



# CRIPTOGRAFÍA PARA INGENIER@S


## Class4crypt

© Jorgeramio 2022

Aula virtual de  
criptografía  
aplicada

Diapositivas  
utilizadas en las  
clases grabadas  
de Class4crypt

Módulo 7 Funciones hash  
Dr. Jorge Ramió Aguirre © 2022

  
Attribution-NonCommercial-  
NoDerivatives 4.0 International  
(CC BY-NC-ND 4.0)

# Class4crypt

## Tu aula virtual de criptografía aplicada



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

Dr. Jorge Ramío Aguirre

*El ingenio es intrínseco al ser humano,  
solo hay que darle una oportunidad  
para que se manifieste.*

<https://www.criptored.es/cvJorge/index.html>

- Módulo 1. Principios básicos de la seguridad
- Módulo 2. Matemáticas discretas en la criptografía
- Módulo 3. Complejidad algorítmica en la criptografía
- Módulo 4. Teoría de la información en la criptografía
- Módulo 5. Fundamentos de la criptografía
- Módulo 6. Algoritmos de criptografía clásica
- ➔ Módulo 7. Funciones hash
- Módulo 8. Criptografía simétrica en bloque
- Módulo 9. Criptografía simétrica en flujo
- Módulo 10. Criptografía asimétrica

# Class4crypt

## Módulo 7. Funciones hash

7.1 Funciones hash en la criptografía

7.2 Función hash MD5, estructura y operaciones

7.3 Función hash SHA-1

7.4 Colisiones en funciones hash MD5 y SHA-1

7.5 SHA-2, SHA-3 y resumen de funciones hash

Lista de reproducción del módulo 7 en el canal Class4crypt

<https://www.youtube.com/playlist?list=PLq6etZPDh0ksJxtuaQjIZKXyVuSq49wKG>

# Class4crypt c4c7.1

## Módulo 7. Funciones hash

### Lección 7.1. Funciones hash en la criptografía

7.1.1. Qué son y qué no son las funciones hash

7.1.2. Ejemplos de funciones hash, cálculo y utilidad en la criptografía

7.1.3. Principio del palomar: seguridad y unicidad de las funciones hash

7.1.4. Propiedades de las funciones hash: facilidad de cálculo, propiedad de unidireccionalidad, propiedad de compresión, propiedad de difusión, propiedad de no predictibilidad, resistencia a primera preimagen, resistencia simple a colisiones y resistencia fuerte a colisiones

7.1.5. Ataque al hash por la paradoja del cumpleaños

Class4crypt c4c7.1 Funciones hash en la criptografía  
<https://www.youtube.com/watch?v=jHMa3oV3IEw>

# ¿Qué son las funciones hash?

- Una función hash o resumen (*digest*) puede definirse como una función que asocia a un texto, archivo o documento electrónico  $M$  de cualquier tamaño, un resumen  $H = h(M)$  suyo, representado en bits, con una longitud fija y supuestamente único
- De esta manera, el hash es una especie de huella digital del archivo o texto, que se muestra siempre en formato hexadecimal
- El tamaño de  $h(M)$  dependerá del algoritmo utilizado
- Así, por ejemplo, la función hash MD5 devuelve 128 bits, el hash SHA-1 devuelve 160 bits y los hashes SHA-2 y SHA-3 devuelven 224, 256, 384 y 512 bits, al aplicarlos sobre un texto o archivo

# El hash no es un sistema de cifra

- Es un error muy común señalar a las funciones hash como algoritmos de cifra
- No son algoritmos de cifra porque no hay ninguna clave
- No confundir con HMAC, que sí incluye una clave
- Dependiendo del entorno de ejecución (hardware, software, web online), las funciones hash tienen un rendimiento o tasa que va desde las pocas decenas de MegaBytes por segundo a dos o tres centenas de MegaBytes por segundo
- Para las operaciones de firma digital donde se usan las funciones hash, esas velocidades son adecuadas. En otros entornos, como los de informática forense, esa velocidad podría ser crítica



# Ejemplos de hash con Hashcalc y Online

HashCalc

Data Format: Text string Data: Un hash es una huella digital de un documento

Key Format: Text string Key:

☐ HMAC

☒ MD5 3be78ac3c82822c4054e72a0c0c06a1d

☐ MD4

☒ SHA1 1b864f023126f46c0da6c5e640e693b5804cce1b

☒ SHA256 31f6c2222ff4c2dd5f6480949bf400aa96bbe8833bcf62312e33c76fa2635a13

☐ SHA384

☐ SHA512

☐ RIPEMD160

☐ PANAMA

☐ TIGER

☐ MD2

☐ ADLER32

☐ CRC32

☐ eDonkey/  
eMule

SlavaSoft

Calculate Close Help

<http://www.slavasoft.com/hashcalc/>

Online Tools

## SHA256

SHA256 online hash function

Un hash es una huella digital de un documento

Cuidado con códigos. Normalmente trabajan en código UTF-8 o Unicode, no ASCII extendido

Input type Text

Hash ☒ Auto Update

31f6c2222ff4c2dd5f6480949bf400aa96bbe8833bcf62312e33c76fa2635a13

Hash	File Hash
CRC-16	CRC-16
CRC-32	CRC-32
MD2	MD2
MD4	MD4
MD5	MD5
SHA1	SHA1
SHA224	SHA224
SHA256	SHA256
SHA384	SHA384
SHA512	SHA512
SHA512/224	SHA512/224
SHA512/256	SHA512/256
SHA3-224	SHA3-224
SHA3-256	SHA3-256
SHA3-384	SHA3-384
SHA3-512	SHA3-512
Keccak-224	Keccak-224
Keccak-256	Keccak-256
Keccak-384	Keccak-384
Keccak-512	Keccak-512

<https://emn178.github.io/online-tools/sha256.html>

# Ejemplo de hash con la utilidad de 7-zip

The image illustrates the process of calculating a hash for a file using 7-Zip. It consists of three main components:

- Class4crypt Progress Window:** A window titled "Class4crypt c4c6.9 Criptoanálisis a la cifra de Hill por Gauss-Jordan.mp4" showing the progress of a verification sum calculation. It displays a progress bar at 94% and a speed of 224 MB/s. The window includes buttons for "Segundo plano", "Pausa", and "Cancelar".
- 7-Zip Context Menu:** A right-click context menu for the file "Class4crypt c4c6.9 Criptoanálisis a la cifra de Hill por Gauss-Jordan.mp4" (4,634,492 KB). The "CRC SHA" option is selected, indicated by a blue arrow. The menu also includes options like "Reproducir", "Añadir a la lista de VLC", "Compartir con Skype", "Mover a OneDrive", "7-Zip", "Analizar los elementos seleccionados", "Compartir", "Abrir con", and "Restaurar versiones anteriores".
- Suma de verificación (CRC) Window:** A window displaying the calculated hash values for the file. It shows the file name, size, and the SHA256 hash.

**Suma de verificación (CRC)**

Nombre	Tamaño	SHA256
Class4crypt c4c6.9 Criptoanálisis a la cifra de Hill por Gauss-Jordan.mp4	4745719337 bytes (4525 MiB)	6E8ECE9549BAA0084FC323537B35A78CE597C20AB33B84F9E6B543FF6193C291



# Utilidad del hash en la criptografía

- Para realizar una firma digital  $F$ , habrá que realizar una cifra usando la clave privada del emisor con un algoritmo de criptografía asimétrica
- En RSA, si esta clave privada es  $d_E$ , el módulo de cifra es  $n_E$  y hay que firmar el mensaje  $M$ , la operación sería  $F = M^{d_E} \bmod n_E$
- Pero sabemos que los algoritmos de cifra asimétrica, que permiten la firma digital a diferencia de los algoritmos de cifra en simétrica, tienen velocidades de cifra/firma mucho menores que estos últimos
- Así, la firma no se hará sobre el mensaje sino sobre el resultado de aplicar un hash a ese mensaje o documento:  $F = h(M)^{d_E} \bmod n_E$
- Esto permite agilizar la operación de firma y, también, comprobar en recepción la integridad del documento recibido firmado digitalmente

# Principio del palomar, unicidad y colisiones

- Como es obvio, en tanto el hash o huella digital será un valor de bits mucho menor que los bits del archivo o mensaje al que se le aplica esa función, no se puede afirmar que el hash sea único
- Habrá una probabilidad de que dos archivos o mensajes diferentes tengan el mismo hash, lo que llamaremos una colisión
- Lógicamente esta probabilidad va a depender del tamaño del hash
- Esto se conoce como el principio del palomar (Johann Dirichlet), que dice que si hay más palomas que huecos en el palomar, entonces en alguno de estos huecos habrá más de una paloma



# La seguridad del resumen de un hash

- Si una función hash nos devuelve 4 bits, los  $2^4 = 16$  estados posibles serán 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, todos equiprobables
- La pregunta que nos puede preocupar es: ¿cuál sería la probabilidad de que dos mensajes distintos tengan igual función hash? Si se firma un hash  $h(M)$ , queremos estar seguros que sea el del mensaje original
- En este escenario la probabilidad será de  $1/2^4 = 1/16$
- Con una probabilidad de  $1/2^4$  (6,25% muy alta) podríamos tener dos mensajes diferentes (incluso contradictorios) con iguales hash
- Por ello, los hashes usan centenas de bits como resumen, hoy en día al menos de 256 bits, y esa probabilidad baja a  $1/2^{256}$  ... en la *práctica* 0

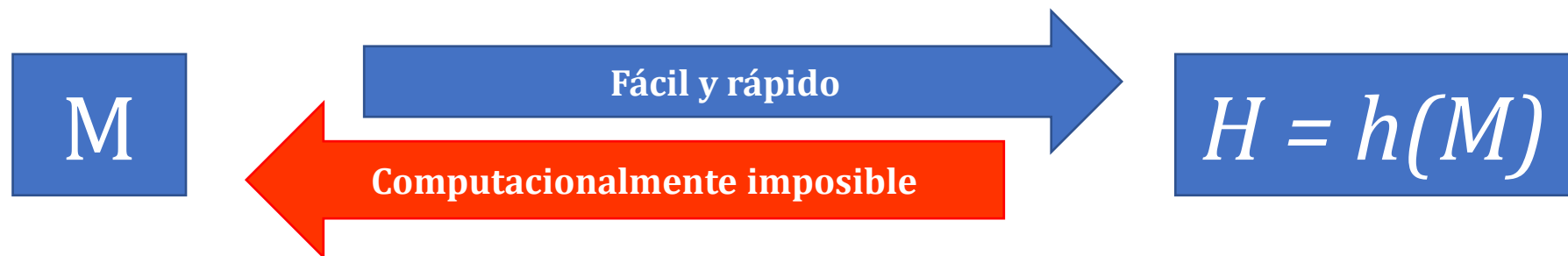
# Propiedades de las funciones hash (1/7)

## 1. Facilidad de cálculo

- Deberá ser fácil y rápido calcular  $h(M)$  a partir de  $M$

## 2. Unidireccionalidad

- Conocido un resumen  $H = h(M)$ , debe ser computacionalmente imposible o no factible encontrar el mensaje  $M$  a partir del resumen  $H$
- Aunque exista una forma para resolver el problema, el tiempo y los recursos necesarios para revertir  $h(M)$  deberán ser muy difíciles de cumplir



# Propiedades de las funciones hash (2/7)

## 3. Compresión

- A partir de un mensaje  $M$  de cualquier longitud, el resumen  $H = h(M)$  debe tener una longitud fija
- En la firma digital, lo normal es que la longitud de  $h(M)$  sea menor que el mensaje  $M$  a firmar
- Aunque tenga cierto parecido, una función hash no es lo mismo que la función zip para compresión de archivos
- La función zip recodifica el archivo mediante un codificador óptimo y, por tanto, optimiza el número de bits comprimiendo el archivo todo lo que puede, usando entre otras cosas la redundancia del lenguaje
- En cambio una función hash devuelve un resumen de longitud fija
- Si  $M$  tiene menos bits que el resultado del hash, como sería el caso del uso de hashes en contraseñas, lógicamente no se cumplirá esta propiedad



# Propiedades de las funciones hash (3/7)

## 4. Difusión o efecto avalancha

- El hash  $h(M)$  debe ser una función compleja de todos los bits del mensaje  $M$
- Por tanto, si se modifica un solo bit del mensaje  $M$ , el hash  $h(M)$  debería cambiar en media la mitad de sus bits
- MD5 sobre  $M_1$  y  $M_2$  que difieren en 1 bit (**B** = 01000010 y **C** = 01000011)
- $M_1$  = BEBA **C**OCA COLA     $h(M_1)$  = EBC97CD6472A3FDBE53B09F6E2BA8687
- $M_2$  = BEBA **B**OCA COLA     $h(M_2)$  = 967A99F55CF64973C553374C08A7BD72

$h(M_1)$     1110 1011 1100 1001 0111 1100 1101 0110 0100 0111 0010 1010  
             0011 1111 1101 1011 1110 0101 0011 1011 0000 1001 1111 0110  
             1110 0010 1011 1010 1000 0110 1000 0111

$h(M_2)$     1001 0110 0111 1010 1001 1001 1111 0101 0101 1100 1111 0110  
             0100 1001 0111 0011 1100 0101 0101 0011 0011 0111 0100 1100  
             0000 1000 1010 0111 1011 1101 0111 0010

De los 128 bits de MD5,  
los 70 bits marcados en  
rojo en  $h(M_2)$  indican  
cambios que se han  
producido entre los  
hashes de  $h(M_1)$  y  $h(M_2)$

# Propiedades de las funciones hash (4/7)

## 5. No predictibilidad

- La fortaleza de las funciones hash reside en la no predictibilidad de la salida obtenida, o del valor resumen o hash
- Es decir, si se calculan los hashes SHA-256 de los números 0, 1, 2 y 3, no será posible predecir cuál es el valor del hash SHA-256 del número 4

```
SHA-256 0: 5FECEB66FFC86F38D952786C6D696C79C2DBC239DD4E91B46729D73A27FB57E9
SHA-256 1: 6B86B273FF34FCE19D6B804EFF5A3F5747ADA4EAA22F1D49C01E52DDB7875B4B
SHA-256 2: D4735E3A265E16EEE03F59718B9B5D03019C07D8B6C51F90DA3A666EEC13AB35
SHA-256 3: 4E07408562BEDB8B60CE05C1DECFE3AD16B72230967DE01F640B7E4729B49FCE
SHA-256 4: ¿?
```

- Teóricamente, una función hash debería comportarse como una función de generación de números aleatorios, conocida habitualmente como un oráculo aleatorio (*random oracle*), especie de caja negra que responde a cada consulta con una respuesta realmente aleatoria



# Propiedades de las funciones hash (5/7)

## 6. Resistencia a preimagen (o primera preimagen)

- Dado un valor resumen  $H = h(M)$  debe de ser computacionalmente imposible encontrar una preimagen  $M$  para ese valor  $H$
- Es decir, será computacionalmente difícil que, conocido  $H$ , se encuentre un mensaje  $M$  tal que  $h(M) = H$  (unidireccionalidad)
- La resistencia a primera preimagen depende de la longitud  $n$  del resumen proporcionado por la función hash
- Mediante técnicas de fuerza bruta, en media se podría obtener una preimagen de un hash  $H$  después de  $2^{(n-1)}$  intentos
- Por ejemplo
  - Para MD5  $2^{(128-1)} = 2^{127} = 1,701 \times 10^{38}$  intentos
  - Para SHA-1  $2^{(160-1)} = 2^{159} = 7,307 \times 10^{47}$  intentos
  - Para SHA-256  $2^{(256-1)} = 2^{255} = 5,789 \times 10^{76}$  intentos

# Propiedades de las funciones hash (6/7)

## 7. Resistencia simple a colisiones (o segunda preimagen)

- El objetivo de toda función hash es que sea computacionalmente difícil que, conocido el mensaje de entrada  $M$ , se encuentre un mensaje de entrada  $M'$  (distinto a  $M$ ) de forma que  $h(M) = h(M')$
- Así, la resistencia simple a colisiones pretende evitar que un potencial atacante que disponga de un mensaje  $M$  y, por tanto, su correspondiente hash  $h(M)$ , pueda encontrar otro mensaje  $M'$  cuyo hash  $h(M')$  sea el mismo que el anterior, lo que le permitiría poder reemplazar un mensaje  $M$  por otro  $M'$
- Hay que recordar que lo que se firma son los hashes, no mensajes ni archivos
- La probabilidad de encontrar una segunda preimagen para una función hash es equivalente a la de encontrar una primera preimagen, en media  $1/2^{n-1}$
- Por lo tanto, cualquier función hash resistente a segunda preimagen será resistente también a primera preimagen

# Propiedades de las funciones hash (7/7)

## 8. Resistencia fuerte a colisiones

- Será computacionalmente difícil encontrar un par de mensajes al azar ( $M, M'$ ) de forma que sus hashes  $h(M) = h(M')$  sean iguales
- Esta resistencia fuerte está relacionada con el ataque conocido como paradoja del cumpleaños
- Si un hash tiene  $n$  bits, la probabilidad media de que prospere un ataque por paradoja del cumpleaños será igual a  $1/(2^{n/2})$ . Es decir, deberíamos hacer una media de  $2^{n/2}$  intentos, una disminución de cómputo muy considerable
- Por ejemplo
  - Para MD5  $2^{(128/2)} = 2^{64} = 1,845 \times 10^{19}$  intentos
  - Para SHA-1  $2^{(160/2)} = 2^{80} = 1,209 \times 10^{24}$  intentos
  - Para SHA-256  $2^{(256/2)} = 2^{128} = 3,403 \times 10^{38}$  intentos
- ¿Cómo puede llevarse a cabo un ataque mediante la paradoja del cumpleaños?



# La paradoja del cumpleaños

- Paradójico: “Hecho o expresión aparentemente contrarios a la lógica”
- Se conoce como paradoja del cumpleaños al problema matemático que nos dice que la confianza o probabilidad mayor que el 50% de que dos personas al azar estén de cumpleaños el mismo día se supera dentro de un grupo de solo 23 personas, y esta probabilidad llega al 99,7% si ese grupo cuenta con 57 personas, siempre probabilístico
- No es una paradoja porque no contradice la lógica sino la intuición
- Explicación: con un mapa de 365 días (sin contar año bisiesto) la primera persona consultada marca una fecha, estando los 365 días libres. La segunda persona consultada cuenta ahora solo 365- 1 días, y así sucesivamente. Es una serie, en donde el escenario con días libres va disminuyendo a cada nueva consulta (ver bibliografía)

# Ataque por paradoja de cumpleaños a hash

- Se crean  $2^{n/2}$  **textos verdaderos** y sus correspondientes  $2^{n/2}$  **textos falsos**, hasta que los hashes de texto verdadero y falso colisionen, usando por ejemplo estos sinónimos marcados entre paréntesis {} para un hash de 20 bits, con  $2^{20} = 1.048.576$  valores

Estimado {Querido} amigo {compañero}: Texto Verdadero  
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.

Aquí  $2^{10} = 1.024$  permutaciones de varios **textos verdaderos**

Estimado {Querido} amigo {compañero}: Texto Falso  
Te envío {hago llegar} esta carta {nota} para indicarte {darte a conocer} que tu número {cupón} **NO** ha sido premiado {agraciado} con el premio principal {gordo} de la lotería.  
Te envío {Recibe} un cordial saludo {fuerte abrazo}, María.

Aquí  $2^{10} = 1.024$  permutaciones de varios **textos falsos**

# Más información en píldoras Thoth



<https://www.youtube.com/watch?v=FRBIc0udwv0>

# Conclusiones de la Lección 7.1

- La función hash es una huella digital de un documento, representada por un número fijo de bits y normalmente especificada en hexadecimal
- Un hash no es un algoritmo de cifra al carecer de clave y tampoco debe confundirse con funciones de compresión como los algoritmos zip
- Entre las funciones hash destacan MD5, SHA-1, familia SHA-2 y SHA-3
- Los hashes deben cumplir un conjunto de propiedades como son la facilidad de cálculo, unidireccionalidad, difusión, no predictibilidad, resistencia a primera preimagen, resistencia simple a colisiones o segunda preimagen y resistencia fuerte a colisiones
- El talón de Aquiles de un hash es el ataque por paradoja del cumpleaños



# Lectura recomendada

- Guion píldora formativa Thoth nº 43, ¿Qué son y para qué sirven las funciones hash?, Jorge Ramió, 2017
  - <https://www.criptored.es/thoth/material/texto/pildora043.pdf>
- ¿Cómo de seguro es la seguridad de 256 bits?, 3Blue1Brown, YouTube, 2017
  - [https://www.youtube.com/watch?v=S9JGmA5\\_unY](https://www.youtube.com/watch?v=S9JGmA5_unY)
- Función hash, Principio del palomar y Random oracle, Wikipedia
  - [https://es.wikipedia.org/wiki/Funci%C3%B3n\\_hash](https://es.wikipedia.org/wiki/Funci%C3%B3n_hash)
  - [https://es.wikipedia.org/wiki/Principio\\_del\\_palomar](https://es.wikipedia.org/wiki/Principio_del_palomar)
  - [https://en.wikipedia.org/wiki/Random\\_oracle](https://en.wikipedia.org/wiki/Random_oracle)
- La paradoja del cumpleaños, Estadística para todos
  - <https://www.estadisticaparatodos.es/taller/cumpleanos/cumpleanos.html>
- Birthday problem, Wikipedia
  - [https://en.wikipedia.org/wiki/Birthday\\_problem](https://en.wikipedia.org/wiki/Birthday_problem)

# Class4crypt c4c7.2

## Módulo 7. Funciones hash

### Lección 7.2. Función hash MD5: estructura y operaciones

7.2.1. Construcción de hashes

7.2.2. La estructura de Merkle-Damgård

7.2.3. Cronología de los algoritmos de hash

7.2.4. Características de la función hash MD5

7.2.5. Funcionamiento del hash MD5

7.2.6. MD5 por dentro con el software CriptoRes

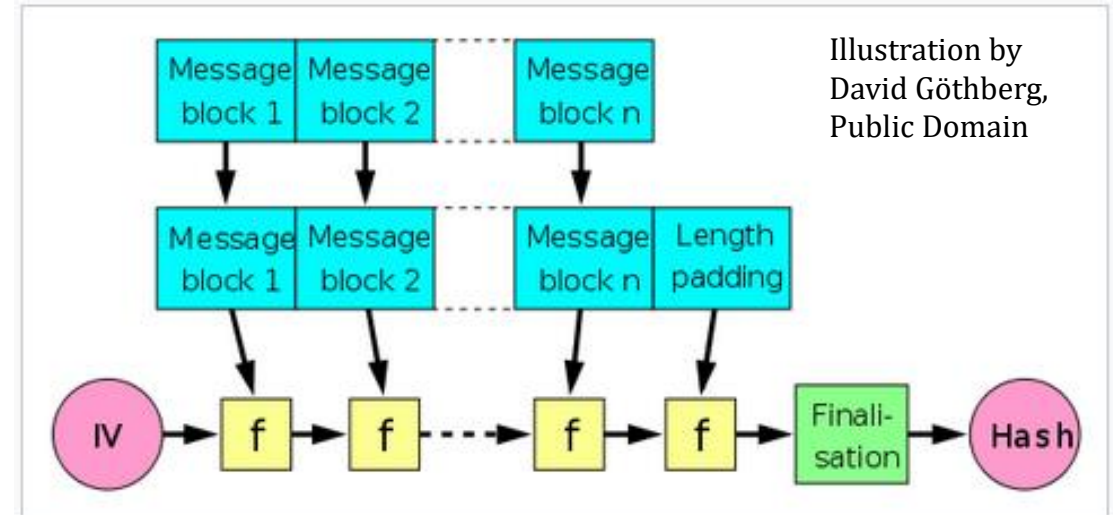
Class4crypt c4c7.2 Función hash MD5: estructura y operaciones  
<https://www.youtube.com/watch?v=rkBVDhfCy1c>

# Construcción de hashes

- Funciones hash iterativas
  - Procesamiento de un mensaje en bloques, aplicando el mismo algoritmo de manera consecutiva o iterativa, que se aplica desde años 80
  - Funciones hash basadas en compresión
    - Transformación de la entrada en una salida de menor tamaño (lo normal)
    - Usa una estructura o construcción de Merkle-Damgård
    - Funciones resumen típicas: MD5, SHA-1 y SHA-2
  - Funciones hash basadas en permutaciones
    - Transformación de la entrada en una salida de igual tamaño
    - Usa funciones esponja (sponge)
    - Función resumen típica: SHA-3 (Keccak)

# La estructura de Merkle-Damgård

- Se construyen bloques de 512 bits a partir del mensaje M
- Al último bloque se le añade siempre bits de relleno, al menos 1 byte, y se indica el tamaño o longitud de M
- Cada bloque de 512 bits se usa en un conjunto de operaciones  $f$  con puertas lógicas y con un vector IV de  $x$  palabras de 32 bits
- Para MD5  $x = 4$ , para SHA-1  $x = 5$  y para SHA-256  $x = 8$
- El hash es el último valor del vector IV después del último bloque



# Añadiendo relleno y longitud en el hash

- Normalmente, el relleno se hace con bytes 0, es decir 0x 00
- Pero no se usa porque esto podría acarrear el siguiente problema
- Supongamos un hash que forme bloques de 8 bytes sobre el texto  $M = \text{Hola amigo} = 0x\ 486f6c6120616d69\ 676f$ , longitud 10 bytes
- Relleno de ceros:  $0x\ 486f6c6120616d69\ 676f000000000000$
- Pero el hash de  $M = 0x\ 486f6c6120616d69\ 676f$  sería igual al hash de  $M = 0x\ 486f6c6120616d69\ 676f00$ , por ejemplo, lo cual es inaceptable. Por eso se cambia a 1 el primer bit de los bytes de relleno  $0x\ 80 = 1000\ 0000$  y los demás son  $0x = 00 = 0000\ 0000$
- Además, para evitar ataques por extensión de la longitud, se reserva un bloque de bytes al final para indicar la longitud de  $M$

# Cronología de los algoritmos de hash

- **N-Hash:** Nippon Telephone and Telegraph, 1990. Resumen de 128 bits
- **Snefru:** Ralph Merkle, 1990. Resúmenes entre 128 y 256 bits. Ha sido criptoanalizado y es lento
- **MD4:** Ronald L. Rivest, 1990. Resumen de 128 bits
- **Haval:** Yuliang Zheng, Josef Pieprzyk y Jennifer Seberry, 1992. Resúmenes hasta 256 bits. Admite 15 configuraciones diferentes.
- **RIPEMD:** Comunidad Europea, RACE, 1992. Resumen de 160 bits
- **MD5:** Ronald L. Rivest, 1991. Resumen de 128 bits. Mejoras sobre MD2 y MD4 (1990), más lento pero con mayor nivel de seguridad
- **SHA-0** (o SHA): National Security Agency (NSA), 1993. Resumen de 160 bits. Vulnerable y reemplazado por SHA-1
- **SHA-1:** National Security Agency (NSA), 1994. Similar a MD5 pero con resumen de 160 bits
- **Tiger:** Ross Anderson, Eli Biham, 1996. Resúmenes hasta 192 bits. Optimizado para máquinas de 64 bits (Alpha)
- **Panama:** John Daemen, Craig Clapp, 1998. Resúmenes de 256 bits. Trabaja en modo función hash o como cifrador de flujo
- **SHA-2:** National Security Agency (NSA), 2001-2004. Resúmenes entre 224 y 512 bits (224, 256, 384, o 512). Mejoras sobre SHA-1
- **SHA-3** (Keccak): Guido Bertoni, Joan Daemen, Michaël Peeters y Gilles Van Assche, 2015. Resúmenes arbitrarios estándar (224, 256, 384, o 512). Más robusto que SHA-2

# Algoritmo MD5 Message Digest 5 (1/2)

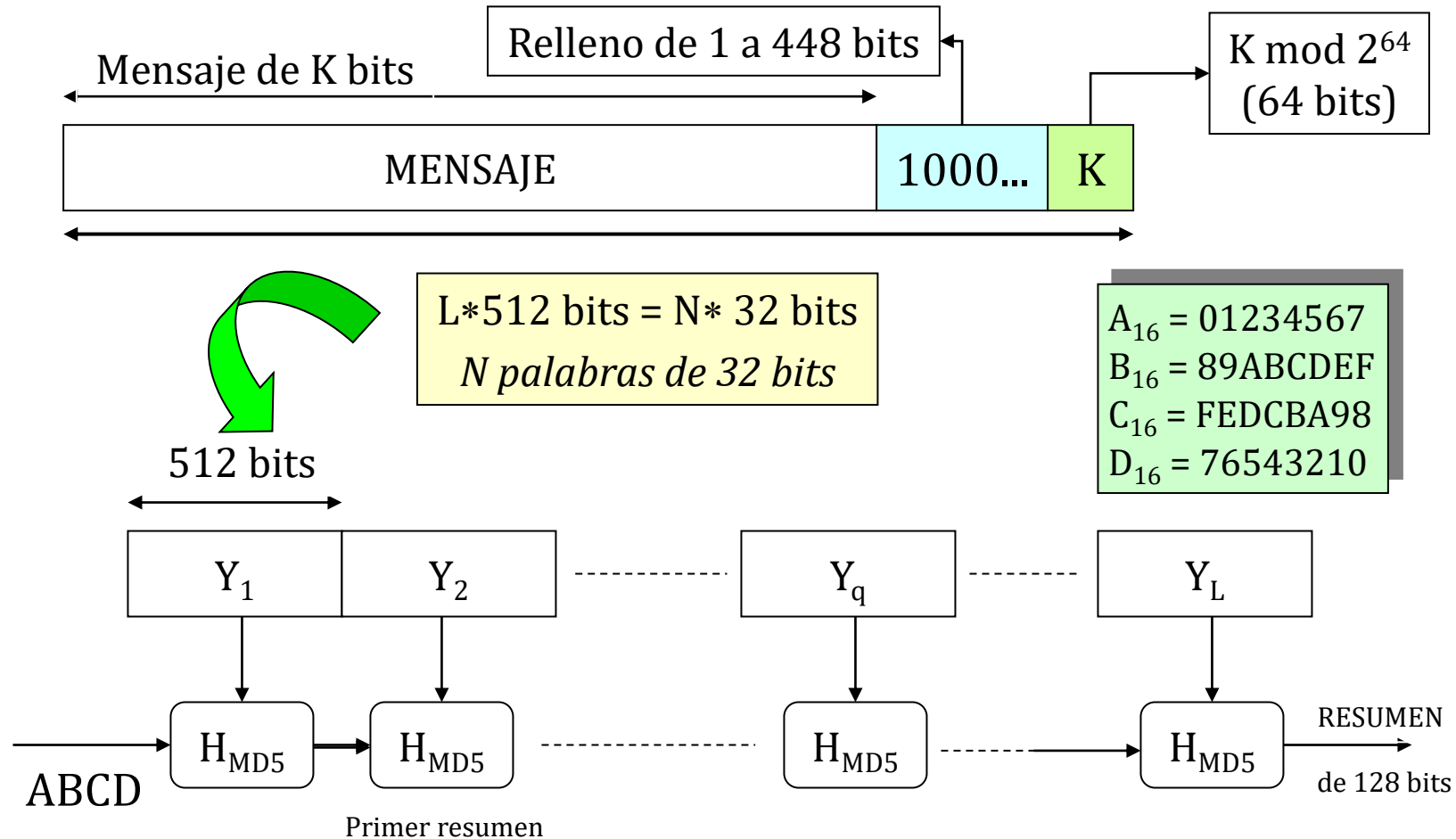
- MD5 fue creado por Ronald Rivest en 1991 y presenta algunas mejoras con respecto a MD2 y MD4 del mismo autor (1990)
- Esta función ya está obsoleta desde mediados de 2005. No obstante, se sigue utilizando en diferentes aplicaciones locales, aunque no en Internet. Es interesante su estudio dada la sencillez del algoritmo, su rapidez y su generalidad
- Procesa bloques de 512 bits con una salida de 128 bits
- Expande el mensaje hasta una longitud 64 bits inferior a un múltiplo de 512 bits. Para el relleno, añade un 1 seguido de tantos 0 como sean necesarios y reserva los últimos 64 bits para añadir información sobre la longitud del mensaje



# Algoritmo MD5 Message Digest 5 (2/2)

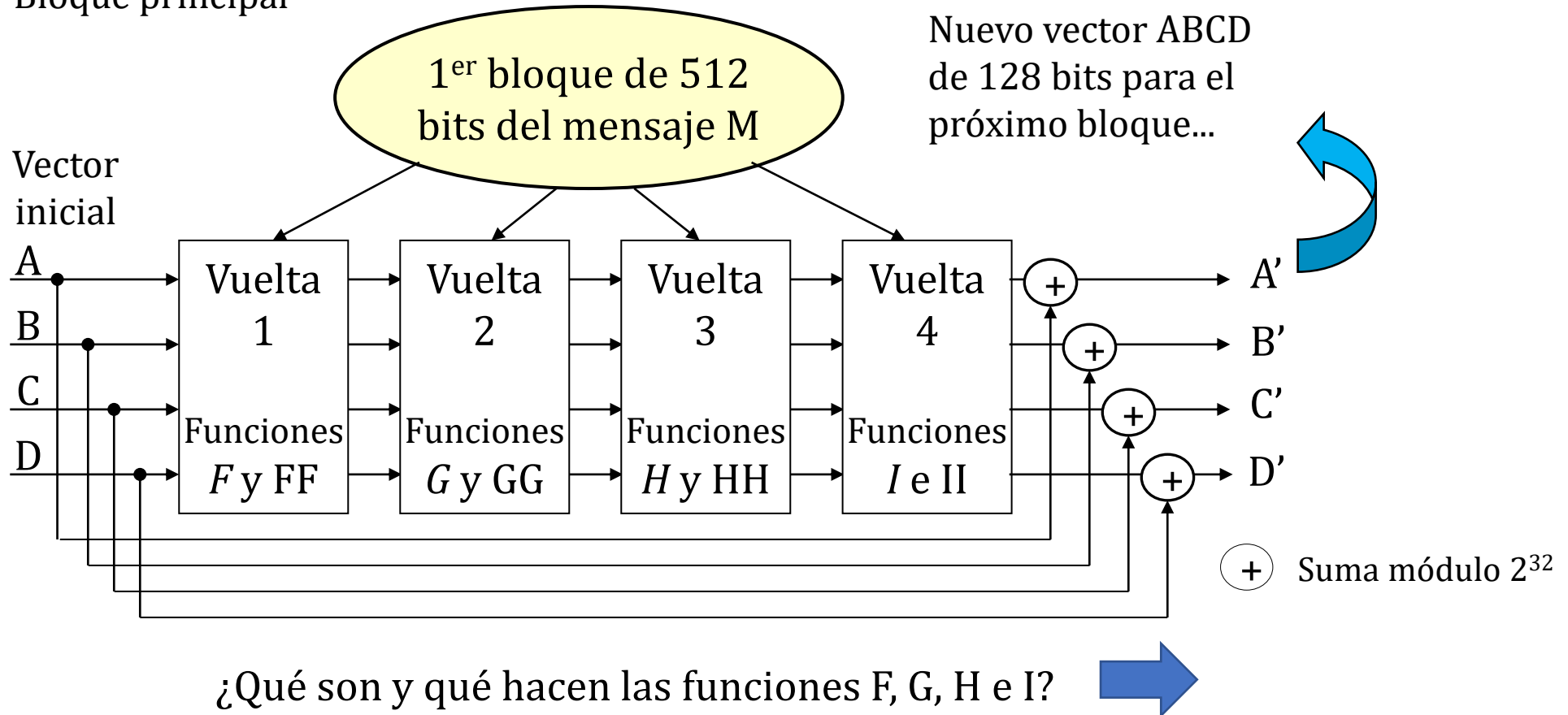
- El algoritmo comienza con cuatro vectores iniciales (IV) ABCD de 32 bits cada uno, cuyo valor inicial no es secreto. A estos vectores y al primer bloque de 512 bits de M se le aplican 64 operaciones de 32 bits con puertas lógicas, cuyo carácter es no lineal
- Las 64 operaciones se engloban en 4 vueltas o rondas
- Como resultado de estas operaciones, se obtienen cuatro nuevos vectores A'B'C'D' que serán la entrada IV' para el segundo bloque de 512 bits, repitiéndose esto con los restantes bloques de M
- La última salida de IV corresponde al resumen final  $H = h(M)$
- Definido en la RFC 1321: <https://tools.ietf.org/html/rfc1321>

# Esquema de MD5



# Bloque principal de MD5

Bloque principal



# Funciones F, G, H e I en MD5

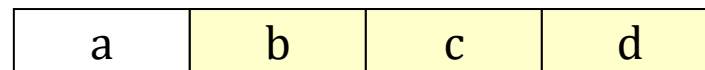
Vector inicial ABCD

$A_{16} = 01234567$

$B_{16} = 89ABCDEF$

$C_{16} = FEDCBA98$

$D_{16} = 76543210$



128 bits

función no lineal

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

**$F(x, y, z)$**

$(x \text{ AND } y) \text{ OR } (\text{NOT } x \text{ AND } z)$

**$G(x, y, z)$**

$(x \text{ AND } z) \text{ OR } (y \text{ AND } \text{NOT } z)$

**$H(x, y, z)$**

$x \text{ XOR } y \text{ XOR } z$

**$I(x, y, z)$**

$y \text{ XOR } (x \text{ OR } \text{NOT } z)$

**$F(b, c, d)$**

$(b \text{ AND } c) \text{ OR } (\text{NOT } b \text{ AND } d)$

**$G(b, c, d)$**

$(b \text{ AND } d) \text{ OR } (c \text{ AND } \text{NOT } d)$

**$H(b, c, d)$**

$b \text{ XOR } c \text{ XOR } d$

**$I(b, c, d)$**

$c \text{ XOR } (b \text{ OR } \text{NOT } d)$

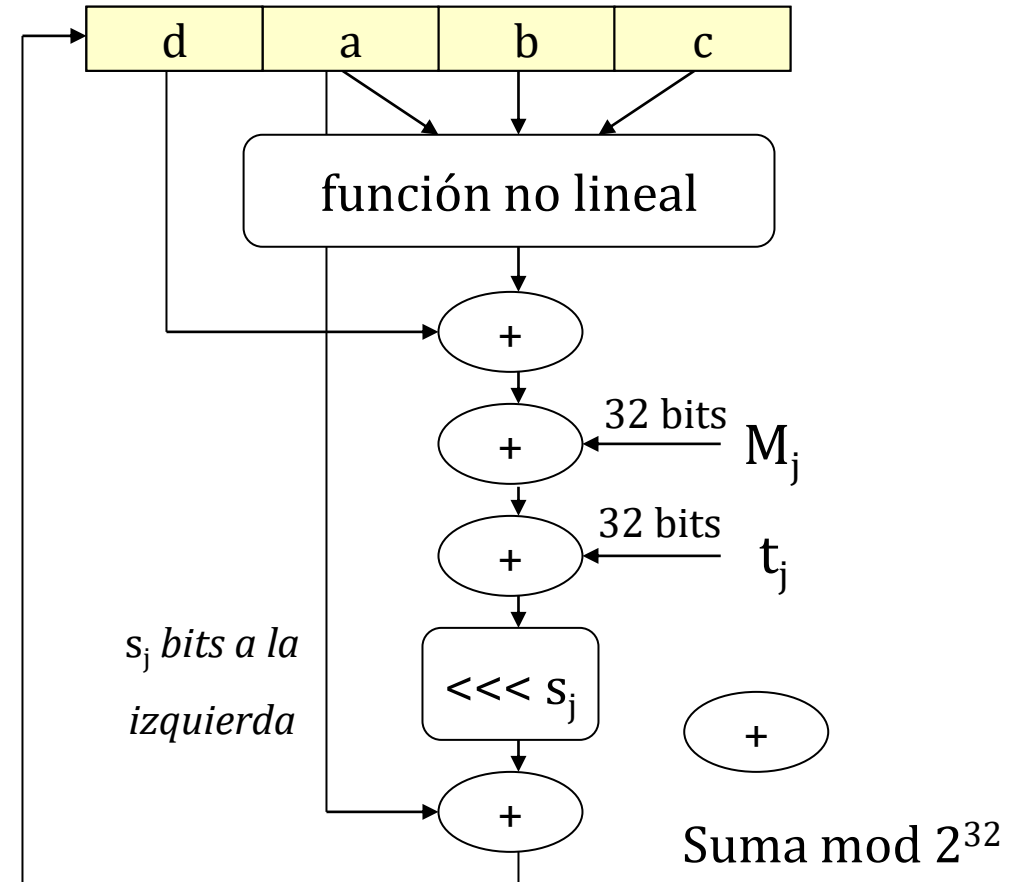
# Funcionamiento de MD5

Desplazamiento del registro →

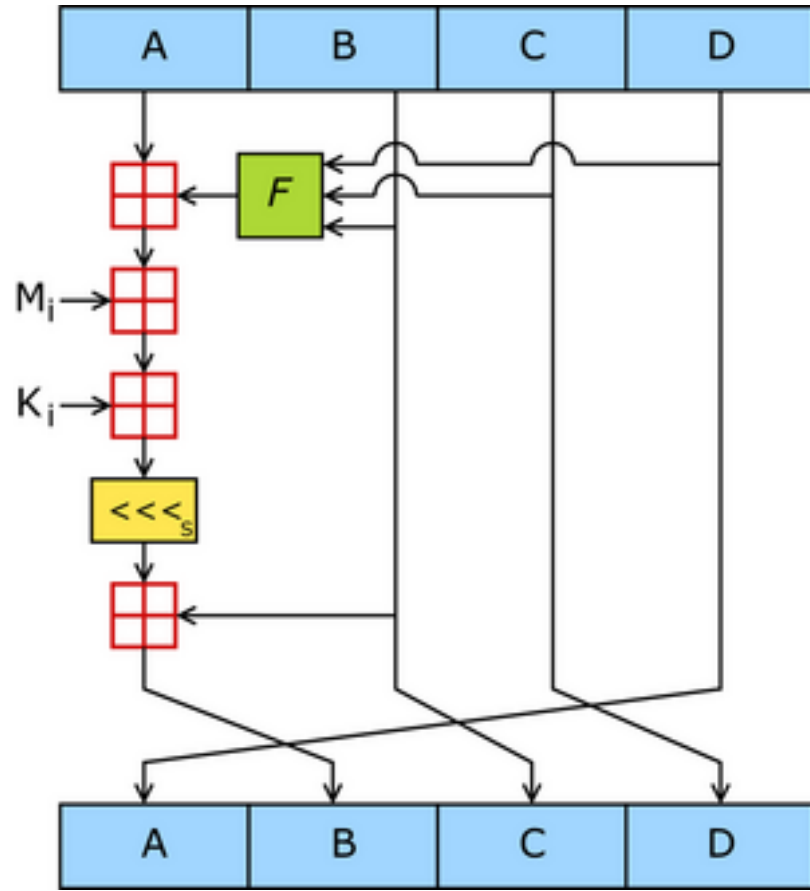
Situación luego del desplazamiento

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  $4 \cdot 16 = 64$  vueltas o rondas para cada uno de los bloques de 512 bits de M



# Funciones en cada vuelta de MD5



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

Dependiendo de la ronda, F puede ser F, G, H, I:

F = (**B** AND **C**) OR (NOT **B** AND **D**) 1ª ronda

G = (**B** AND **D**) OR (**C** AND NOT **D**) 2ª ronda

H = (**B** XOR **C** XOR **D**) 3ª ronda

I = (**C** XOR (**B** OR NOT **D**)) 4ª ronda

 es una suma mod  $2^{32}$ , no un XOR

- Funciones en cada una de las 4 vueltas o rondas
- FF (a,b,c,d,Mj,tj,s)  $a = b + ((a + F(b,c,d) + Mj + tj) \lll s)$
- GG (a,b,c,d,Mj,tj,s)  $a = b + ((a + G(b,c,d) + Mj + tj) \lll s)$
- HH (a,b,c,d,Mj,tj,s)  $a = b + ((a + H(b,c,d) + Mj + tj) \lll s)$
- II (a,b,c,d,Mj,tj,s)  $a = b + ((a + I(b,c,d) + Mj + tj) \lll s)$

# Operaciones en 1ª y 2ª vueltas de MD5

## Primera vuelta

FF (a, b, c, d, M<sub>j</sub>, t<sub>j</sub>, s<sub>j</sub>)

FF(a, b, c, d, M<sub>0</sub>, D76AA478, 7)  
FF(d, a, b, c, M<sub>1</sub>, E8C7B756, 12)  
FF(c, d, a, b, M<sub>2</sub>, 242070DB, 17)  
FF(b, c, d, a, M<sub>3</sub>, C1BDCEEE, 22)  
FF(a, b, c, d, M<sub>4</sub>, F57C0FAF, 7)  
FF(d, a, b, c, M<sub>5</sub>, 4787C62A, 12)  
FF(c, d, a, b, M<sub>6</sub>, A8304613, 17)  
FF(b, c, d, a, M<sub>7</sub>, FD469501, 22)  
FF(a, b, c, d, M<sub>8</sub>, 698098D8, 7)  
FF(d, a, b, c, M<sub>9</sub>, 8B44F7AF, 12)  
FF(c, d, a, b, M<sub>10</sub>, FFFF5BB1, 17)  
FF(b, c, d, a, M<sub>11</sub>, 895CD7BE, 22)  
FF(a, b, c, d, M<sub>12</sub>, 6B901122, 7)  
FF(d, a, b, c, M<sub>13</sub>, FD987193, 12)  
FF(c, d, a, b, M<sub>14</sub>, A679438E, 17)  
FF(b, c, d, a, M<sub>15</sub>, 49B40821, 22)

## Segunda vuelta

GG (a, b, c, d, M<sub>j</sub>, t<sub>j</sub>, s<sub>j</sub>)

GG(a, b, c, d, M<sub>1</sub>, F61E2562, 5)  
GG(d, a, b, c, M<sub>6</sub>, C040B340, 9)  
GG(c, d, a, b, M<sub>11</sub>, 265E5A51, 14)  
GG(b, c, d, a, M<sub>0</sub>, E9B6C7AA, 20)  
GG(a, b, c, d, M<sub>5</sub>, D62F105D, 5)  
GG(d, a, b, c, M<sub>10</sub>, 02441453, 9)  
GG(c, d, a, b, M<sub>15</sub>, D8A1E681, 14)  
GG(b, c, d, a, M<sub>4</sub>, E7D3FBC8, 20)  
GG(a, b, c, d, M<sub>9</sub>, 21E1CDE6, 5)  
GG(d, a, b, c, M<sub>14</sub>, C33707D6, 9)  
GG(c, d, a, b, M<sub>3</sub>, F4D50D87, 14)  
GG(b, c, d, a, M<sub>8</sub>, 455A14ED, 20)  
GG(a, b, c, d, M<sub>13</sub>, A9E3E905, 5)  
GG(d, a, b, c, M<sub>2</sub>, FCEFA3F8, 9)  
GG(c, d, a, b, M<sub>7</sub>, 676F02D9, 14)  
GG(b, c, d, a, M<sub>12</sub>, 8D2A4C8A, 20)



# Operaciones en 3ª y 4ª vueltas de MD5

Tercera vuelta

HH (a, b, c, d, M<sub>j</sub>, t<sub>j</sub>, s<sub>j</sub>)

HH(a, b, c, d, M<sub>5</sub>, FFFA3942, 4)  
HH(d, a, b, c, M<sub>8</sub>, 8771F681, 11)  
HH(c, d, a, b, M<sub>11</sub>, 6D9D6122, 16)  
HH(b, c, d, a, M<sub>14</sub>, FDE5380C, 23)  
HH(a, b, c, d, M<sub>1</sub>, A4BEEA44, 4)  
HH(d, a, b, c, M<sub>4</sub>, 4BDECFA9, 11)  
HH(c, d, a, b, M<sub>7</sub>, F6BB4B60, 16)  
HH(b, c, d, a, M<sub>10</sub>, BEBFBC70, 23)  
HH(a, b, c, d, M<sub>13</sub>, 289B7EC6, 4)  
HH(d, a, b, c, M<sub>0</sub>, EAA127FA, 11)  
HH(c, d, a, b, M<sub>3</sub>, D4EF3085, 16)  
HH(b, c, d, a, M<sub>6</sub>, 04881D05, 23)  
HH(a, b, c, d, M<sub>9</sub>, D9D4D039, 4)  
HH(d, a, b, c, M<sub>12</sub>, E6DB99E5, 11)  
HH(c, d, a, b, M<sub>15</sub>, 1FA27CF8, 16)  
HH(b, c, d, a, M<sub>2</sub>, C4AC5665, 23)

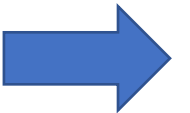
Cuarta vuelta

II (a, b, c, d, M<sub>j</sub>, t<sub>j</sub>, s<sub>j</sub>)

II(a, b, c, d, M<sub>0</sub>, F4292244, 6)  
II(d, a, b, c, M<sub>7</sub>, 411AFF97, 10)  
II(c, d, a, b, M<sub>14</sub>, AB9423A7, 15)  
II(b, c, d, a, M<sub>5</sub>, FC93A039, 21)  
II(a, b, c, d, M<sub>12</sub>, 655B59C3, 6)  
II(d, a, b, c, M<sub>3</sub>, 8F0CCC92, 10)  
II(c, d, a, b, M<sub>10</sub>, FFEFF47D, 15)  
II(b, c, d, a, M<sub>1</sub>, 85845DD1, 21)  
II(a, b, c, d, M<sub>8</sub>, 6FA87E4F, 6)  
II(d, a, b, c, M<sub>15</sub>, FE2CE6E0, 10)  
II(c, d, a, b, M<sub>6</sub>, A3014314, 15)  
II(b, c, d, a, M<sub>13</sub>, 4E0811A1, 21)  
II(a, b, c, d, M<sub>4</sub>, F7537E82, 6)  
II(d, a, b, c, M<sub>11</sub>, BD3AF235, 10)  
II(c, d, a, b, M<sub>2</sub>, 2AD7D2BB, 15)  
II(b, c, d, a, M<sub>9</sub>, EB86D391, 21)

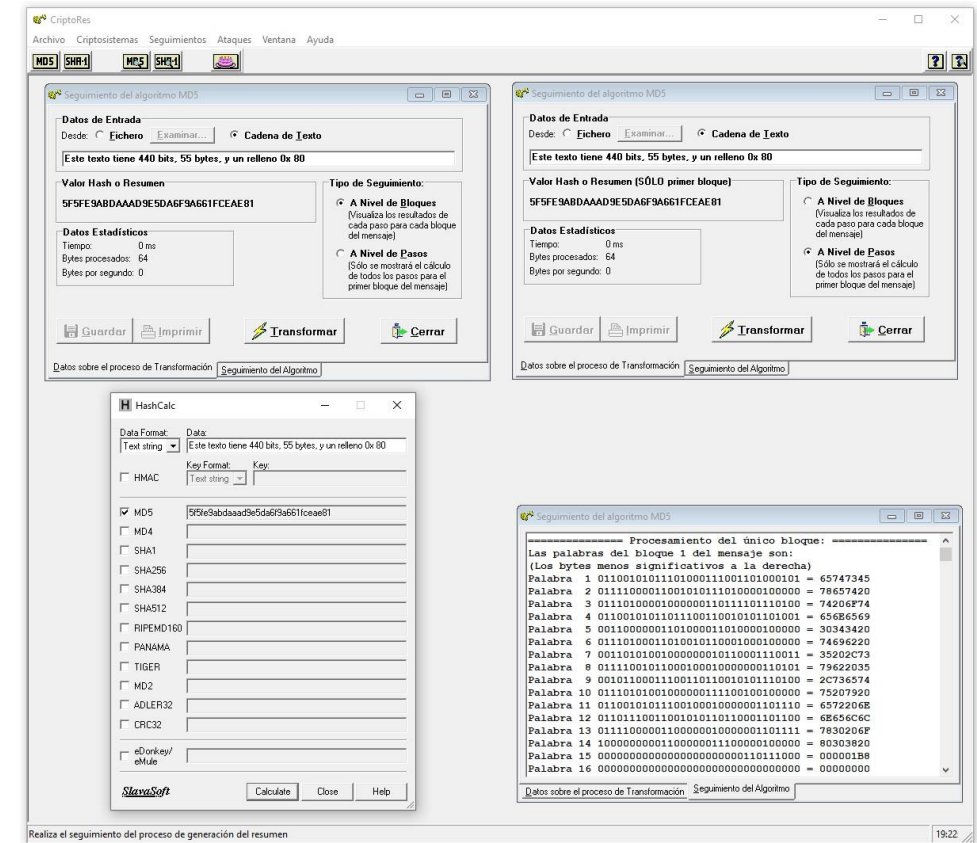
# MD5 por dentro con CriptoRes

- Mediante la pestaña seguimiento de CriptoRes, podrá observar los bytes de relleno y los 64 bits (dos últimas palabras de 32 bits) reservadas para la longitud del mensaje al final de éste
- Siempre existirá un relleno. Por ejemplo, si no hay entrada, el hash de nulo se calcula sobre el mensaje 0x 80 00 00 00 00 ... 00 00 00
- Ejercicios con Criptores
  - Si el mensaje tiene un tamaño de 55 bytes (440 bits), como habrá que reservar los últimos 64 bits para la longitud, se añadirá 1 byte de relleno 0x 80 al primer y único bloque M ( $55 + 1 + 8 = 64$  bytes = 512 bits)
  - Si M tiene 56 bytes (448 bits), entonces se añadirá relleno con 0x 80 y se forzará a que haya un segundo bloque sólo con rellenos, dejando los últimos 64 bits reservados para la longitud del mensaje



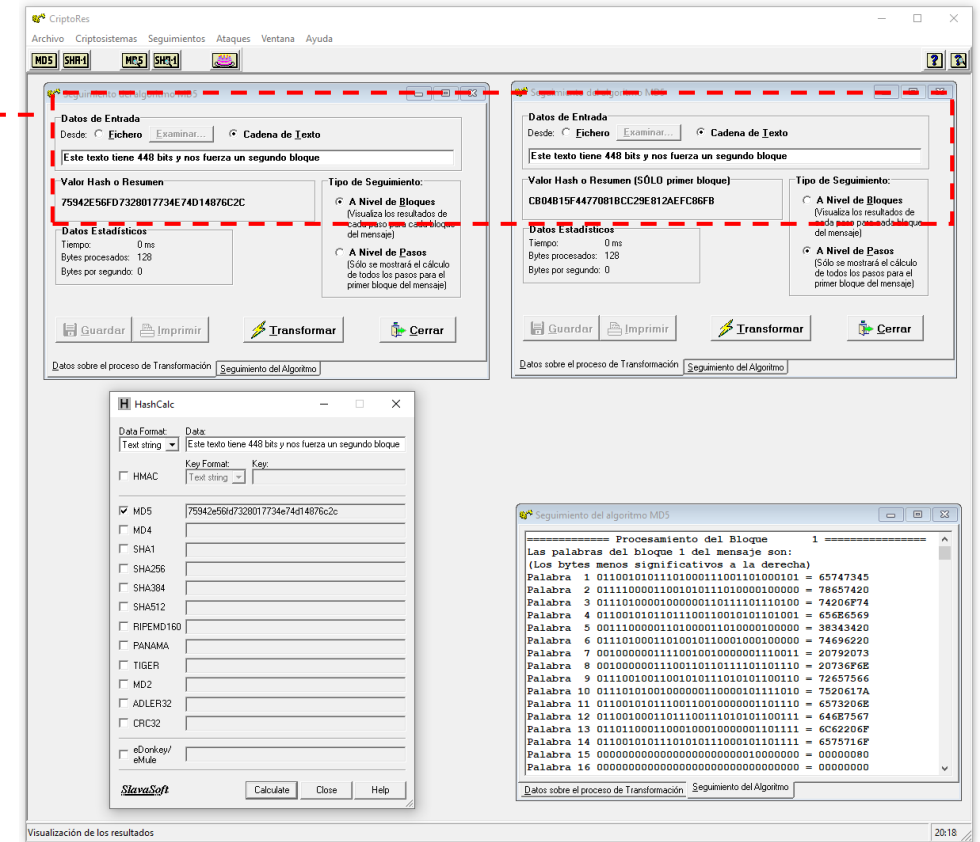
# Mínimo relleno 0x 80 en primer bloque

- Vamos a obtener el hash MD5 del texto M con 55 bytes, 440 bits, que se indica
- $M = \text{Este texto tiene 440 bits, 55 bytes, y un relleno } 0x\ 80$
- $h(M) = 5F5FE9ABDAAAD9E5DA6F9A661FCEAE81$
- Tenemos un único bloque de  $512 - 64 = 448$  bits para texto + relleno
- Aparece el texto M en hexadecimal y lectura Little Endian en cada palabra de 32 bits, y al final de la palabra 14 vemos un único byte 0x 80 de relleno
- Por las palabras 15 y 16, la longitud del M es  $0x\ 1B8 = 440$  en decimal, los 440 bits del mensaje
- Todo correcto. ¿Qué pasará si añadimos 1 byte?



# Forzando rellenos en el segundo bloque

- En este software se ha limitado ver el relleno y la longitud del mensaje solo en el primer bloque
- Vamos a obtener el hash MD5 del texto M con 56 bytes, 448 bits, que se indica
- M = Este texto tiene 448 bits y nos fuerza un segundo bloque
- $h(M) = 75942E56FD7328017734E74D14876C2C$
- Nos fuerza a que se procesen ahora dos bloques
- El relleno 0x 80 comienza en la palabra 15 del primer bloque, y sigue en todo el segundo bloque con 0x 00 hasta la palabra 14, ya que las palabras 15 y 16 de este segundo último bloque estarán reservadas para indicar la longitud del mensaje M



Hash 1er bloque y hash dos bloques diferentes

# Más información en píldoras Thoth

Píldora formativa Thoth 44 ¿Cómo funciona el hash ...

MD5

Message Digest 5

año 1991

md5 Hash	Version
8468300a61ea1e3b7607f46f7643b57a	ubuntu-11.04-alternate-amd64 iso
afa29ce3dccc0ab12332036dc17d9e1f	ubuntu-11.04-alternate-i386 iso
7de611b50c283c1755b4007a4feb0379	ubuntu-11.04-desktop-amd64 iso
8b1085bed496b82ef1485ef19074c281	ubuntu-11.04-desktop-i386 iso
355ca2417522cb4a77e0295bf45c5cd5	ubuntu-11.04-server-amd64 iso
b1a479c6593a90029414d201cb63a9cc	ubuntu-11.04-server-i386 iso

Ver en YouTube

Píldora 44: ¿Cómo funciona el hash MD5?

<https://www.youtube.com/watch?v=dxqxxwi9pww>

# Conclusiones de la Lección 7.2

- MD5 usa la construcción de Merkle-Damgård de compresión
- Por lo tanto, divide el mensaje en bloques de 512 bits, incluyendo siempre un relleno de ceros que comienza por 0x 80 y dejando los últimos 64 bits para indicar el tamaño de texto
- Trabaja con 4 vectores iniciales ABCD de 32 bits cada uno que se mezclan con el bloque de 512 bits del mensaje M en 64 vueltas o rondas con las funciones F, G, H e I, con 16 vueltas en cada una
- En cada vuelta, se tomarán 16 palabras diferentes  $M_j$  de 32 bits del mensaje M, desde  $M_0$  hasta  $M_{15}$ , además de 16 constantes  $t_j$  diferentes y unos desplazamientos  $s_j$  determinados en una tablas
- MD5 entrega un resumen de 128 bits y hoy no es recomendable su uso

# Lectura recomendada (1/2)

- Guion píldora formativa Thoth nº 44, ¿Cómo funciona el hash MD5?, Jorge Ramió, 2017
  - <https://www.criptored.es/thoth/material/texto/pildora044.pdf>
- CLCript 02: MD5 y SHA-1: relleno y endianness, Jorge Ramió, 2019
  - [https://www.criptored.es/descarga/CLCript\\_entrega\\_02\\_MD5\\_y\\_SHA1\\_relleno\\_y\\_endianness.pdf](https://www.criptored.es/descarga/CLCript_entrega_02_MD5_y_SHA1_relleno_y_endianness.pdf)
- Merkle–Damgård construction, Wikipedia
  - [https://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd\\_construction](https://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction)
- The MD5 Message-Digest Algorithm, Request for Comments 1321, R Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc. April 1992
  - <https://tools.ietf.org/html/rfc1321>



# Lectura recomendada (2/2)

- Padding (cryptography), Wikipedia
  - [https://en.wikipedia.org/wiki/Padding \(cryptography\)](https://en.wikipedia.org/wiki/Padding_(cryptography))
- Understanding the length extension attack, Cryptography
  - <https://crypto.stackexchange.com/questions/3978/understanding-the-length-extension-attack>
- Everything you need to know about hash length extension attacks, SkullSecurity
  - <https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks>



# Class4crypt c4c7.3

## Módulo 7. Funciones hash

### Lección 7.3. Función hash SHA-1

- 7.3.1. Información compartida con la clase c4c7.2 sobre MD5: estructura Merkle-Damgård , relleno ISO/IEC 7816-4, bits de indicación del tamaño
- 7.3.2. Funciones hash SHA y el proyecto Capstone con DSA
- 7.3.3. Por qué se habla de SHA-0
- 7.3.4. Características de la función hash SHA-1
- 7.3.5. Funcionamiento del hash SHA-1
- 7.3.6. SHA-1 por dentro con el software CriptoRes

Class4crypt c4c7.3 Función hash SHA-1  
<https://www.youtube.com/watch?v=w-MIDN0tm5A>

# Información compartida con Lección c4c7.2

- En SHA-1, al igual que en MD5:
  - Se usa la construcción de Merkle-Damgård de compresión
  - Se usa relleno ISO/IEC 7816-4, representado en hexadecimal como 0x 80 00 00..., es decir, en binario un 1 seguido de 0s
  - Una vez puesto al menos el primer byte de relleno 0x 80, se reservan los últimos 64 bits del documento, para indicar el tamaño del mismo y evitar ataques por extensión de longitud
- Si necesita repasar estos conceptos, por favor vea la clase c4c7.2 Función hash MD5, estructura y operaciones, en esta url:
  - <https://www.youtube.com/watch?v=rkBVDhfCy1c>

# Secure Hash Algorithm SHA y SHA-0



- En 1993, la National Security Agency NSA de los Estados Unidos diseña una función hash de 160 bits, con la idea de que permita sustituir a la función hash MD5 de Ron Rivest (1991) de solo 128 bits
- Además, en ese mismo año 1993 el gobierno USA da inicio al proyecto Capstone para nuevos estándares de criptografía de uso público y gubernamental, impulsado por la propia NSA y el National Institute of Standards and Technology NIST
- En la firma digital Digital Signature Algorithm DSA que se incluye en el proyecto Capstone, es obligatorio usar un hash de 160 bits
- En 1995 la NSA actualiza SHA como SHA-1, justificando que en aquel algoritmo había encontrado debilidades, y pide que pase a llamarse SHA-0 y no se use, **sin publicar las razones técnicas que lo han forzado**

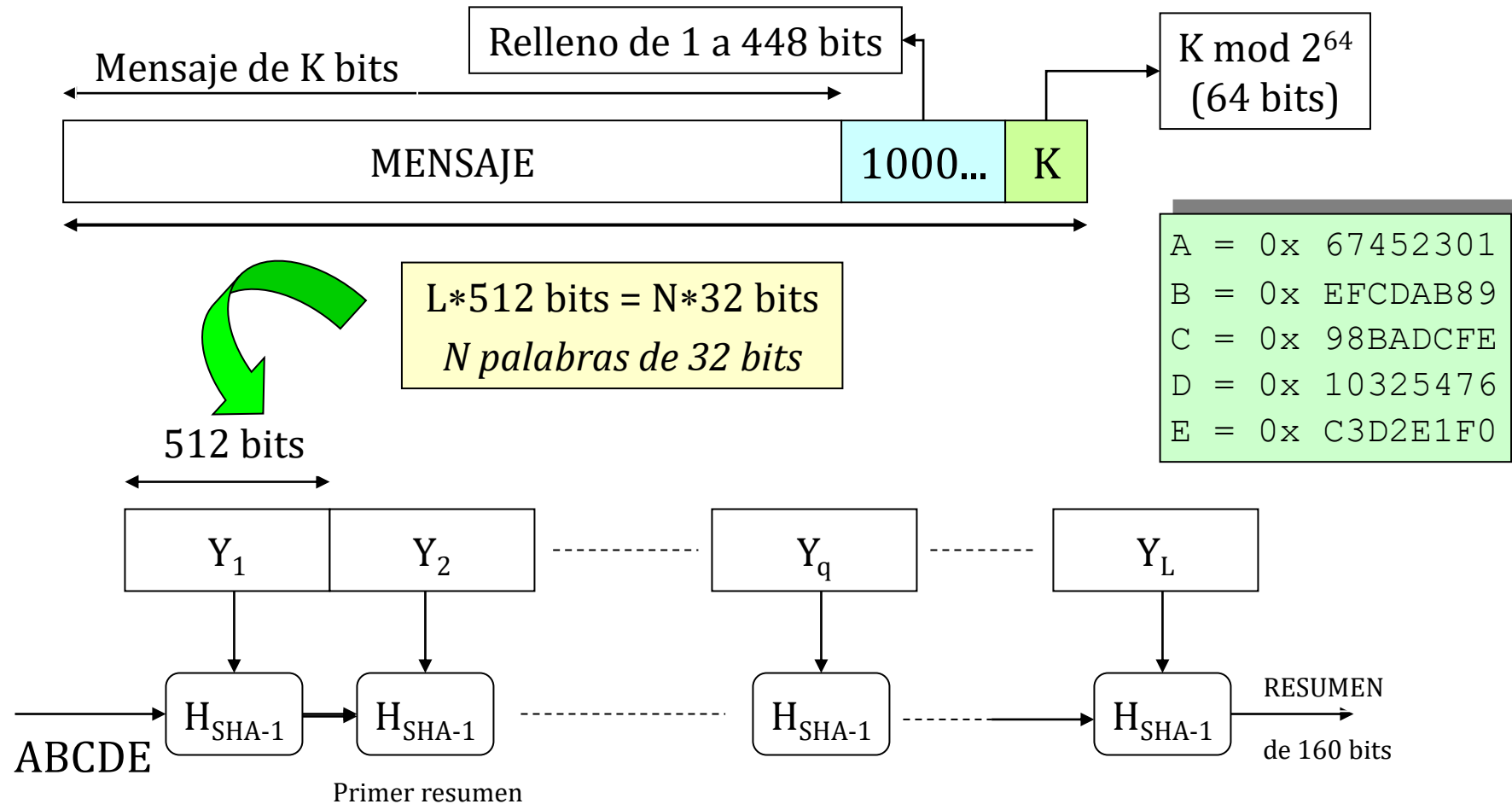
# Características de SHA-1 (1/2)

- SHA-1 fue creado por la NSA en 1995 y soluciona las debilidades que mostraba su predecesor SHA o SHA-0 (1993)
- Esta función hash comienza su declive cuando en 2005 MD5 es atacado, debido a que sus operaciones son muy similares
- Procesa bloques de 512 bits con una salida de 160 bits
- Expande el mensaje hasta una longitud 64 bits inferior a un múltiplo de 512 bits
- El relleno se indica con un bit 1 seguido de tantos bits 0 como sean necesarios, reservando los últimos 64 bits del mensaje para añadir información sobre la longitud en bits del mismo

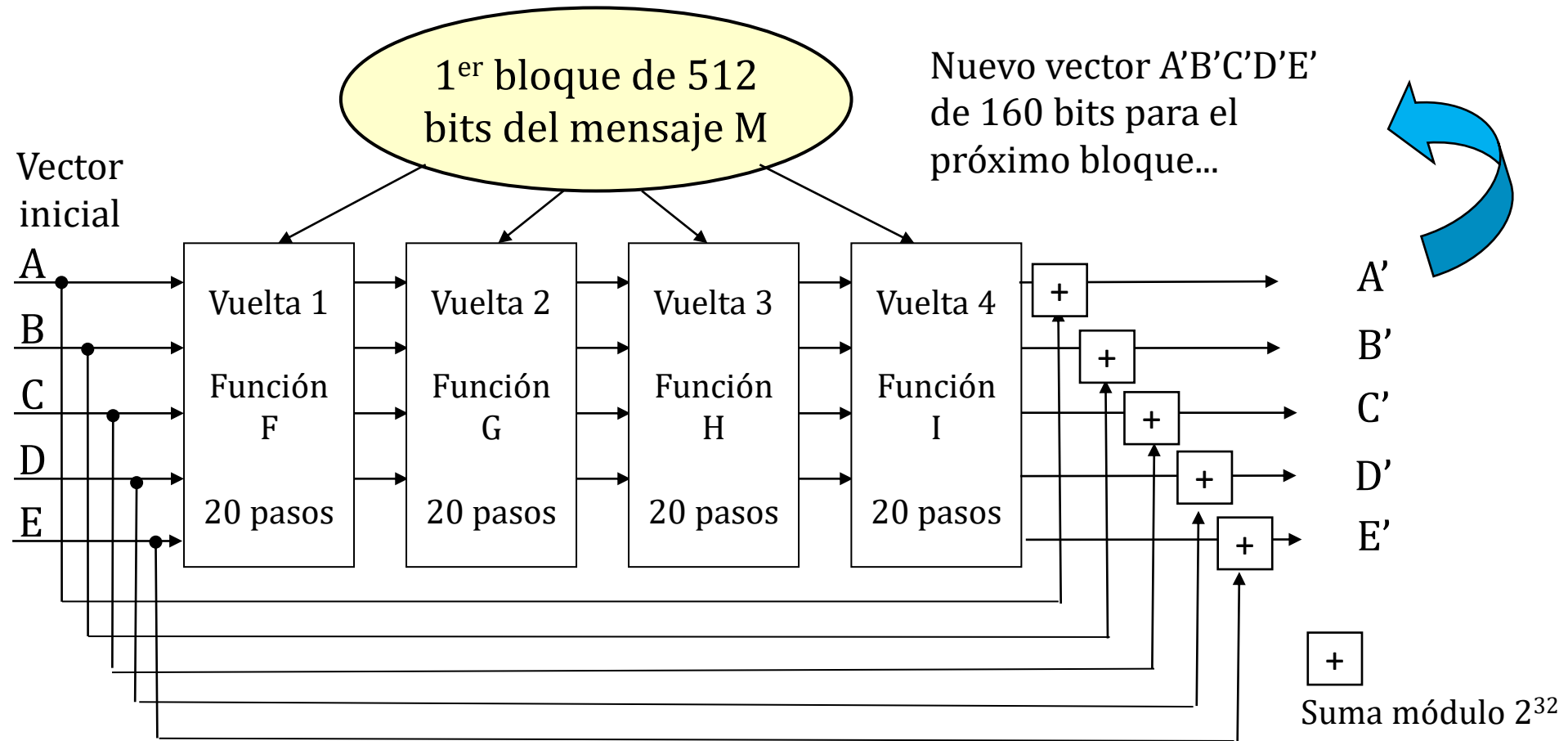
# Características de SHA-1 (2/2)

- El algoritmo comienza con 5 vectores iniciales (IV) ABCDE de 32 bits cada uno, cuyo valor inicial no es secreto. A estos vectores y al primer bloque de 512 bits de M se le aplican 80 operaciones de 32 bits con puertas lógicas, cuyo carácter es no lineal
- Las 80 operaciones se engloban en 4 vueltas o rondas F, G, H, I
- Como resultado de estas operaciones, se obtienen cinco nuevos vectores A'B'C'D'E' que serán la entrada IV' para el segundo bloque de 512 bits, repitiéndose este proceso con los restantes bloques de M hasta terminar el mensaje
- La última salida de IV corresponde al resumen final  $H = h(M)$
- RFC 6234: <https://tools.ietf.org/html/rfc6234#page-36>

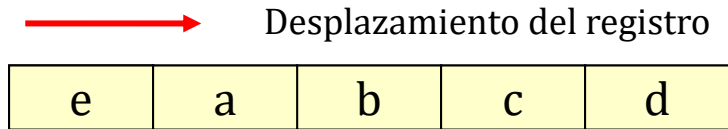
# Esquema de SHA-1



# Bloque principal de SHA-1



# Funciones en vueltas F, G, H e I en SHA-1



- Se realiza el proceso con la función F para las palabras  $M_0$  a  $M_{15}$  de 32 bits del primer bloque. Para obtener las palabras  $M_{16}$  a  $M_{19}$  que faltan, se usará un algoritmo de expansión que veremos más adelante
- En las vueltas 2, 3 y 4 se repite el proceso con funciones G, H e I desde  $M_{20}$  hasta  $M_{79}$
- Tenemos 4 vueltas multiplicado por 20 palabras = 80 pasos por cada bloque de 512 bits... pero, ¿cómo es posible sacar 80 palabras diferentes de 32 bits de un bloque de 512 bits y un mensaje M que cuenta solo con 16 palabras?

- F (b, c, d) → pasos t = 0 a 19  
(b AND c) OR ((NOT b) AND d)
- G (b, c, d) → pasos t = 20 a 39  
b XOR c XOR d
- H (b, c, d) → pasos t = 40 a 59  
(b AND c) OR (b AND d) OR (c AND d)
- I (b, c, d) → pasos t = 60 a 79  
b XOR c XOR d

Funciones F, G, H e I en SHA-1



# Las 80 palabras de SHA-1 en cada vuelta

- Cada bloque de 16 palabras del mensaje ( $M_0 \dots M_{15}$ ) se expandirá a 80 palabras ( $W_0 \dots W_{79}$ ) según el siguiente algoritmo que, cuando se acaban las 16 palabras de 32 bits del mensaje  $M$ , genera las otras 64 palabras restantes haciendo una operación lógica xor con palabras anteriores
- Para  $t = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$ 
  - $W_t = M_t$
- Para  $t = 16, 17, 18, 19, 20, 21, \dots 74, 75, 76, 77, 78, 79$ 
  - $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$

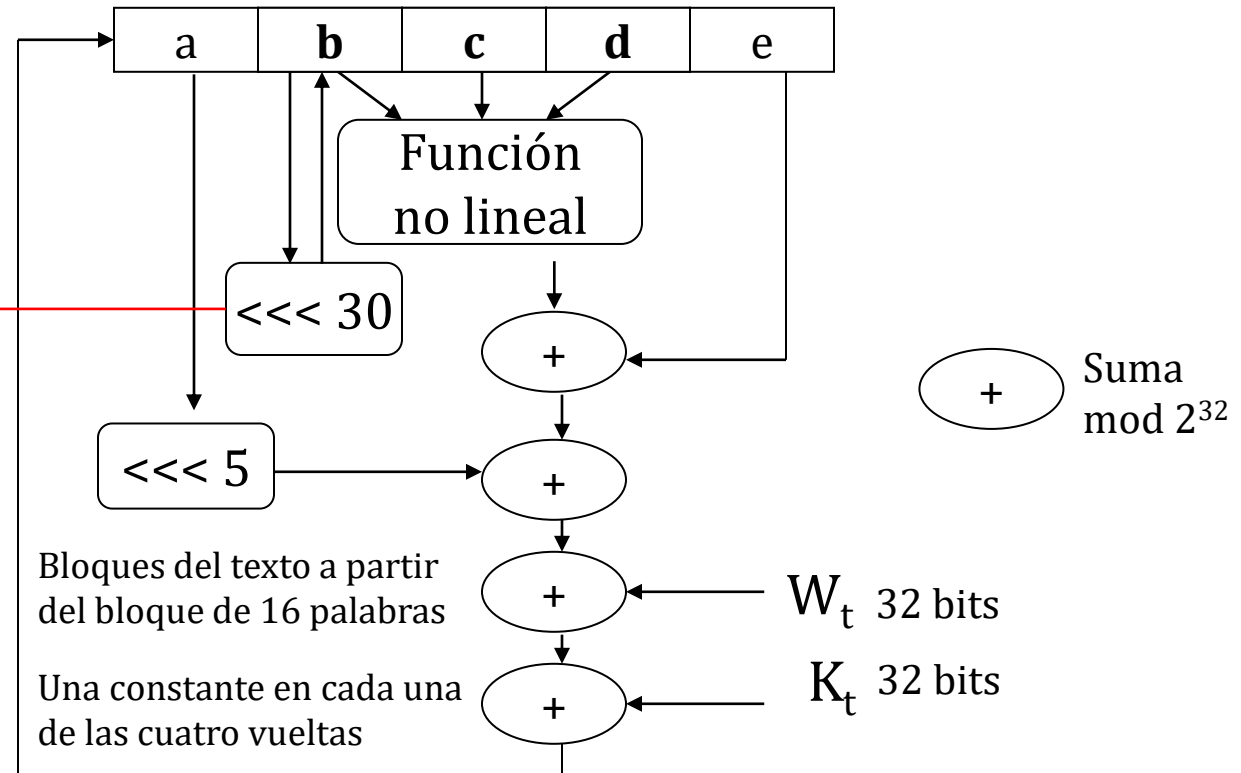
# Funcionamiento de SHA-1

Vector de 160 bits

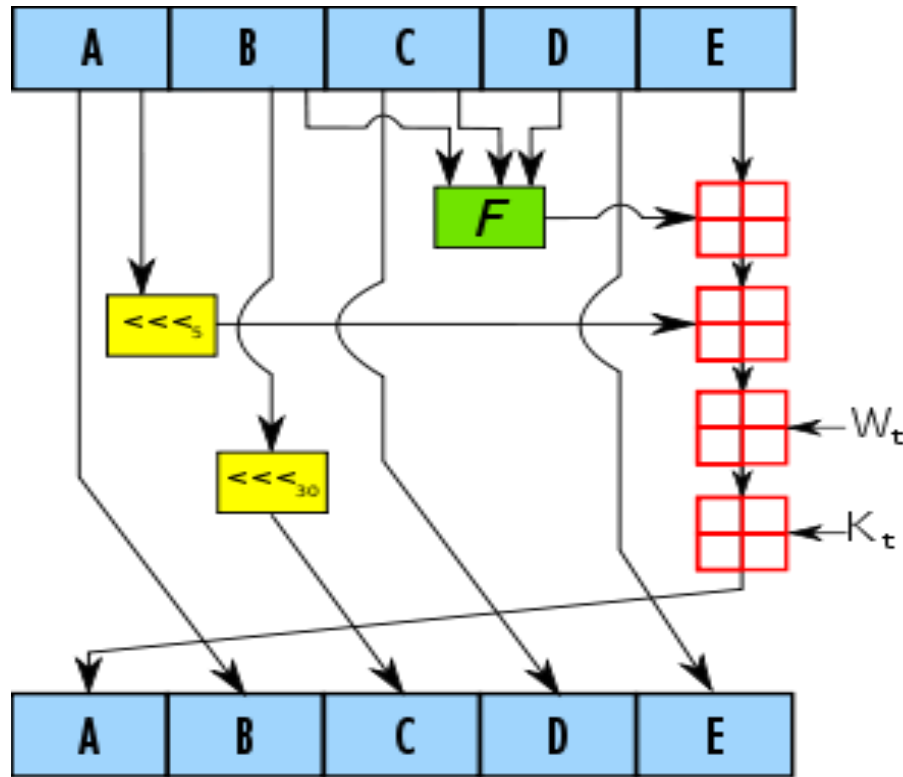
A = 0x 67452301  
B = 0x EFCDA89  
C = 0x 98BADCFE  
D = 0x 10325476  
E = 0x C3D2E1F0

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

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



# Funciones en cada vuelta de SHA-1



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

Dependiendo de la ronda, la función  $F$  puede ser F, G, H, I:

$F = (\mathbf{B} \text{ AND } \mathbf{C}) \text{ OR } (\text{NOT } \mathbf{B} \text{ AND } \mathbf{D})$       1ª ronda

$G = (\mathbf{B} \text{ XOR } \mathbf{C} \text{ XOR } \mathbf{D})$       2ª ronda

$H = (\mathbf{B} \text{ AND } \mathbf{C}) \text{ OR } (\mathbf{B} \text{ AND } \mathbf{D}) \text{ OR } (\mathbf{C} \text{ AND } \mathbf{D})$       3ª Ronda

$I = (\mathbf{B} \text{ XOR } \mathbf{C} \text{ XOR } \mathbf{D})$       4ª ronda

 es una suma mod  $2^{32}$

- $K_t = 5A827999$  para  $t = 0, \dots, 19$
- $K_t = 6ED9EBA1$  para  $t = 20, \dots, 39$
- $K_t = 8F1BBCDC$  para  $t = 40, \dots, 59$
- $K_t = CA62C1D6$  para  $t = 60, \dots, 79$

# Algoritmo de desplazamiento en cada paso

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

- Algoritmo de desplazamiento del vector abcde en cada paso

Para  $t = 0$  hasta 79 hacer

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

$$a = e$$

$$e = d$$

$$d = c$$

$$c = b \lll 30$$

$$b = a$$

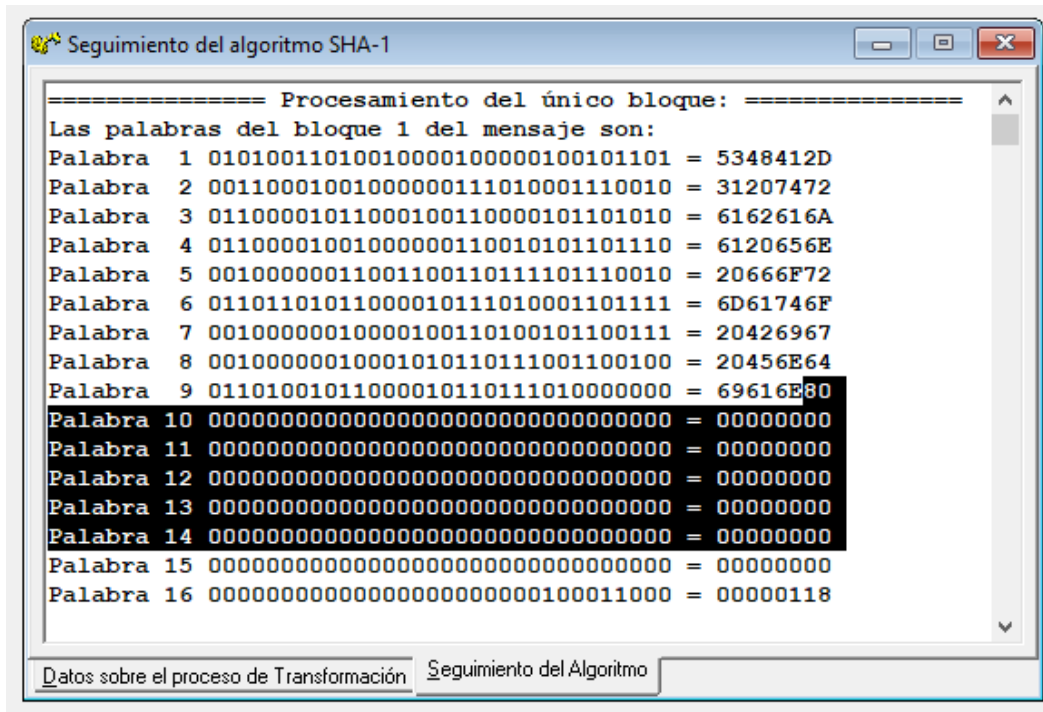
$$a = \text{TEMP}$$

# SHA-1 por dentro con CriptoRes

- Mediante la pestaña seguimiento de CriptoRes, podrá observar los bytes de relleno y los 64 bits (dos últimas palabras de 32 bits) reservadas para la longitud del mensaje al final de éste
  - Comprobar el relleno y el tamaño del mensaje M de un solo bloque
  - $M = \text{SHA-1 trabaja en formato Big Endian}$
  - Usando operaciones en hexadecimal, calcular cómo queda el vector inicial abcde después del primer desplazamiento o primer paso
- Siempre existirá un relleno. Por ejemplo, si no hay entrada, el hash de nulo se calcula sobre el mensaje 0x 80 00 00 00 00 ... 00 00 00
  - $h(\text{vacío})_{\text{SHA-1}} = \text{DA39A3EE5E6B4B0D3255BFEF95601890AFD80709}$
- Encontrará otros ejemplos con CriptoRes en la lección de MD5 c4c7.2

# Relleno y tamaño del mensaje

- M = SHA-1 trabaja en formato Big Endian
- $h(M)_{\text{SHA-1}} = \text{D66680113B6E804C75B1E9A78F60E5456D6BB965}$



```
===== Procesamiento del único bloque: =====
Las palabras del bloque 1 del mensaje son:
Palabra 1 01010011010010000100000100101101 = 5348412D
Palabra 2 00110001001000000111010001110010 = 31207472
Palabra 3 01100001011000100110000101101010 = 6162616A
Palabra 4 01100001001000000110010101101110 = 6120656E
Palabra 5 00100000011001100110111101110010 = 20666F72
Palabra 6 01101101011000010111010001101111 = 6D61746F
Palabra 7 00100000010000100110100101100111 = 20426967
Palabra 8 00100000010001010110111001100100 = 20456E64
Palabra 9 01101001011000010110110100000000 = 69616E80
Palabra 10 00000000000000000000000000000000 = 00000000
Palabra 11 00000000000000000000000000000000 = 00000000
Palabra 12 00000000000000000000000000000000 = 00000000
Palabra 13 00000000000000000000000000000000 = 00000000
Palabra 14 00000000000000000000000000000000 = 00000000
Palabra 15 00000000000000000000000000000000 = 00000000
Palabra 16 0000000000000000000000000100011000 = 00000118
```

- Lectura Big Endian
- SHA- = 0x 5348412D
- 1 tr = 0x 31207472
- El relleno comienza en el último byte de la palabra 9 y termina en la palabra 14, en total 168 bits
- Tamaño del mensaje M = 0x 118, 280 bits, es decir, 35 bytes

# Cálculos en el primer paso (datos)

- Usando solo la calculadora de Windows en hexadecimal con Dword, palabras de 32 bits, se pide encontrar cómo queda el vector IV después del primer paso o primera operación
- Datos para cálculo de  $TEMP = F_t(b,c,d) + e + (a \lll 5) + W_t + K_t$
- $a = 67452301 = 01100111010001010010001100000001$
- $a \lll 5 = 11101000101001000110000000101100 = 0x E8A4602C$
- $b = EFCDAB89 = 1110111111001101101010101110001001$
- $b \lll 30 = 01111011111100110110101011100010 = 7BF36AE2$
- $c = 98BADCFE$
- $d = 10325476$
- $e = C3D2E1F0$
- $M_0 = 5348412D$  (primeros 4 bytes del mensaje M)
- $K_t = 5A827999$

Vector abcde en  $t = 0$

67452301	EFCDAB89	98BADCFE	10325476	C3D2E1F0
----------	----------	----------	----------	----------

# Cálculos en el primer paso (operaciones)


- Calculando  $F_t(b,c,d) + e + (a \lll 5) + W_t + K_t$
- $F_t(b,c,d) = (b \text{ AND } c) \text{ or } (\text{NOT } b \text{ AND } d)$
- $b \text{ AND } c = \text{EFCDAB89 AND } 98\text{BADCFE} = 88888888$
- $\text{NOT } b = \text{NOT EFCDAB89} = 10325476$
- $\text{NOT } b \text{ AND } d = 10325476 \text{ and } 10325476 = 10325476$
- $(b \text{ AND } c) \text{ or } (\text{NOT } b \text{ AND } d) = 88888888 \text{ or } 10325476 = 98\text{BADCFE}$
- $+ e \quad 98\text{BADCFE} + \text{C3D2E1F0} = 5\text{C8DBEEE}$
- $+ (a \lll 5) \quad 5\text{C8DBEEE} + \text{E8A4602C} = 45321\text{F1A}$
- $+ M_0 \quad 45321\text{F1A} + 5348412\text{D} = 987\text{A6047}$
- $+ K_t \quad 987\text{A6047} + 5\text{A827999} = \text{F2FCD9E0}$ . Por lo tanto,  $\text{TEMP} = \text{F2FCD9E0}$

$e = d = 10325476$   
 $d = c = 98\text{BADCFE}$   
 $c = b \lll 30 = 7\text{BF36AE2}$   
 $b = a = 67452301$   
 $a = \text{TEMP} = \text{F2FCD9E0}$



67452301	EFCDAB89	98BADCFE	10325476	C3D2E1F0
----------	----------	----------	----------	----------

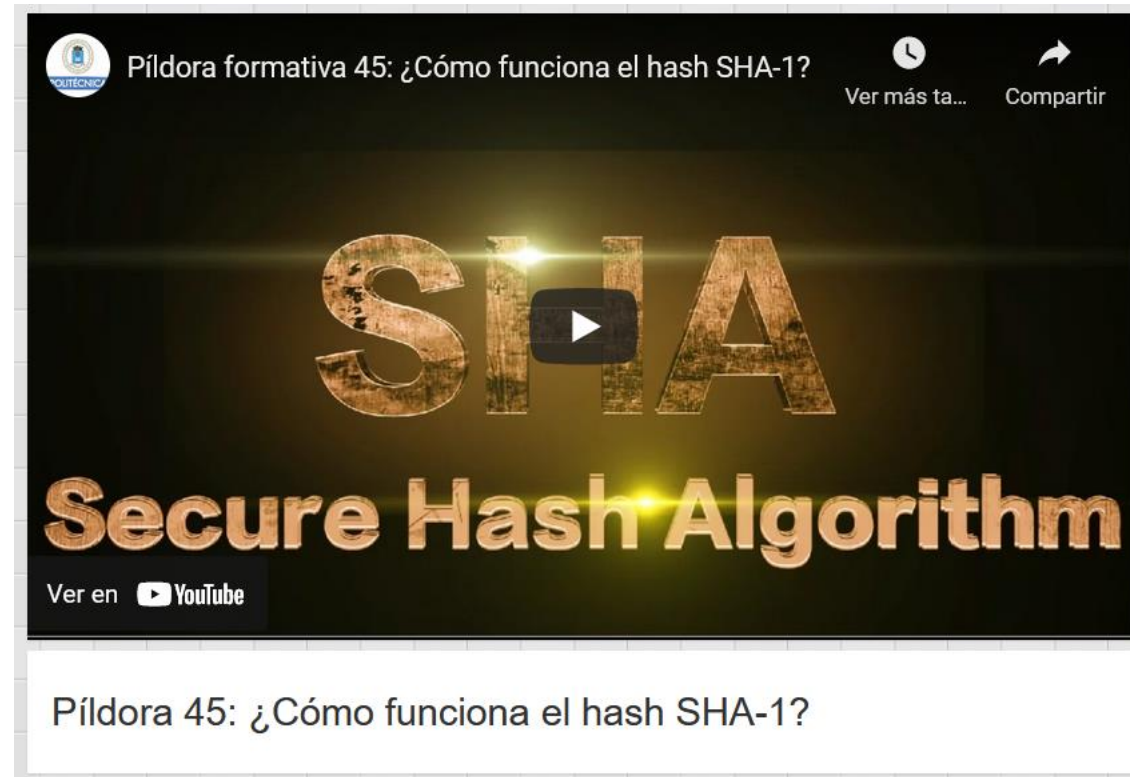
Vector abcde en  $t = 1$

Vector abcde en  $t = 0$  

F2FCD9E0	67452301	7BF36AE2	98BADCFE	10325476
----------	----------	----------	----------	----------



# Más información en píldoras Thoth



<https://www.youtube.com/watch?v=pG8785ZEFuM>

# Conclusiones de la Lección 7.3

- SHA-1 usa la construcción de Merkle-Damgård de compresión
- Por lo tanto, divide el mensaje en bloques de 512 bits, incluyendo siempre un relleno de ceros que comienza por 0x 80 y dejando los últimos 64 bits para indicar el tamaño del documento
- Trabaja con un vector inicial de 5 palabras ABCDE de 32 bits cada una, que se mezclan con el bloque de 512 bits del mensaje M en 80 pasos con 4 vueltas con funciones F, G, H e I, 20 pasos vueltas en cada una
- Para cada vuelta se tomarán 20 palabras diferentes de 32 bits, a partir de M y permitiendo mediante un algoritmo obtener hasta 80 palabras, usándose además 4 constantes  $K_t$  una diferente para cada vuelta
- SHA-1 entrega un resumen de 160 bits y su uso cada vez es menor

# Lectura recomendada (1/2)

- Guion píldora formativa Thoth nº 45, ¿Cómo funciona el hash SHA-1?, Jorge Ramió, 2017
  - <https://www.criptored.es/thoth/material/texto/pildora045.pdf>
- CLCript 02: MD5 y SHA-1: relleno y endianness, Jorge Ramió, 2019
  - [https://www.criptored.es/descarga/CLCript\\_entrega\\_02\\_MD5\\_y\\_SHA1\\_relleno\\_y\\_endianness.pdf](https://www.criptored.es/descarga/CLCript_entrega_02_MD5_y_SHA1_relleno_y_endianness.pdf)
- Capstone (cryptography), Wikipedia
  - [https://en.wikipedia.org/wiki/Capstone\\_\(cryptography\)](https://en.wikipedia.org/wiki/Capstone_(cryptography))
- US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF), RFC IETF
  - <https://tools.ietf.org/html/rfc6234#page-36>

# Lectura recomendada (2/2)

- SHA-1, Wikipedia
  - <https://en.wikipedia.org/wiki/SHA-1#SHA-0>
- SHA1 Online, Online Tools
  - <https://emn178.github.io/online-tools/sha1.html>
- ASCII, Hex, Binary, Decimal, Base64 converter, RapidTables
  - <https://www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html>
- Re-Hashed: The Difference Between SHA-1, SHA-2 and SHA-256 Hash Algorithms, Patrick Nohe
  - <https://www.thesslstore.com/blog/difference-sha-1-sha-2-sha-256-hash-algorithms/>
- Why did the NSA create SHA?, Cryptography
  - <https://crypto.stackexchange.com/questions/79522/why-did-the-nsa-create-sha>

# Class4crypt c4c7.4

## Módulo 7. Funciones hash

### Lección 7.4. Colisiones en funciones hash MD5 y SHA-1

7.4.1. El efecto de difusión o avalancha, que no siempre se cumple

7.4.2. Repasando las debilidades de las funciones hash

7.4.3. Colisiones en MD5

7.4.3.1. Certificados digitales X.509

7.4.3.2. Archivos ejecutables

7.4.3.3. Archivos de documentos PDF e imágenes JPG

7.4.4. Colisiones en SHA-1

Class4crypt c4c7.4 Colisiones en funciones hash MD5 y SHA-1  
<https://www.youtube.com/watch?v=T7JZVqdVFBU>

# El efecto difusión o avalancha



- Un efecto que demuestra fortaleza: el cambio de un solo bit del mensaje  $M$  deberá afectar en media al 50% de los bits del hash
- Y en la teoría esto sí se cumple. Lo veremos a continuación con un simple ejercicio práctico
- Pero en MD5, y últimamente en SHA-1, ya no solo se cambia un bit sino varios bytes del mensaje  $M$ , y los hashes de  $M$  y  $M'$  son iguales, siendo esto no por algo fortuito sino programado
- Por lo tanto, a partir de dos archivos  $M$  y  $M'$  diferentes existen técnicas que permiten modificarlos de tal forma que se obtenga que  $h(M) = h(M')$ , es decir una colisión forzada

# Práctica del efecto de difusión o avalancha

- Encontrar los hashes MD5 de estos dos textos donde solo hay un bit de diferencia:
- $M_1 = \text{El café cuesta 2 euros}$      $M_2 = \text{El café cuesta 6 euros}$     ( $2 = 0011\ 0010$  y  $6 = 0011\ 0110$ )
- $h_{MD5}(M_1) = D89D86EE07E5D3DAAF9D59109A470BC2 =$
- $1101100010011101100001101110111000000111111001011101001111011010101011$   
 $1110011101010110010001000010011010010001110000101111000010$
- $h_{MD5}(M_2) = 80A1F7DCAB1A111B4CAC3E4A2410C3AF =$
- $1000000010100001111101111101110010101011000110100001000100011011010011$   
 $0010101100001111100100101000100100000100001100001110101111$
- $h_{MD5}(M_1) \text{ XOR } h_{MD5}(M_2) =$
- $010110000011110001110001001100101010110011111111100001011000001111000$   
 $1100110001011001110101101010111110010101111100100001101101$
- Los bits en rojo (1) muestran los bits que han cambiado entre  $h_{MD5}(M_1)$  y  $h_{MD5}(M_2)$
- De los 128 bits del hash MD5, los bits que han cambiado entre  $h_{MD5}(M_1)$  y  $h_{MD5}(M_2)$  son 68, prácticamente el 50% de los bits

# Las colisiones en las funciones hash



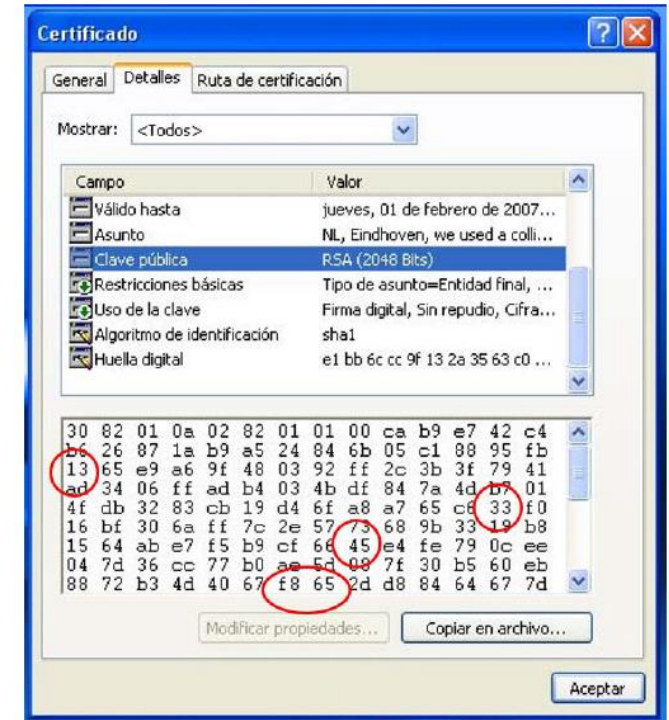
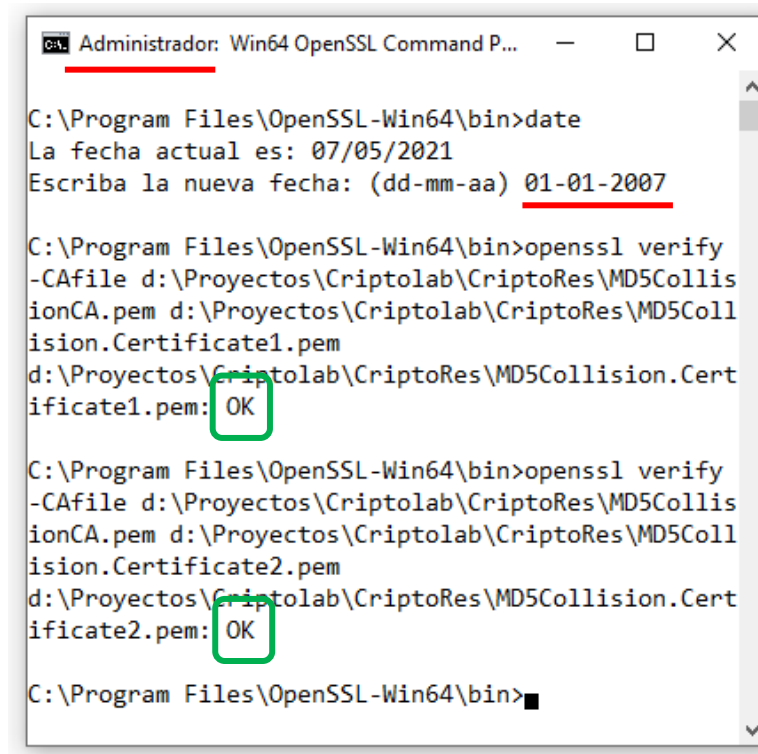
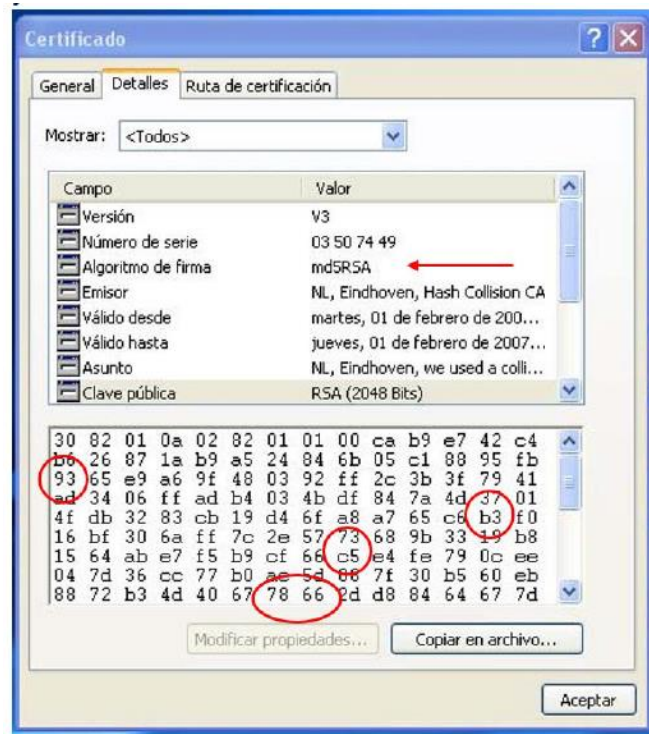
- En la lección c4c7.1 vimos que los hashes deber tener fortaleza ante ataques por primera preimagen y segunda preimagen, necesitando como media  $2^{n-1}$  intentos, o bien por la paradoja del cumpleaños, siendo ahora necesarios en media solo  $2^{n/2}$  intentos
- El problema no es que existan esas probabilidades de colisión, que siempre las habrá por el principio del palomar, sino el hecho de que estas colisiones en vez de ser casuales, sean fruto de una modificación a conciencia realizada sobre los archivos
- Y esto último es lo que le viene sucediendo desde el año 2004 a MD5 y desde 2017 a SHA-1. MD5 es mucho más vulnerable que SHA-1, y no solo porque su resumen tenga 32 bits menos



# Primer aviso de colisiones en MD5 en 2004

- 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 ante colisiones, ambas con construcción Merkle-Damgård
- En aquellos primeros ataques se demuestra que puede forzarse el cambio de ciertos bits en el mensaje para que dos mensajes, ahora diferentes por ese cambio, tengan el mismo hash MD5, si bien dichos mensajes en hexadecimal no tienen sentido
- Con el paso del tiempo, esas modificaciones pueden dar lugar a que documentos completamente opuestos, de cualquier formato, tengan el mismo hash MD5, usando la paradoja del cumpleaños

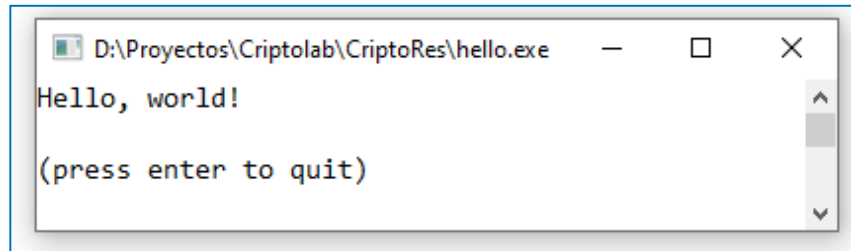
# Colisiones con MD5 en certificados X.509



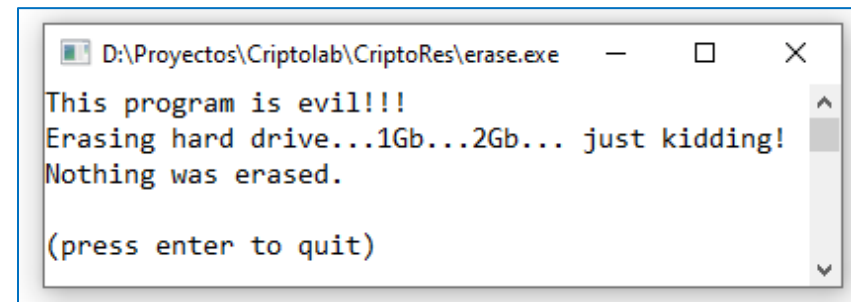
# Ejercicio propuesto de colisiones en X.509

- Descargue desde el sitio web estos archivos: MD5Collision.certificate1.cer, MD5Collision.certificate2.cer y MD5CollisionCA.cer
- Comandos a ejecutar en C:\OpenSSL-Win64\bin>
- Convertir archivos .cer en archivos .pem
- openssl x509 -in MD5Collision.certificate1.cer -inform DER -out MD5Collision.certificate1.pem
- openssl x509 -in MD5Collision.certificate2.cer -inform DER -out MD5Collision.certificate2.pem
- openssl x509 -in MD5CollisionCA.cer -inform DER -out MD5CollisionCA.pem
- Verificación de colisión certificate1 y certificate2
- Recuerde que estos certificados tienen validez hasta el 1 de febrero de 2007 (cambie la fecha del PC)
- openssl verify -CAfile MD5CollisionCA.pem MD5Collision.certificate1.pem
- openssl verify -CAfile MD5CollisionCA.pem MD5Collision.certificate2.pem

# Colisiones con MD5 en archivos ejecutables

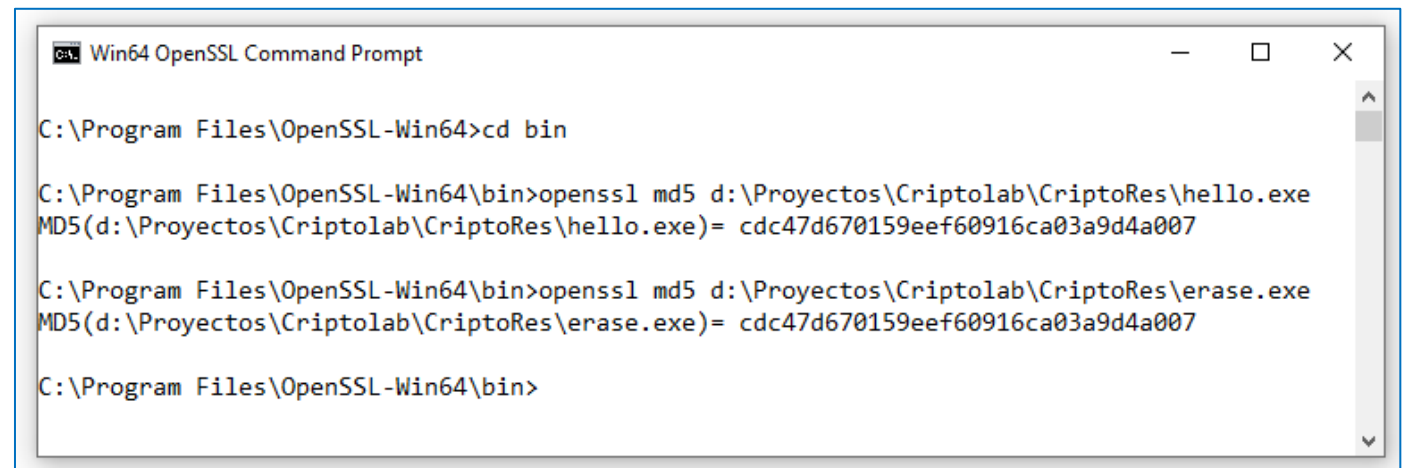


```
D:\Proyectos\Criptolab\CriptoRes\hello.exe
Hello, world!
(press enter to quit)
```



```
D:\Proyectos\Criptolab\CriptoRes\erase.exe
This program is evil!!!
Erasing hard drive...1Gb...2Gb... just kidding!
Nothing was erased.
(press enter to quit)
```

Dos archivos, uno “bueno” y uno “malo”



```
Win64 OpenSSL Command Prompt
C:\Program Files\OpenSSL-Win64>cd bin
C:\Program Files\OpenSSL-Win64\bin>openssl md5 d:\Proyectos\Criptolab\CriptoRes\hello.exe
MD5(d:\Proyectos\Criptolab\CriptoRes\hello.exe)= cdc47d670159eef60916ca03a9d4a007
C:\Program Files\OpenSSL-Win64\bin>openssl md5 d:\Proyectos\Criptolab\CriptoRes\erase.exe
MD5(d:\Proyectos\Criptolab\CriptoRes\erase.exe)= cdc47d670159eef60916ca03a9d4a007
C:\Program Files\OpenSSL-Win64\bin>
```

$\text{Hash}_{\text{MD5}} = \text{CDC47D670159EEF60916CA03A9D4A007}$

- Estos dos archivos puedes descargarlos desde la página web “MD5 Collision Demo” de Peter Selinger (2006)
  - <https://www.mathstat.dal.ca/~selinger/md5collision/>

# Práctica colisiones en archivos ejecutables

- C:\Program Files\OpenSSL-Win64\bin>
- C:\Program Files\OpenSSL-Win64\bin>openssl md5 d:\Proyectos\Criptolab\CriptoRes\hello.exe  
MD5(d:\Proyectos\Criptolab\CriptoRes\hello.exe)= cdc47d670159eef60916ca03a9d4a007
- C:\Program Files\OpenSSL-Win64\bin>openssl md5 d:\Proyectos\Criptolab\CriptoRes\erase.exe  
MD5(d:\Proyectos\Criptolab\CriptoRes\erase.exe)= cdc47d670159eef60916ca03a9d4a007

# Colisiones con MD5 en documentos PDF

Desarrollado en 2007 por Marc Stevens, CWI, Amsterdam, The Netherlands; Arjen K. Lenstra, EPFL, Lausanne, Switzerland, and Bell Labs, USA; Benne de Weger, TU/e, Eindhoven, The Netherlands

A:  <a href="#">John Edwards.pdf</a>	G:  <a href="#">Fred Thompson.pdf</a>
B:  <a href="#">John McCain.pdf</a>	H:  (hidden)
C:  <a href="#">Mitt Romney.pdf</a>	I:  <a href="#">Paris Hilton.pdf</a>
D:  <a href="#">Ralph Nader.pdf</a>	J:  <a href="#">Al Gore.pdf</a>
E:  (hidden)	K:  <a href="#">Jeb Bush.pdf</a>
F:  <a href="#">Barack Obama.pdf</a>	L:  <a href="#">Oprah Winfrey.pdf</a>

All twelve documents we prepared, the ten given above and two hidden ones, have the MD5 hash value

3D515DEAD7AA16560ABA3E9DF05CBC80.

We predict that the winner of the 2008 election for  
President of the United States  
will be:

**Barack Obama**

We predict that the winner of the 2008 election for  
President of the United States  
will be:

**Paris Hilton**



# Práctica colisiones en archivos PDF

- C:\Program Files\OpenSSL-Win64\bin>
- C:\Program Files\OpenSSL-Win64\bin>openssl md5 d:\Proyectos\Criptolab\CriptoRes\barack.pdf  
MD5(d:\Proyectos\Criptolab\CriptoRes\barack.pdf)= 3d515dead7aa16560aba3e9df05cbc80
- C:\Program Files\OpenSSL-Win64\bin>openssl md5 d:\Proyectos\Criptolab\CriptoRes\paris.pdf  
MD5(d:\Proyectos\Criptolab\CriptoRes\paris.pdf)= 3d515dead7aa16560aba3e9df05cbc80

# Colisiones con MD5 en imágenes JPG (1/2)

Desarrollado en 2012 por Nathaniel McHugh Sheffield, U.K., investigador en seguridad, tarda 10 horas en encontrar una colisión por paradoja del cumpleaños



JamesBrown.jpg  
(102.400 bytes)

BarryWhite.jpg  
(102.400 bytes)

Hash<sub>MD5</sub> = e06723d4961a0a3f950e7786f3766338

Descarga: <https://arstechnica.com/information-technology/2014/11/crypto-attack-that-hijacked-windows-update-goes-mainstream-in-amazon-cloud/>





# Colisiones con MD5 en imágenes JPG (2/2)



ship.jpg  
335.104 bytes



plane.jpg  
335.104 bytes

$\text{Hash}_{\text{MD5}} = 253\text{dd}04\text{e}87492\text{e}4\text{fc}3471\text{de}5\text{e}776\text{bc}3\text{d}$

Descarga: <https://natmchugh.blogspot.com/2015/02/create-your-own-md5-collisions.html>

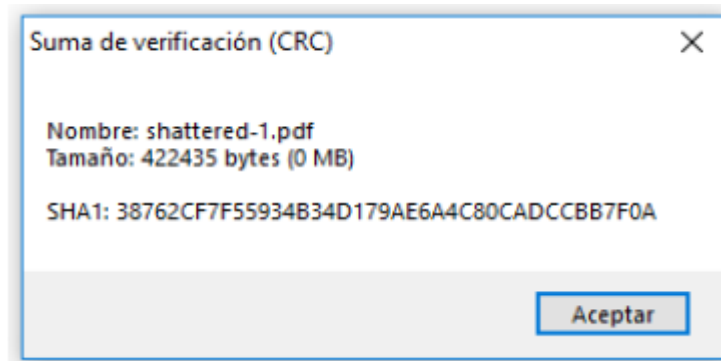
# Práctica colisiones en archivos JPG

- C:\Program Files\OpenSSL-Win64\bin>
- C:\Program Files\OpenSSL-Win64\bin>`openssl md5 d:\Proyectos\Criptolab\CriptoRes\BarryWhite.jpg`  
MD5(d:\Proyectos\Criptolab\CriptoRes\BarryWhite.jpg)= e06723d4961a0a3f950e7786f3766338
- C:\Program Files\OpenSSL-Win64\bin>`openssl md5 d:\Proyectos\Criptolab\CriptoRes\JamesBrown.jpg`  
MD5(d:\Proyectos\Criptolab\CriptoRes\JamesBrown.jpg)= e06723d4961a0a3f950e7786f3766338
- C:\Program Files\OpenSSL-Win64\bin>`openssl md5 d:\Proyectos\Criptolab\CriptoRes\ship.jpg`  
MD5(d:\Proyectos\Criptolab\CriptoRes\ship.jpg)= 253dd04e87492e4fc3471de5e776bc3d
- C:\Program Files\OpenSSL-Win64\bin>`openssl md5 d:\Proyectos\Criptolab\CriptoRes\plane.jpg`  
MD5(d:\Proyectos\Criptolab\CriptoRes\plane.jpg)= 253dd04e87492e4fc3471de5e776bc3d

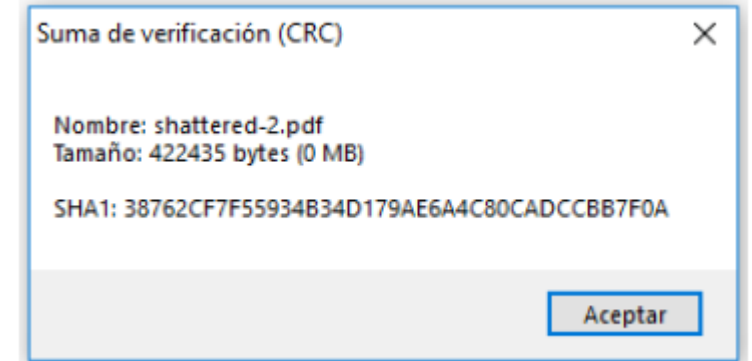
# Colisiones con SHA-1 en PDF: Shattered



PDF 1 | PDF 2



Marc Stevens, Pierre Karpman  
(CWI); Elie Bursztein, Ange  
Albertini, Yarik Markov (Google)  
27 de febrero de 2017  
Descarga: <https://shattered.io/>



# Práctica colisiones en archivos PDF SHA-1

- C:\Program Files\OpenSSL-Win64\bin>
- C:\Program Files\OpenSSL-Win64\bin>[openssl sha1 d:\Proyectos\Criptolab\CriptoRes\shattered-1.pdf](#)  
SHA1(d:\Proyectos\Criptolab\CriptoRes\shattered-1.pdf)= 38762cf7f55934b34d179ae6a4c80cadccbb7f0a
- C:\Program Files\OpenSSL-Win64\bin>[openssl sha1 d:\Proyectos\Criptolab\CriptoRes\shattered-2.pdf](#)  
SHA1(d:\Proyectos\Criptolab\CriptoRes\shattered-2.pdf)= 38762cf7f55934b34d179ae6a4c80cadccbb7f0a

# Conclusiones de la Lección 7.4

- Las funciones hash tienen como una fortaleza la propiedad de difusión o avalancha: el cambio de un solo bit en el mensaje debe afectar al 50% de bits
- Pero, en algunos casos, el cambio de varios bytes hace que dos mensajes diferentes colisionen, lo que unido a que las funciones hash son vulnerables a ataques por la paradoja del cumpleaños, facilita encontrar esas colisiones
- MD5 comienza a experimentar estas debilidades en 2004 y SHA-1 en 2017
- Pueden usarse diversas técnicas, por ejemplo añadir al final de cada archivo bits que permitan colisionar los hashes de ambos archivos modificados
- MD5 y SHA-1 aún se observan en muchos sitios, por ejemplo en el web de descarga de Apache (<https://httpd.apache.org/download.cgi>)
- Se recomienda hoy usar SHA-256 o superior, y si fuese posible SHA-3

# Lectura recomendada (1/2)

- CLCrypt 13: Colisiones en hashes MD5 y SHA-1, Jorge Ramió, 2019
  - [https://www.criptored.es/descarga/CLCrypt entrega 13 Colisiones en MD5 y SHA1.pdf](https://www.criptored.es/descarga/CLCrypt%20entrega%2013%20Colisiones%20en%20MD5%20y%20SHA1.pdf)
- How to Break MD5 and Other Hash Functions, Xiaoyun Wang and Hongbo Yu, 2004
  - <http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf>
- Colliding X.509 Certificates based on MD5-collisions, Lenstra, Wang y Weber, 2005
  - <http://www.win.tue.nl/~bdeweger/CollidingCertificates/>
- An Illustrated Guide to Cryptographic Hashes, Steve Friedl, 2005
  - <http://www.unixwiz.net/techtips/iguide-crypto-hashes.html>
- MD5 Collision Demo, Peter Selinger, Dep. of Mathematics and Statistics /Dalhousie University, 2006
  - <https://www.mathstat.dal.ca/~selinger/md5collision/>
- Predicting the winner of the 2008 US Presidential Elections using a Sony PlayStation 3, Stevens, Lenstra, de Weger, 2007
  - <https://www.win.tue.nl/hashclash/Nostradamus/>



# Lectura recomendada (2/2)

- Crypto attack that hijacked Windows Update goes mainstream in Amazon Cloud, Dan Goodin, 2014
  - <https://arstechnica.com/information-technology/2014/11/crypto-attack-that-hijacked-windows-update-goes-mainstream-in-amazon-cloud/>
- How I created two images with the same MD5 hash, Nat McHugh, 2014
  - <https://natmchugh.blogspot.com/2014/10/how-i-created-two-images-with-same-md5.html>
- Hack the Hash, Nathaniel McHugh, 050615
  - <https://speakerdeck.com/natmchugh/hack-the-hash>
- Shattered, Marc Stevens, Pierre Karpman, Elie Bursztein, Ange Albertini, Yarik Markov, 2017
  - <https://shattered.io/>
- SHA-1 'Fully and Practically Broken' by New Collision, Dennis Fisher, 2021
  - <https://duo.com/decipher/sha-1-fully-and-practically-broken-by-new-collision>

# Class4crypt c4c7.5

## Módulo 7. Funciones hash

### Lección 7.5. SHA-2, SHA-3 y resumen de funciones hash

7.5.1. Resúmenes SHA-224, SHA-256, SHA-384 y SHA-512

7.5.2. Construcción esponja

7.5.3. Primitiva criptográfica Keccak

7.5.4. El algoritmo SHA-3

7.5.5. Resumen comparativo MD5, SHA-1, SHA-2 y SHA-3

Class4crypt c4c7.5 SHA-2, SHA-3 y resumen de funciones hash  
<https://www.youtube.com/watch?v=maeFvCJUc9g>



# Construcción Merkle-Damgård y SHA-2

- La familia de hashes SHA-2 se basa en la construcción Merkle-Damgård, usada por MD5 y SHA-1
- Para repasar estos conceptos, puede ver la lección c4c7.2 Función hash MD5: estructura y operaciones, o bien la presentación entregada y publicada en PDF

## Módulo 7. Funciones hash

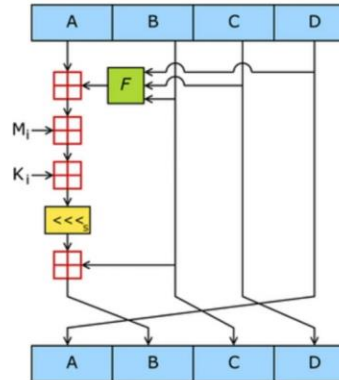
### Lección 7.2 Función hash MD5: estructura y operaciones



Clase c4c7.2  
27/04/2021



## Funciones en cada vuelta de MD5



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


Dependiendo de la ronda, F puede ser F, G, H, I:

F = (B AND C) OR (NOT B AND D) 1ª ronda

G = (B AND D) OR (C AND NOT D) 2ª ronda

H = (B XOR C XOR D) 3ª ronda

I = (C XOR (B OR NOT D)) 4ª ronda

 es una suma mod  $2^{32}$ , no un XOR

- Funciones en cada una de las 4 vueltas o rondas
- FF (a,b,c,d,Mj,tj,s)  $a = b + ((a + F(b,c,d) + Mj + tj) \lll s)$
- GG (a,b,c,d,Mj,tj,s)  $a = b + ((a + G(b,c,d) + Mj + tj) \lll s)$
- HH (a,b,c,d,Mj,tj,s)  $a = b + ((a + H(b,c,d) + Mj + tj) \lll s)$
- II (a,b,c,d,Mj,tj,s)  $a = b + ((a + I(b,c,d) + Mj + tj) \lll s)$

Class4crypt c4c7.2 - © jorgeramio 2021

Lección 7.2 - página 20

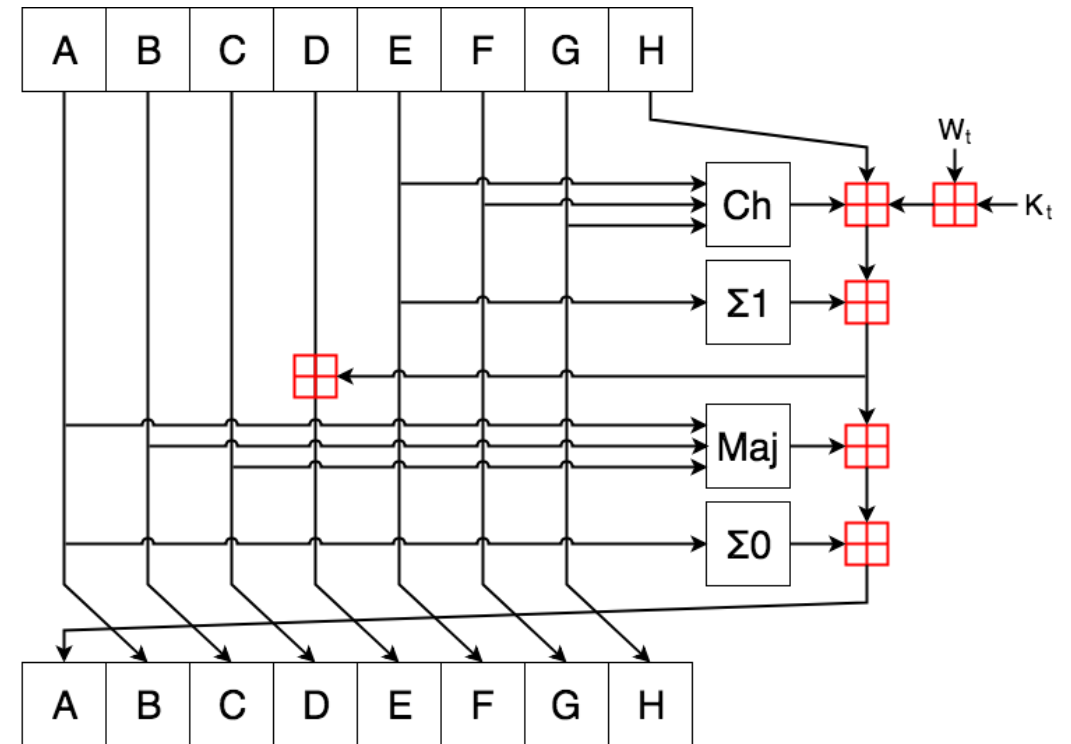
<https://www.youtube.com/watch?v=rkBVDhfCy1c>

# Características de la familia SHA-2

- A la vista de futuras vulnerabilidades en SHA-1, la NSA crea en 2001 una familia de hashes conocida como SHA-2
- SHA-2 entrega resúmenes de 224, 256, 384 y 512 bits
- El SHA-224 es el SHA-256 truncado a los 224 bits de la derecha
- El SHA-384 es el SHA-512 truncado a los 384 bits de la derecha
- SHA-256 y su variante SHA-224 emplean palabras de 32 bits
- SHA-512 y su variante SHA-384 emplean palabras de 64 bits
- Las variantes SHA-512/224 y SHA-512/256 son recomendadas en vez de SHA-224 y SHA-256, al estar basadas en SHA-512 y tener mayor resistencia a los ataques por extensión de longitud

# Esquema genérico de la función SHA-2

- SHA-256
  - Bloques de 512 bits, con 64 vueltas, vectores y palabras de 32 bits, suma mod  $2^{32}$  y un mensaje máximo de  $2^{64}-1$  bits
- SHA-512
  - Bloques de 1.024 bits, con 80 vueltas, vectores y palabras de 64 bits, suma mod  $2^{64}$  y un mensaje máximo de  $2^{128}-1$  bits
- Ch (Choose) y Maj (Majority): son operaciones lógicas AND y XOR
- $\Sigma 1$  y  $\Sigma 0$ : XOR repetido sobre el mismo registro con tres desplazamientos



# Datos y operaciones en SHA-256

- Vector de inicio (32 bits parte decimal de la raíz cuadrada primos [2, 19])
  - $A = 6a09e667$     $B = bb67ae85$     $C = 3c6ef372$     $D = a54ff53a$
  - $E = 510e527f$     $F = 9b05688c$     $G = 1f83d9ab$     $H = 5be0cd19$
- Terminadas las 16 palabras  $W_t$  de 32 bits del bloque ( $16 \cdot 32 = 512$ ), se extienden a un total de 64 palabras (para los 64 pasos de cada bloque) haciendo operaciones XOR, rotaciones y desplazamiento de palabras anteriores
- $K_t$  es una constante de 32 bits basada en los primeros 64 primos en [2, 311]
- $Ch(E, F, G) = (E \text{ AND } F) \text{ XOR } (\text{NOT } E \text{ AND } G)$
- $Maj(A, B, C) = (A \text{ AND } B) \text{ XOR } (A \text{ AND } C) \text{ XOR } (B \text{ AND } C)$
- $\Sigma_0(A) = (A \ggg 2) \text{ XOR } (A \ggg 13) \text{ XOR } (A \ggg 22)$
- $\Sigma_1(E) = (E \ggg 6) \text{ XOR } (E \ggg 11) \text{ XOR } (E \ggg 25)$

# Datos y operaciones en SHA-512

- Vector de inicio (64 bits parte decimal de la raíz cuadrada primos [2, 19])
  - $A = 6a09e667f3bcc908$     $B = bb67ae8584caa73b$     $C = 3c6ef372fe94f82b$
  - $D = a54ff53a5f1d36f1$     $E = 510e527fade682d1$     $F = 9b05688c2b3e6c1f$
  - $G = 1f83d9abfb41bd6b$     $H = 5be0cd19137e2179$
- Terminadas las 16 palabras  $W_t$  de 64 bits del bloque ( $16 \cdot 64 = 1.024$ ), se extienden a un total de 80 palabras (para los 80 pasos de cada bloque) haciendo operaciones XOR, rotaciones y desplazamiento de palabras anteriores
- $K_t$  es una constante de 64 bits basada en los primeros 80 primos en [2, 409]
- $Ch(E, F, G) = (E \text{ AND } F) \text{ XOR } (\text{NOT } E \text{ AND } G)$
- $Maj(A, B, C) = (A \text{ AND } B) \text{ XOR } (A \text{ AND } C) \text{ XOR } (B \text{ AND } C)$
- $\Sigma_0(A) = (A \ggg 28) \text{ XOR } (A \ggg 34) \text{ XOR } (A \ggg 39)$
- $\Sigma_1(E) = (E \ggg 14) \text{ XOR } (E \ggg 18) \text{ XOR } (E \ggg 41)$

# Resumen características familia SHA-2

Algoritmo y variante		Tamaño del hash (bits)	Longitud de la palabra (bits)	Tamaño del bloque (bits)	Vueltas	Tamaño del vector interno (bits)	Tamaño máximo del mensaje (bits)
SHA-2	SHA-224	224	32	512	64	256	$2^{64} - 1$
	SHA-256	256				(8 x 32)	
	SHA-384	384	64	1.024	80	512	$2^{128} - 1$
	SHA-512	512				(8 x 64)	
	SHA-512/224	224					
	SHA-512/256	256					

- Los ataques de preimagen o colisiones actuales sólo afectan a un número concreto de vueltas de SHA-2, inferior al total empleado realmente
- A fecha de hoy, mayo de 2021, SHA-256 sigue siendo el algoritmo de hash de referencia implantado en navegadores web (https), software de correo electrónico, aplicaciones, tarjetas inteligentes, etc.

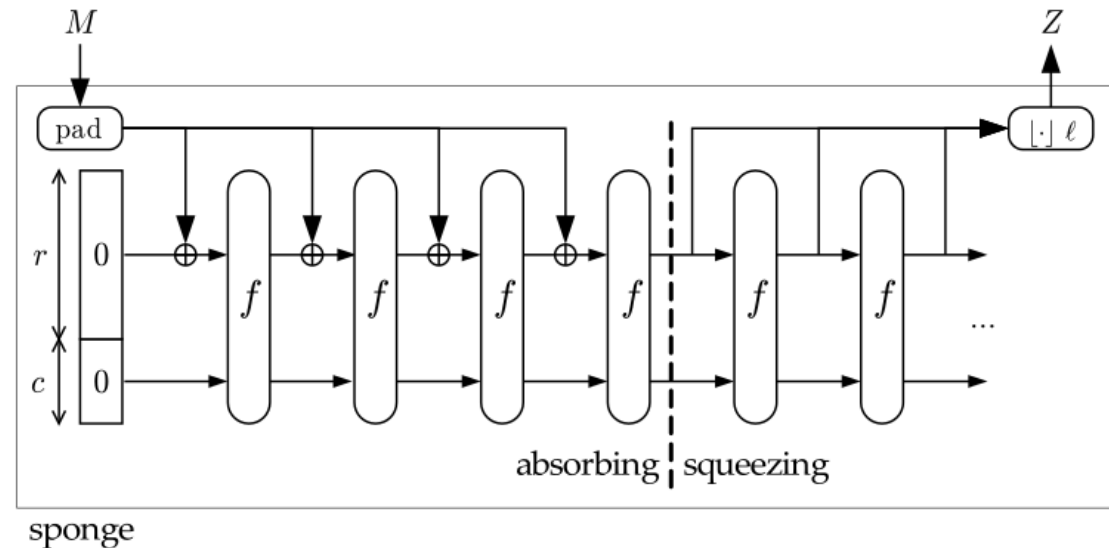
# El NIST busca un nuevo estándar de hash

- Ante los ataques recibidos por MD5 y la consiguiente alarma mostrada por expertos mundiales con los hashes de construcción Merkle-Damgård, el NIST publica una nota de prensa el 23 de enero 2007 con el borrador de la llamada a concurso para encontrar el nuevo estándar de hash SHA-3
- Tres años después, en diciembre de 2010, el NIST selecciona en la tercera ronda a cinco finalistas: BLAKE, Ghøstl, JH, Keccak y Skein
- En octubre de 2012, el NIST anuncia que el ganador es Keccak, desarrollado por los investigadores belgas Joan Daemen, Michael Peeters, Gilles Van Assche y el italiano Guido Bertoni
- Los autores publican en el NIST el estándar de hash en febrero de 2013
- SHA-3 es publicado como estándar oficial por el NIST en agosto de 2015



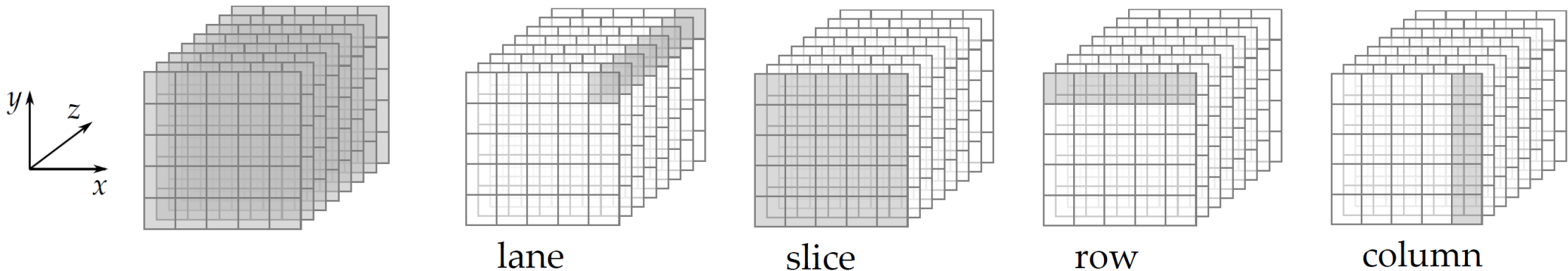
# Construcción esponja

- Construcción esponja: función  $F(M) = h(M)$  para hash SHA-3
  - Entrada de longitud variable y salida de longitud variable
  - Permutación (o transformación)  $f$  de longitud fija
  - Número de bits o estado =  $b$ , donde  $b = r + c$  ( $r$  bloque o *rate* y  $c$  *capacity*)
- Bloques  $M$  + relleno
  - Bloques de  $r$  bits
  - $S$  = estado ( $b$  bits)
  - Inicializado a cero
- Fase de absorción (entrada)
- Fase de exprimido (salida)



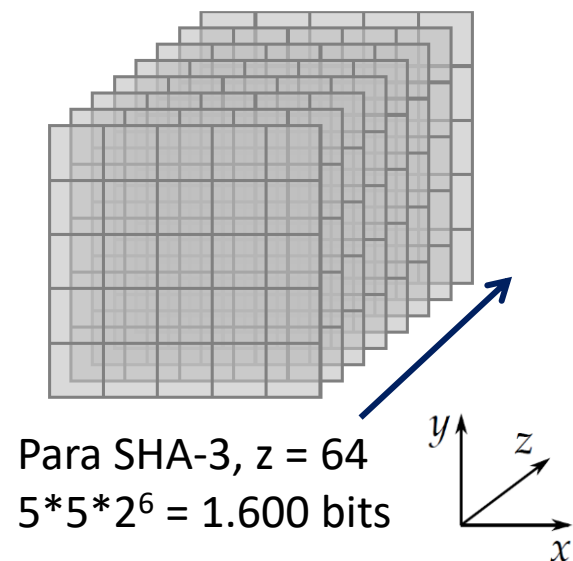


# Estados de Keccak



- En cada celda puede haber un solo bit
- Rebanadas o caras (slice) cada una con  $5 \times 5 = 25$  bits
- $5 \times 5$  carriles o pistas (lane) cada uno con  $2^L$  bits ( $L = 0, 1, 2, 3, 4, 5, 6$ )
- La función  $f$  contempla 5 operaciones sobre bits en las tres dimensiones x, y, z (XOR, AND, NOT) conocidas como  $\theta$  (theta),  $\rho$  (rho),  $\pi$  (pi),  $\chi$  (chi) y  $\iota$  (iota) con un número de vueltas igual a  $12 + 2 \times L$
- En los ejes x, y las operaciones serán en mod 5 y en el eje z en mod  $2^L$

# SHA-3



- Usaremos 64 carriles ( $L = 6$ ,  $2^L = 2^6 = 64$ ) por lo que se trabajará en un estado con  $5 \cdot 5 \cdot 2^6 = 1.600$  bits
- Los valores de bloque  $r$  a tratar vendrán dados por el hash que se desee, que por compatibilidad con SHA-2 serán 224, 256, 384 y 512 bits
- Como el bloque  $r$  será mayor que el hash deseado, como hash se eligen los primeros  $n$  bits de esos  $r$  bits

- La capacidad determina la seguridad del esquema
- La seguridad ante las colisiones por paradoja del cumpleaños será  $n/2$

Salida bits SHA-3	b (bits con $L = 64$ )	r (rate/bloque)	c (capacity)
$n = 224$	1.600	1.152	448
$n = 256$	1.600	1.088	512
$n = 384$	1.600	832	768
$n = 512$	1.600	576	1.024

# Resumen SHA-3

Instance	Output size $d$	Rate $r$ = block size	Capacity $c$	Definition	Security strengths in bits		
					Collision	Preimage	2nd preimage
SHA3-224( $M$ )	224	1152	448	Keccak[448]( $M \parallel 01$ , 224)	112	224	224
SHA3-256( $M$ )	256	1088	512	Keccak[512]( $M \parallel 01$ , 256)	128	256	256
SHA3-384( $M$ )	384	832	768	Keccak[768]( $M \parallel 01$ , 384)	192	384	384
SHA3-512( $M$ )	512	576	1024	Keccak[1024]( $M \parallel 01$ , 512)	256	512	512

- La capacidad  $c$  no se ve afectada por la entrada/salida
- El NIST sugiere en 2013 bajar los valores de la capacidad  $c$ , que dan fortaleza al hash ante ataques, pero ante la oposición de la comunidad científica, ratifica los valores originales de sus autores, que son los mostrados en la figura
- El nivel máximo de seguridad del hash será la mitad de esa capacidad  $c/2$

# Velocidad MD5, SHA-1, SHA-256, SHA-512

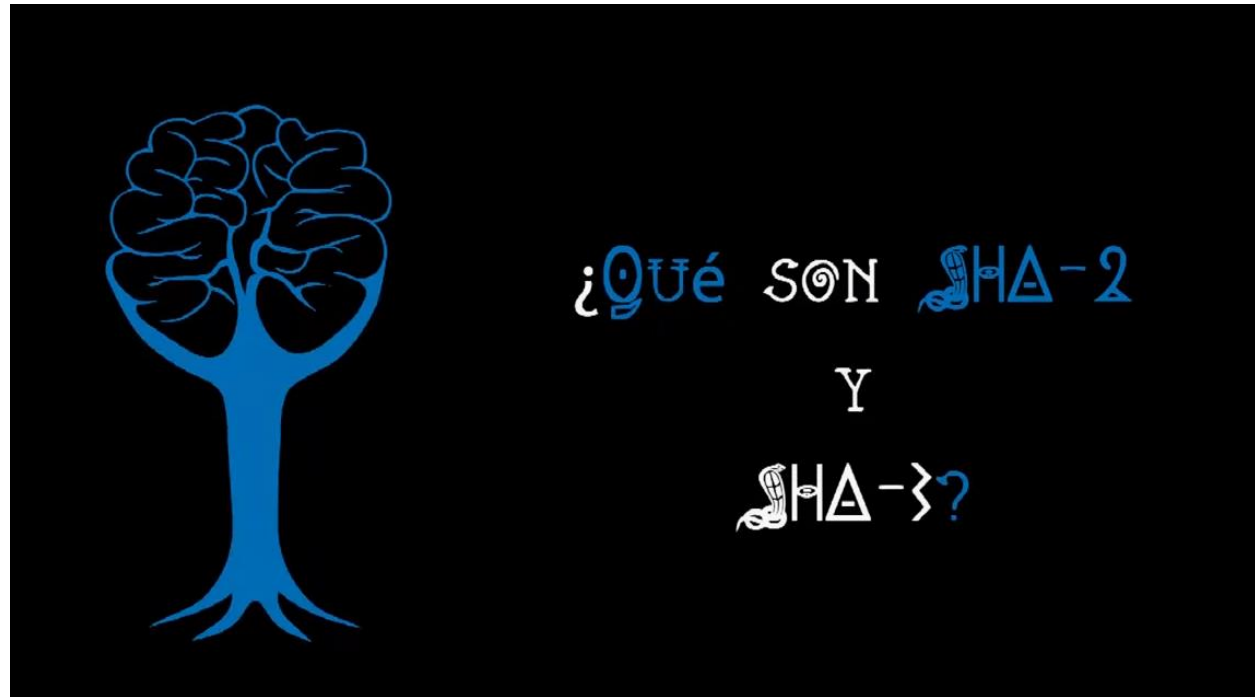
- C:\Program Files\OpenSSL-Win64\bin>openssl speed md5 sha1 sha256 sha512
- Doing md5 for 3s on 16 size blocks: 26531121 md5's in 3.02s... etc.
- Doing sha1 for 3s on 16 size blocks: 26633058 sha1's in 3.00s... etc.
- Doing sha256 for 3s on 16 size blocks: 16162872 sha256's in 2.95s... etc.
- Doing sha512 for 3s on 16 size blocks: 11840646 sha512's in 3.02s... etc.
- OpenSSL 1.1.1k 25 Mar 2021
- built on: Fri Mar 26 01:21:29 2021 UTC
- The 'numbers' are in 1000s of bytes per second processed.
- | type   | 16 bytes   | 64 bytes   | 256 bytes  | 1024 bytes | 8192 bytes  | 16384 bytes |      |
|--------|------------|------------|------------|------------|-------------|-------------|------|
| md5    | 140766.15k | 332584.57k | 586496.01k | 728391.68k | 780781.64k  | 784057.76k  | (2º) |
| sha1   | 142042.98k | 347354.42k | 719203.61k | 983454.72k | 1088995.07k | 1106036.86k | (1º) |
| sha256 | 87570.27k  | 198132.86k | 379088.27k | 475598.15k | 516243.46k  | 517067.49k  | (4º) |
| sha512 | 62822.91k  | 253749.43k | 430817.53k | 649378.91k | 762947.70k  | 772795.08k  | (3º) |
- SHA-3 es más lento que SHA-2, ver siguiente diapositiva

# Resumen características de funciones hash

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Operations	Security against <u>collision</u> <u>attacks</u> (bits)	Security against <u>length</u> <u>extension</u> <u>attacks</u> (bits)	Performance on <u>Skylake</u> (median <u>cpb</u> )		First published
									Long messages	8 bytes	
<u>MD5</u> (as reference)		128	128 (4 × 32)	512	64	And, Xor, Rot, Add (mod 2 <sup>32</sup> ), Or	≤ 18 (collisions found)	0	4.99	55.00	1992
<u>SHA-0</u>		160	160 (5 × 32)	512	80	And, Xor, Rot, Add (mod 2 <sup>32</sup> ), Or	< 34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
<u>SHA-1</u>							< 63 (collisions found)		3.47	52.00	1995
<u>SHA-2</u>	SHA-224	224	256 (8 × 32)	512	64	And, Xor, Rot, Add (mod 2 <sup>32</sup> ), Or, Shr	112	32	7.62	84.50	2004
	SHA-256	256					128	0	7.63	85.25	2001
	SHA-384	384	512 (8 × 64)	1024	80	And, Xor, Rot, Add (mod 2 <sup>64</sup> ), Or, Shr	192	128 (≤ 384)	5.12	135.75	2001
	SHA-512	512					256	0	5.06	135.50	
	SHA-512/224	224					112	288	≈ SHA-384	≈ SHA-384	2012
	SHA-512/256	256					128	256			
<u>SHA-3</u>	SHA3-224	224	1600 (5 × 5 × 64)	1152	24	And, Xor, Rot, Not	112	448	8.12	154.25	2015
	SHA3-256	256					128	512	8.59	155.50	
	SHA3-384	384					192	768	11.06	164.00	
	SHA3-512	512					256	1024	15.88	164.00	
	SHAKE128	d (arbitrary)	1344 1088	min(d/2, 128)	256	7.08	155.25				
	SHAKE256	d (arbitrary)			512	8.59	155.50				
						min(d/2, 256)					

Ref: Wikipedia

# Más información en píldoras Thoth



<https://www.youtube.com/watch?v=hEa-IC1-JQI>

# Conclusiones de la Lección 7.5

- La familia SHA-2 propuesta por la NSA en 2001, y algunas variantes en 2004 y 2012, usa también la construcción Merkle-Damgård usada por MD5 y SHA-1
- Las más conocidas son SHA-224, SHA-256, SHA-384 y SHA-512
- Son funciones hash más seguras que SHA-1, trabajan con 32 y 64 bits, siendo actualmente la más conocida y utilizada SHA-256
- El algoritmo SHA-3 usa una construcción esponja que tiene dos fases, la de absorción y la de exprimido, y que entrega un número variable de bits según las necesidades del usuario, utilizando el algoritmo Keccak
- Para el hash SHA-3, Keccak limita la salida a un único bloque en el que reduce su tamaño al valor del hash deseado, es decir, 224, 256, 384 o 512 bits
- El NIST lo establece como estándar en 2015 pero su uso sigue siendo escaso

# Lectura recomendada (1/2)

- Guion píldora formativa Thoth nº 46, ¿Qué son SHA-2 y SHA-3?, Jorge Ramió, 2017
  - <https://www.criptored.es/thoth/material/texto/pildora046.pdf>
- Criptografía: Función SHA-256, Javier Domínguez Gómez, 2018
  - [https://academy.bit2me.com/wp-content/uploads/2019/10/Criptography\\_SHA\\_256\\_es.pdf](https://academy.bit2me.com/wp-content/uploads/2019/10/Criptography_SHA_256_es.pdf)
- SHA-2, Wikipedia
  - <https://es.wikipedia.org/wiki/SHA-2>
- US Secure Hash Algorithms, Internet Engineering Task Force IETF, ISSN 2070-1721, May 2011
  - <https://datatracker.ietf.org/doc/html/rfc6234#page-51>
- Convertir de decimal a hexadecimal con parte fraccionaria - Ejercicio #3, Pasos por ingeniería, 2017
  - [https://www.youtube.com/watch?v=2Z4Xa5\\_MIUU](https://www.youtube.com/watch?v=2Z4Xa5_MIUU)
- Rainbow Tables: A Path to Password Gold for Cybercriminals, Hashedout, Casey Crane, 2021
  - <https://www.thesslstore.com/blog/rainbow-tables-a-path-to-password-gold-for-cybercriminals/>
- Hash Functions, SHA-3 Project, NIST
  - <https://csrc.nist.gov/projects/hash-functions/sha-3-project>



# Lectura recomendada (2/2)

- SHA-3, Wikipedia
  - <https://en.wikipedia.org/wiki/SHA-3>
- Keccak and the SHA-3 Standardization, Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, 2013
  - <https://csrc.nist.gov/csrc/media/projects/hash-functions/documents/keccak-slides-at-nist.pdf>
- Lecture 21 (update): SHA-3 Hash Function, Christof Paar, 2017
  - <https://www.youtube.com/watch?v=JWskjzgila4>
- SHA-3 and The Hash Function Keccak, An extension chapter for Understanding Cryptography, Christof Paar, Jan Pelzl,
  - <http://professor.unisinos.br/linds/teoinfo/Keccak.pdf>
- Some notes on SHA-3, Stefan's Blog
  - [https://stbuehler.de/blog/article/2014/05/15/some\\_notes\\_on\\_sha-3.html](https://stbuehler.de/blog/article/2014/05/15/some_notes_on_sha-3.html)
- BLAKE2 - fast secure hashing
  - <https://www.blake2.net/>