14)
 GG(b, c, d, a, M₄, E7D3FBC8, 20)
 GG(a, b, c, d, M₉, 21E1CDE6, 5)
 GG(d, a, b, c, M₁₄, C33707D6, 9)
 GG(c, d, a, b, M₃, F4D50D87, 14)
 GG(b, c, d, a, M₈, 455A14ED, 20)
 GG(a, b, c, d, M₁₃, A9E3E905, 5)
 GG(d, a, b, c, M₂, FCEFA3F8, 9)
 GG(c, d, a, b, M₇, 676F02D9, 14)
 GG(b, c, d, a, M₁₂, 8D2A4C8A, 20)

Segunda vuelta

Operaciones en 3ª y 4ª vueltas en MD5

HH (a, b, c, d, M_j, t_j, s)

HH(a, b, c, d, M₅, FFFA3942, 4)
 HH(d, a, b, c, M₈, 8771F681, 11)
 HH(c, d, a, b, M₁₁, 6D9D6122, 16)
 HH(b, c, d, a, M₁₄, FDE5380C, 23)
 HH(a, b, c, d, M₁, A4BEEA44, 4)
 HH(d, a, b, c, M₄, 4BDECFA9, 11)
 HH(c, d, a, b, M₇, F6BB4B60, 16)
 HH(b, c, d, a, M₁₀, BEBFBC70, 23)
 HH(a, b, c, d, M₁₃, 289B7EC6, 4)
 HH(d, a, b, c, M₀, EAA127FA, 11)
 HH(c, d, a, b, M₃, D4EF3085, 16)
 HH(b, c, d, a, M₆, 04881D05, 23)
 HH(a, b, c, d, M₉, D9D4D039, 4)
 HH(d, a, b, c, M₁₂, E6DB99E5, 11)
 HH(c, d, a, b, M₁₅, 1FA27CF8, 16)
 HH(b, c, d, a, M₂, C4AC5665, 23)

Tercera vuelta

II (a, b, c, d, M_j, t_j, s)


II(a, b, c, d, M₀, F4292244, 6)
 II(d, a, b, c, M₇, 411AFF97, 10)
 II(c, d, a, b, M₁₄, AB9423A7, 15)
 II(b, c, d, a, M₅, FC93A039, 21)
 II(a, b, c, d, M₁₂, 655B59C3, 6)
 II(d, a, b, c, M₃, 8F0CCC92, 10)
 II(c, d, a, b, M₁₀, FFEFF47D, 15)
 II(b, c, d, a, M₁, 85845DD1, 21)
 II(a, b, c, d, M₈, 6FA87E4F, 6)
 II(d, a, b, c, M₁₅, FE2CE6E0, 10)
 II(c, d, a, b, M₆, A3014314, 15)
 II(b, c, d, a, M₁₃, 4E0811A1, 21)
 II(a, b, c, d, M₄, F7537E82, 6)
 II(d, a, b, c, M₁₁, BD3AF235, 10)
 II(c, d, a, b, M₂, 2AD7D2BB, 15)
 II(b, c, d, a, M₉, EB86D391, 21)

Cuarta vuelta

Función resumen SHA-1

Algoritmo:

Es muy similar a MD5. También trata bloques de 512 bits de mensaje con un total de 80 vueltas, pero en este caso el vector inicial tiene una palabra más de 32 bits (E) por lo que el resumen será de 160 bits.

Un resumen de 128 bits (MD5) tiene una complejidad algorítmica de tan sólo 2^{64} , un valor en la actualidad muy comprometido... 

En cambio la función SHA-1, Secure Hash Algorithm, entregará un resumen de 160 bits; con una complejidad algorítmica de 2^{80} . 

<http://www.faqs.org/rfcs/rfc3174.html>



Esquema del resumen SHA-1

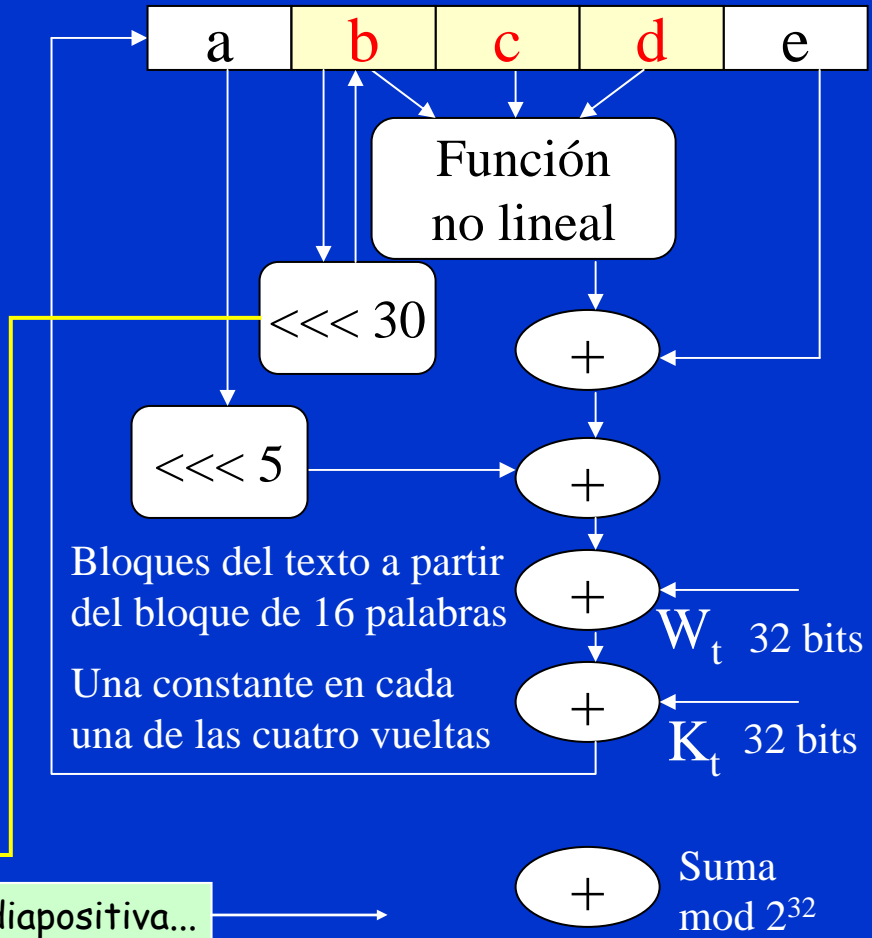
Vector inicial ABCDE

Registro de 160 bits

$A_{16} = 67452301$
 $B_{16} = EFC DAB89$
 $C_{16} = 98BADC FE$
 $D_{16} = 10325476$
 $E_{16} = C3D2E1F0$

Después de esta última operación, se produce el desplazamiento del registro hacia la derecha

Véase la próxima diapositiva...



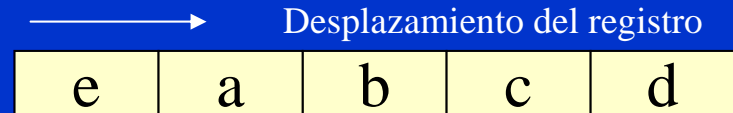
Vueltas funciones F, G, H, I en SHA-1

F (b, c, d) → vueltas t = 0 a 19
 (b AND c) OR ((NOT b) AND d)

G (b, c, d) → vueltas t = 20 a 39
 b XOR c XOR d

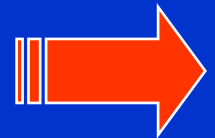
H (b, c, d) → vueltas t = 40 a 59
 (b AND c) OR (b AND d) OR
 (c AND d)

I (b, c, d) → vueltas t = 60 a 79
 b XOR c XOR d



Se repite el proceso con la función F para las restantes 15 palabras de 32 bits del bloque actual hasta llegar a 20. En vueltas 2, 3 y 4 se repite el proceso con funciones G, H e I.

Tenemos $4 \cdot 20 = 80$ pasos por cada bloque de 512 bits. Pero ... ¿cómo es posible repetir 80 veces un bloque que sólo cuenta con 16 bloques de texto de 32 bits cada uno?



Las 80 vueltas en SHA-1

Vector de 160 bits

a	b	c	d	e
---	---	---	---	---

Cada bloque de 16 palabras del mensaje ($M_0 \dots M_{15}$) se expandirá en 80 palabras ($W_0 \dots W_{79}$) según el algoritmo:

$$W_t = M_t \text{ (para } t = 0, \dots, 15)$$

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1 \text{ (para } t = 16, \dots, 79)$$

y además: $K_t = 5A827999$ para $t = 0, \dots, 19$

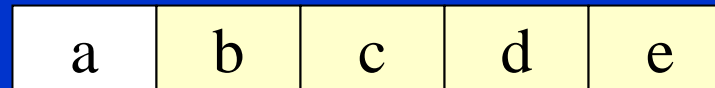
$$K_t = 6ED9EBA1 \text{ para } t = 20, \dots, 39$$

$$K_t = 8F1BBCDC \text{ para } t = 40, \dots, 59$$

$$K_t = CA62C1D6 \text{ para } t = 60, \dots, 79$$

Algoritmo y desplazamiento en SHA-1

Vector de 160 bits



El algoritmo para cada bloque de 512 bits será:

Para $t = 0$ hasta 79 hacer:

$$\text{TEMP} = (a \lll 5) + f_t(b,c,d) + e + W_t + K_t$$

$$a = e$$

$$e = d$$

$$d = c$$

$$c = b \lll 30$$

$$b = a$$

$$a = \text{TEMP}$$

Comparativa entre MD5 y SHA-1

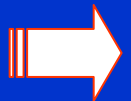
- SHA-1 genera una salida de 160 bits de longitud mientras que MD5 genera sólo 128 bits.
 - La dificultad de generar un mensaje que tenga un resumen dado es del orden de 2^{128} operaciones para MD5 y 2^{160} para SHA-1.
 - La dificultad de generar dos mensajes aleatorios distintos y que tengan el mismo resumen (ataques basados en paradoja del cumpleaños) es del orden de 2^{64} operaciones para MD5 y 2^{80} para SHA-1.
- Esta diferencia de 16 bits a favor de SHA-1 lo convierte en más seguro y resistente a ataques por fuerza bruta que el algoritmo MD5. Aunque es más lento que MD5, SHA-1 es hoy el estándar como función hash.

Pasos y tasas de cifra en MD5 y SHA-1

- Ambos algoritmos procesan bloques de 512 bits y emplean 4 funciones primitivas para generar el resumen del mensaje, pero...
 - SHA-1 realiza un mayor número de pasos que MD5: 80 frente a los 64 que realiza MD5.
 - SHA-1 debe procesar 160 bits de buffer en comparación con los 128 bits de MD5.
 - Por estos motivos la ejecución del algoritmo SHA-1 es más lenta que la de MD5 usando un mismo hardware. Por ejemplo, un programa realizado en C entregaba en un Pentium a 266 MHz (no importa que esta velocidad sea tan baja) una tasa del orden de 20 Mbits/seg para SHA-1 y para MD5 esta tasa llegaba a los 60 Mbits/seg.

Más diferencias entre MD5 y SHA-1

- La longitud máxima del mensaje para SHA-1 debe ser menor de 2^{64} bits, mientras que MD5 no tiene limitaciones de longitud.
- MD5 emplea 64 constantes (una por cada paso), mientras que SHA-1 sólo emplea 4 (una para cada 20 pasos).
- MD5 se basa en la arquitectura little-endian, mientras que SHA-1 se basa en la arquitectura big-endian. Por ello el vector ABCD inicial en MD5 y SHA-1 son iguales:
 - A = 01234567 (MD5) \Rightarrow 67452301 (SHA-1)
 - B = 89ABCDEF (MD5) \Rightarrow EFCDAB89 (SHA-1)
 - C = FEDCBA98 (MD5) \Rightarrow 98BADCFE (SHA-1)
 - D = 76543210 (MD5) \Rightarrow 10325476 (SHA-1)



Arquitecturas little-endian y big-endian

Arquitectura little-endian:

- Esta es la arquitectura empleada en procesadores Intel de la familia 80xxx y Pentium.
- Para almacenar una palabra en memoria, **el byte menos significativo** de los que forman dicha palabra se guarda en la posición más baja de la memoria.

Arquitectura big-endian:

- Empleada por otras arquitecturas como SUN.
- Para almacenar una palabra en memoria, **el byte más significativo** de los que forman dicha palabra se guarda en la posición más baja de memoria.

<http://www.algoritmia.net/articles.php?id=57>



Ejemplo little-endian versus big-endian

Supongamos que queremos almacenar en memoria la siguiente palabra de 32 bits (4 bytes) representada en hexadecimal:

$$76543210 \Rightarrow \begin{array}{|c|c|c|c|} \hline 76 & 54 & 32 & 10 \\ \hline \end{array}$$

Si consideramos que las posiciones de memoria más bajas se encuentran a la izquierda y las más altas a la derecha:

En formato *little-endian* se representa:

01	23	45	67
----	----	----	----

En formato *big-endian* se representa:

67	45	23	01
----	----	----	----

Últimos ataques a las funciones hash

- Ya a finales del año 2004 científicos chinos de la Shandong University presentan trabajos en los que se analizan las debilidades reales de las funciones hash como MD5 y SHA-1 ante colisiones.
- Aunque no está claro que este tipo de ataques pudiese derivar en acciones de fraude, como sería suplantar un hash por otro igual y que en recepción se aceptase como válido si bien este último proviene de un mensaje distinto, es un claro motivo de preocupación actual.
- El problema de estas vulnerabilidades estriba en que muchos servidores Web presentan un certificado digital X.509 firmado en el mejor de los casos a partir de una función hash SHA-1 y, lo que es peor aún, todavía hay algunos que usan un hash MD5.... este último mucho menos seguro que el primero.

http://www.criptored.upm.es/guiateoria/gt_m238a.htm



http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html



Funciones hash para la autenticación

- Las funciones hash vistas (MD5, SHA-1, etc.) pueden usarse además para autenticar a dos usuarios.
- Como carecen de una clave privada no pueden usarse de forma directa para estos propósitos. No obstante, existen algoritmos que permiten añadirles esta función.
- Entre ellos está HMAC, una función que usando los hash vistos y una clave secreta, autentica a dos usuarios mediante sistemas de clave secreta. Las funciones MAC, **M**essage **A**uthentication **C**ode, y HMAC se tratarán en el próximo capítulo dedicado a la autenticación y firma digital.
- HMAC se usa en plataformas IP seguras como por ejemplo en Secure Socket Layer, SSL.

Fin del capítulo

Cuestiones y ejercicios

1. ¿Qué propiedades debe tener una función hash para que su uso en criptografía sea de interés? ¿Vale cualquier función reductora?
2. ¿Por qué prospera un ataque basado en la paradoja del cumpleaños con un tiempo significativamente menor que un ataque elemental?
3. Se va a aplicar la función MD5 a un mensaje de longitud 250 bytes. ¿Cómo se rellena y cómo queda el último bloque?
4. ¿Por qué razón decimos que la función SHA-1 es actualmente un estándar más seguro que la función MD5? ¿Es seguro hoy SHA-1?
5. ¿Cómo puede la función SHA-1 hacer 80 vueltas con bloques de 32 bits partiendo de un bloque de texto o mensaje de sólo 512 bits?
6. ¿Qué función hash es más rápida, MD5 o SHA-1? ¿Por qué?
7. Si en un mensaje cambiamos un bit, ¿en cuánto debería cambiar su hash aproximadamente con respecto a su resumen anterior?

Use el portapapeles

Prácticas del tema 15 (1/3)

Software CriptoRes:

http://www.criptored.upm.es/software/sw_m001h.htm



1. Usando el botón de generación de resumen, encuentre el hash MD5 del mensaje $M = \text{Hola}$. Para el mismo mensaje, repita el resumen para SHA-1.
2. Busque en su computador el archivo SegInfCrip_v40.zip, encuentre el hash MD5 y el hash SHA-1. Repita un par de veces la operación de resumen y compare las velocidades de cálculo de cada función.
3. Obtenga la función hash MD5 de $M = \text{Prueba 122 del hash}$. Abra otra ventana y encuentre ahora el hash de $M = \text{Prueba 123 del hash}$. Observe que los mensajes difieren sólo en un bit ($2 = 0011\ 0010$; $3 = 0011\ 0011$).
4. Abra la calculadora de Windows dos veces y copie el hexadecimal de cada hash en cada una, cambie luego a binario. Haga un XOR entre los dos valores y compruebe que con cambiar sólo un bit del mensaje, el hash cambia más de la mitad de bits. Recuerde que estas calculadoras trabajan con 64 bits. Asegúrese que ambos valores tengan el mismo número de bits.

Use el portapapeles

Prácticas del tema 15 (2/3)

5. Para el mensaje $M = 123$, encuentre el hash MD5 a través del botón de seguimiento. Abra la pestaña del seguimiento del algoritmo y observe que se ha operado con un solo bloque.
6. Active la opción A Nivel de Pasos y vuelva a calcular el hash. Observe los valores **313233** (hexadecimal de **123**) y luego el valor **80** que significa un relleno (1 seguido de ceros). Al final verá un bloque de 64 bits (últimas dos palabras) cuyo valor es **18**. Con la calculadora de Windows compruebe que ese valor en decimal corresponde a 24, los 3 bytes del mensaje.
7. Observe ahora el relleno y el número de bits para el hash MD5 del mensaje $M =$ **En este caso tenemos 232 bits**. Compruebe este valor.
8. Si el mensaje tiene exactamente 512 bits, siempre se incluye un bloque con relleno. Compruébelo con el mensaje de 64 bytes $M =$ **Y en este caso habrá siempre un bloque extra como ya se ha dicho**. Recuerde: $h = 68$ y $o = 6F$. Observe que a nivel de pasos no se ve relleno (se muestra sólo el primer bloque) y que a nivel de bloques muestra el procesamiento de dos bloques.

Use el portapapeles

Prácticas del tema 15 (3/3)

9. Para el mensaje $M = 123$, encuentre ahora el hash SHA-1 a través del botón de seguimiento. Abra la pestaña del seguimiento del algoritmo y observe que se ha operado con un solo bloque. Observe el vector ABCDE.
10. Active la opción A Nivel de Pasos y vuelva a calcular el hash. Compare la representación del valor 123 en hexadecimal (313233) con el resultado de MD5, notación little-endian versus big-endian.
11. Observe que en SHA-1 no se reservan los últimos 64 bits para indicar el tamaño del archivo.
12. SHA-1 no acepta mensajes de tamaño mayores que 2^{64} bits. Aunque pudiera parecer una limitación, ¿a cuántos bytes correspondería este valor?
13. Compruebe gráficamente la paradoja del cumpleaños pinchando en el icono con figura de tarta. Primero indique 5 iteraciones y acepte las opciones por defecto de un seguimiento preciso de cada cumpleaños. A continuación introduzca otros números, juegue un poco con las opciones del programa y observe que el valor medio de intentos es cercano a 23.