Universidad de Oviedo

Departamento de Informática

Campus de Gijón

Funciones de resumen (hash)

Presentación

Daniel F. García

Función hash o resumen (1)

Una función hash H es un procedimiento o función matemática
que genera un resumen h de tamaño fijo (varios bytes)
a partir de un mensaje M de tamaño variable

Los valores h devueltos por una función hash H
h = H(M)

se denominan →

- hash values
- hash codes
- hash sums
- checksums
- hashes
- digests

Propiedades generales que debe tener una función hash:

1. Bajo coste computacional
   La facilidad de cálculo debe ser máxima
2. Determinismo
   Para un mensaje de entrada dado la función debe generar siempre el mismo resumen
3. Uniformidad
   La función debe proyectar (mapear) el rango de valores de entrada en el rango de valores de salida tan uniformemente como sea posible

   Esto disminuirá el número de **colisiones**

Función hash o resumen (2)

Se produce una **colisión** cuando una función hash genera
la misma salida (resumen) para dos o más entradas distintas

Propiedades adicionales que debe tener una función hash:
(cuando se utiliza en un sistema criptográfico)

4. Unidireccionalidad

Conocido un resumen h = H(M) es computacionalmente imposible encontrar M a partir de h

$M = H^{-1}(h)$ es imposible de calcular
h = H(M) es muy fácil de calcular

5. Difusión

El resumen H(M) debe ser una función compleja de todos los bits del mensaje M
Si se modifica un solo bit de M → Deberían cambiar la mitad de los bits del resumen

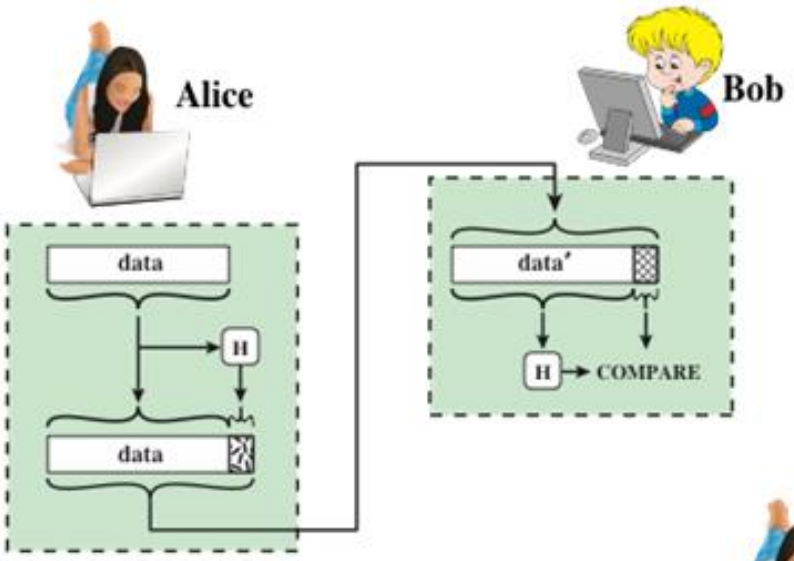
6. Resistencia débil a colisiones

Conocido un M será imposible encontrar un M' tal que H(M) = H(M')

7. Resistencia fuerte a colisiones

Será difícil generar un par (M, M') tal que H(M) = H(M')

Aplicación de F. Hash: Autenticación de mensajes (1)



Autenticar un M ←→ Verificar la integridad del M

Alice calcula H(data) y envía data+H(data)

Bob calcula H(data') y lo compara con H recibido

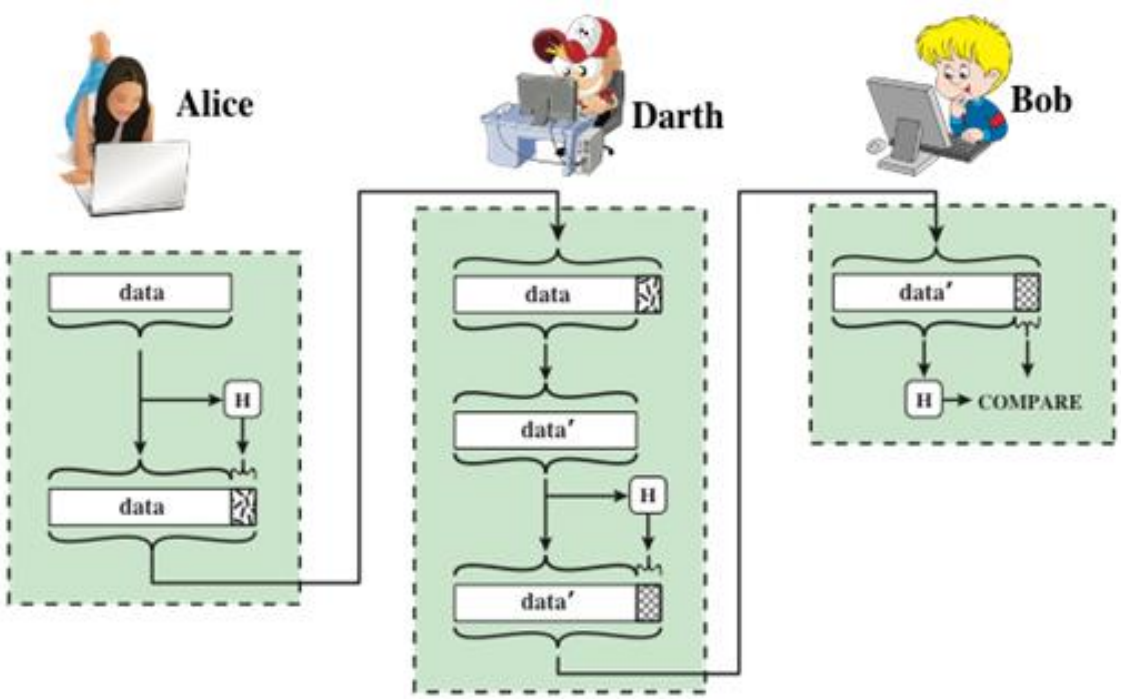
SI H(data') ≠ H recibido
ENTONCES data y/o hash alterados

PERO …

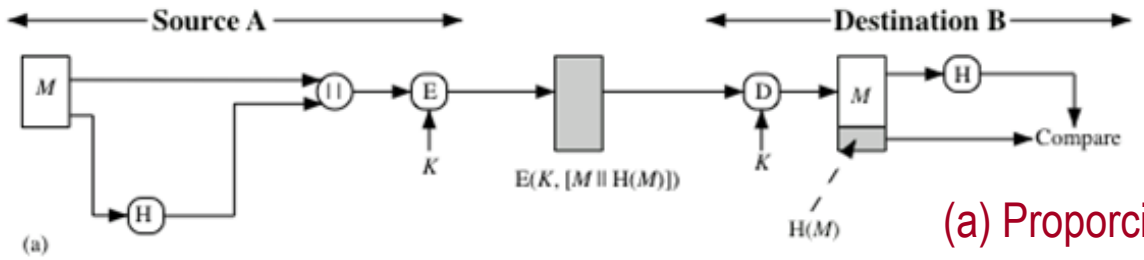
¡Hay que proteger el hash!

Ataque clásico: Man-in-the-middle

Darth intercepta el mensaje
modifica data, recalcula el hash
y lo reenvía a Bob

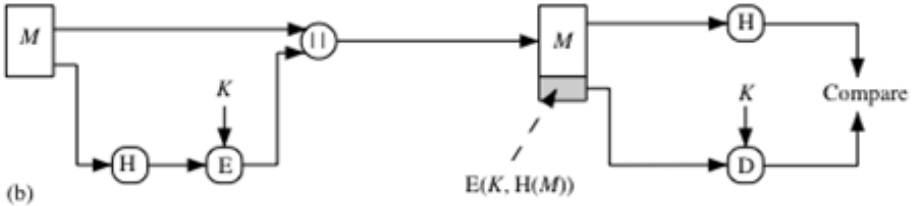


Aplicación de F. Hash: Autenticación de mensajes (2)



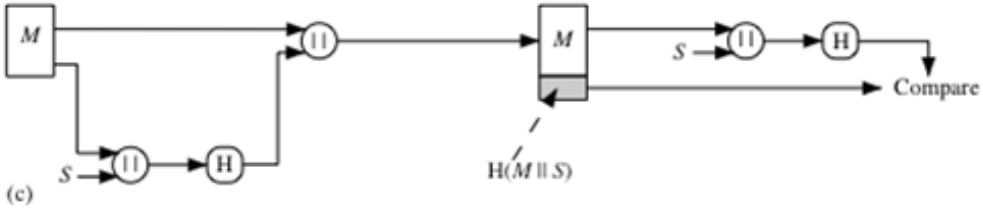
Se protege el mensaje y el hash con cifrado simétrico

(a) Proporciona confidencialidad e integridad



Solo se protege el hash con cifrado simétrico

(b) Proporciona integridad

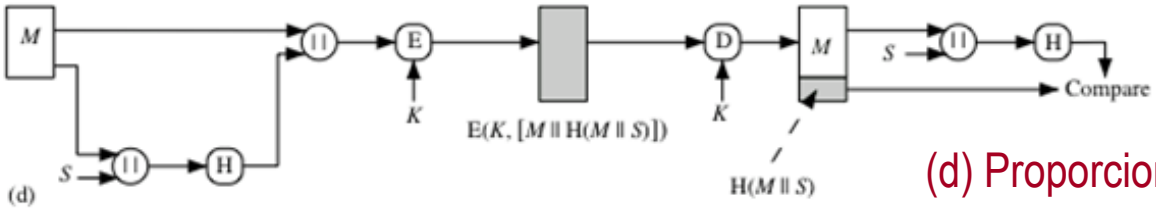


Se añade un secreto compartido S a M antes de calcular el hash

Un oponente no tiene S para recalcular el hash

Ventaja: no hay que cifrar nada

(c) Proporciona integridad

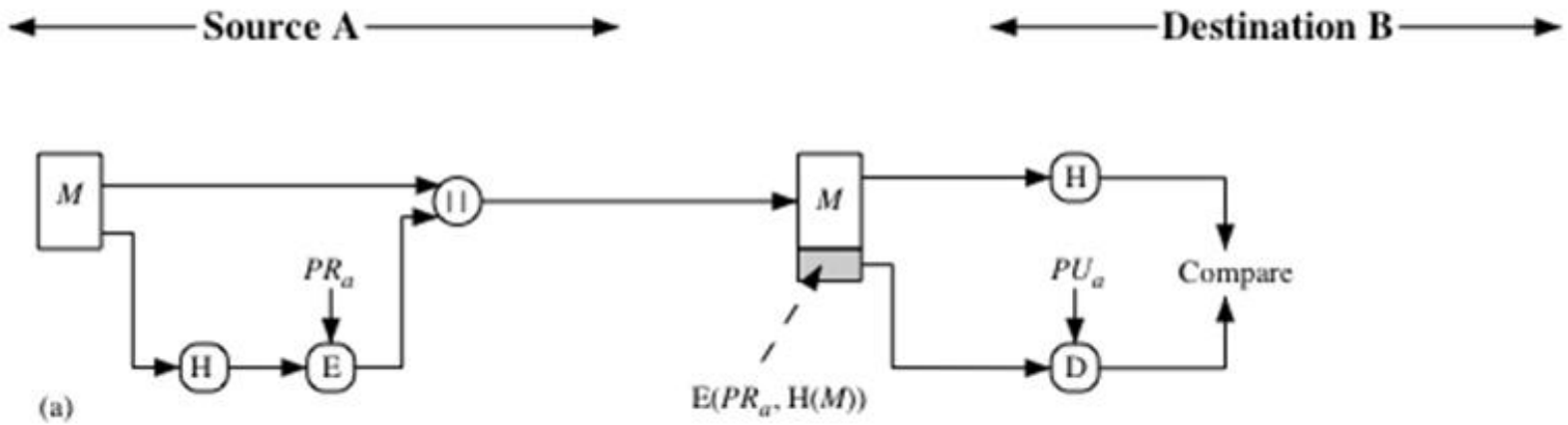


Añade cifrado simétrico al esquema anterior

(d) Proporciona confidencialidad e integridad

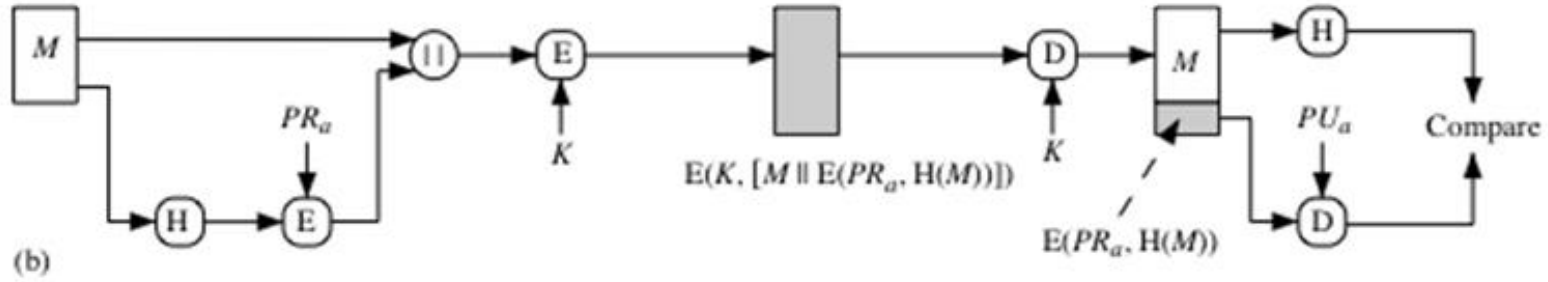
Aplicación de F. Hash: Firma Digital

Firmar un M → Permite verificar el remitente de M además de la integridad de M



A cifra H(M) con la clave privada de A (*PRa*)

B descifra H(M) con la clave pública de A (*PUa*)
Si Compare OK → El mensaje está integro y es de A (solo A dispone de *PRa* para cifrarlo)



(b) Cifra (mensaje + firma) con Alg. simétrico para proporcionar confidencialidad

Funciones de hash MD (Message Digest)

Las funciones MD han sido diseñadas por Ron Rivest (y otros)

1989 **MD2** Genera un hash de 128 bits (16 bytes)
Se han encontrado vulnerabilidades y en 2009 se deshabilitó su uso
https://www.rfc-editor.org/rfc/rfc1329

1990 **MD4** Genera un hash de 128 bits (16 bytes)
También se han encontrado muchas vulnerabilidades
https://www.rfc-editor.org/rfc/rfc1320

1992 **MD5** Genera un hash de 128 bits (16 bytes) - Aún es usado en la actualidad
El US-CERT lo considera criptográficamente roto en 2010
Y recomienda usar funciones hash de la familia SHA-2
https://www.rfc-editor.org/rfc/rfc1321

US-CERT = US Computer Emergency Readiness Team https://www.cisa.gov/uscert/

2008 **MD6** Genera un hash variable de 0 a 512 bits
Fue retirada de la competición que mantuvo el NIST para elegir la función SHA-3
Actualmente no es utilizada al disponerse de alternativas mejores

SHA: Secure Hash Algorithm

Las funciones de resumen SHA han sido diseñadas por encargo de la NSA (NSA, *National Security Agency of USA*)

Publicadas por el NIST (*National Institute of Standards and Technology*)

Las funciones SHA conforman el estándar SHS de US-FIPS (*Federal Information Processing Standards*)
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

Alg. SHA	Año	Longit Entrada	Longit Salida
SHA-0	1993	2^{64}-1	160
SHA-1	1995	2^{64}-1	160
SHA-2	2001		
SHA-224		2^{64}-1	224
SHA-256		2^{64}-1	256
SHA-384		2^{128}-1	384
SHA-512		2^{128}-1	512
SHA3-224	2012	∞	224
SHA3-256		∞	256
SHA3-384		∞	384
SHA3-512		∞	512
SHAKE128		∞	Elegir
SHAKE256		∞	Elegir

Hay 4 “familias” de algoritmos SHA → SHA-0, SHA-1, SHA-2, SHA-3

No usar SHA-0 ni SHA-1

El 2-Oct-2012 el NIST eligió el algoritmo Keccak como el nuevo algoritmo SHA-3
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

Detalles del Alg. Keccak:
https://keccak.team/

SHA-1 Funcionamiento (1)

El algoritmo SHA-1 utiliza como entrada bloques de 512 bits y genera un resumen de 160 bits

- Para calcular el resumen de un bloque realiza 80 iteraciones
- En cada iteración realiza una serie de operaciones

Hay una fase de preprocesamiento → { Rellenar el mensaje M; Dividir el M rellenado en bloques; Establecer el valor inicial del hash H^0

Rellenar el mensaje

Suponer que el mensaje M tiene L bits

- Añadir al final del mensaje un bit "1" seguido de K bits "0"

  K es la menor solución no negativa de la ecuación: $L+1+K \equiv 448 \bmod 512$

- Añadir un bloque de 64 bits que representa L en binario

Ejemplo: M="abc" que en código ASCII tiene una longitud: 8x3=24 bits

Se rellena con un bit "1" y 448 – (24+1) = 423 bits "0"

Se añaden 64 bits que representan 24 en binario

$$\underbrace{0110\ 0001}_{a}\ \underbrace{0110\ 0010}_{b}\ \underbrace{0110\ 0011}_{c}\ 1\ \underbrace{00\cdots00}_{423}\ \underbrace{00\cdots011000}_{64}$$ ← L=24

SHA-1 Funcionamiento (2)

Dividir el mensaje rellenado en bloques

El mensaje M se divide en N bloques de 512 bits/bloque (64 bytes/bloque)

$M \rightarrow M^1, M^2, \ldots, M^N$

Cada bloque M^i se divide en 16 palabras de 32 bits (4 bytes)

$M^i \rightarrow M^i_0, M^i_1, \ldots, M^i_{15}$

Establecer el valor inicial del hash H^0

El hash de SHA-1 es de 160 bits = 5 palabras de 32 bits

El estándar de FIPS especifica los valores siguientes:

$H^0_0 = 6745\ 2301$
$H^0_1 = EFCD\ AB89$
$H^0_2 = 98BA\ DCFE$
$H^0_3 = 1032\ 5476$
$H^0_4 = C3D2\ E1F0$

SHA-1 Funcionamiento (3)

FASE del cálculo del hash

Cada bloque del mensaje (M^1, M^2, … , M^N) se procesa en orden

FOR i=1 to N
{

1) Preparar la secuencia de palabras W^i_t a partir del bloque M^i

2) Inicializar 5 variables de trabajo: A, B, C, D y E
Con el hash calculado para el bloque i-1

3) FOR t = 0 to 79
Transformar el Hash inicial que contienen A, B, C, D y E en 80 iteraciones

4) Calcular el i-ésimo valor intermedio del hash H^i

$$\left.\begin{aligned} H_0^i &= H_0^{i-1} + A \\ H_1^i &= H_1^{i-1} + B \\ H_2^i &= H_2^{i-1} + C \\ H_3^i &= H_3^{i-1} + D \\ H_4^i &= H_4^{i-1} + E \end{aligned}\right\} \rightarrow \text{Las sumas se realizan en módulo } 2^{32}$$

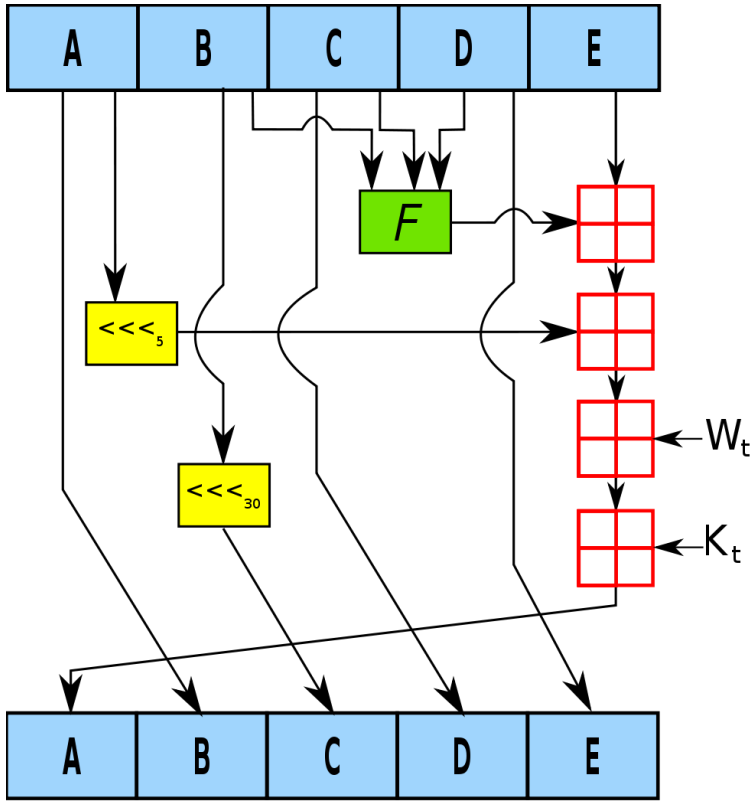
} // Fin del FOR

SHA-1 Funcionamiento (4)

Descripción de una iteración de transformación del hash

Se explica el cuerpo del bucle: 3) FOR t = 0 to 79

En cada iteración t el hash inicial se transforma en un nuevo hash así:



En cada iteración t la palabra de 160 bits que contiene el hash es rotada circularmente a la derecha 32 bits

Pero NO se hace una rotación simple con dos palabras B y E son transformadas durante la rotación

La palabra B es rotada circularmente a la izquierda 30 bits antes de copiarla en la palabra C

La palabra E sufre 4 transformaciones

1) Se le suma (en mod 2^{32}) la salida de F(t):

Ch(B,C,D) = (B AND C) XOR (NOT(B) AND D) *(0≤t≤19)*

Parity(B,C,D) = B XOR C XOR D *(20≤t≤39)*

Maj(B,C,D) = (B AND C) XOR (B AND D) XOR (C AND D) *(40≤t≤59)*

Parity(B,C,D) = B XOR C XOR D *(60≤t≤79)*

2) Se le suma A rotada circularmente 5 bits a la izquierda

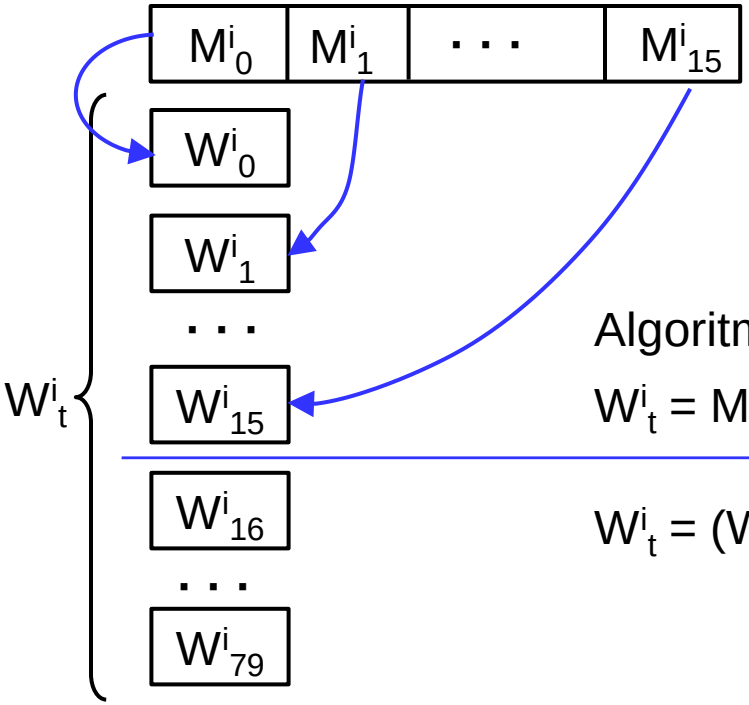
3) Se le suma W_t ← Aquí se integra la información a resumir

4) Se le suma K_t

SHA-1 Funcionamiento (5)

Preparación de la secuencia de palabras W^i_t

Cada bloque M^i formado por 16 palabras de 32 bits se expande en 80 palabras



Expansión en 80 palabras W_t de 32 bits

Algoritmo de expansión:

$W^i_t = M^i_t \quad (0 \le t \le 15)$

$W^i_t = (W^i_{t-3} \oplus W^i_{t-8} \oplus W^i_{t-14} \oplus W^i_{t-16}) <<<1 \quad (16 \le t \le 79)$

Constantes a sumar en las iteraciones K^i_t

La constante depende del número de iteración →

$$\begin{cases} K_t = 5A82\ 7999 & (0 \le t \le 19) \\ K_t = 6ED9\ EBA1 & (20 \le t \le 39) \\ K_t = 8F1B\ BCDC & (40 \le t \le 59) \\ K_t = CA62\ C1D6 & (60 \le t \le 79) \end{cases}$$

Son palabras de 32 bits y se suman en mod 2^{32}

SHA-512 Funcionamiento (1)

El algoritmo SHA-512 utiliza como entrada bloques de 1024 bits y genera un resumen de 512 bits

- Para calcular el resumen de un bloque realiza 80 iteraciones
- En cada iteración realiza una serie de operaciones

Hay una fase de preprocesamiento → { Rellenar el mensaje M; Dividir el M rellenado en bloques; Establecer el valor inicial del hash H^0

Rellenar el mensaje

Suponer que el mensaje M tiene L bits

- Añadir al final del mensaje un bit "1" seguido de K bits "0"

  K es la menor solución no negativa de la ecuación: $L+1+K \equiv 896 \bmod 1024$

- Añadir un bloque de **128** bits que representa L en binario

Ejemplo: M="abc" que en código ASCII tiene una longitud: 8x3=24 bits

Se rellena con un bit "1" y 896 – (24+1) = 871 bits "0"

Se añaden 128 bits que representan 24 en binario

$$\underbrace{0110\ 0001}_{a}\ \underbrace{0110\ 0010}_{b}\ \underbrace{0110\ 0011}_{c}\ 1\ \underbrace{00\cdots00}_{871}\ \underbrace{00\cdots011000}_{128}$$

← L=24

SHA-512 Funcionamiento (2)

Dividir el mensaje rellenado en bloques

El mensaje M se divide en N bloques de 1024 bits/bloque (128 bytes/bloque)

$M \rightarrow M^1, M^2, \ldots, M^N$

Cada bloque M^i se divide en 16 palabras de 64 bits (8 bytes)

$M^i \rightarrow M^i_0, M^i_1, \ldots, M^i_{15}$

Establecer el valor inicial del hash H^0

El hash de SHA-512 es de 512 bits = 8 palabras de 64 bits

El estándar de FIPS especifica los valores siguientes:

$H^0_0 = 6A09E667F3BCC908$
$H^0_1 = BB67AE8584CAA73B$
$H^0_2 = 3C6EF372FE94F82B$
$H^0_3 = A54FF53A5F1D36F1$
$H^0_4 = 510E527FADE682D1$
$H^0_5 = 9B05688C2B3E6C1F$
$H^0_6 = 1F83D9ABFB41BD6B$
$H^0_7 = 5BE0CD19137E2179$

SHA-512 Funcionamiento (3)

FASE del cálculo del hash

Cada bloque del mensaje (M^1, M^2, … , M^N) se procesa en orden

FOR i=1 to N
{

1) Preparar la secuencia de palabras W^i_t a partir del bloque M^i

2) Inicializar 8 variables de trabajo: A, B, C, D, E, F, G y H (cada una de 64 bits)
Con el hash calculado para el bloque i-1

3) FOR t = 0 to 79
Transformar el Hash inicial que contienen A ⋯ H en 80 iteraciones

4) Calcular el i-ésimo valor intermedio del hash H^i

$$H_0^i = H_0^{i-1} + A \qquad H_4^i = H_4^{i-1} + E$$
$$H_1^i = H_1^{i-1} + B \qquad H_5^i = H_5^{i-1} + F$$
$$H_2^i = H_2^{i-1} + C \qquad H_6^i = H_6^{i-1} + G$$
$$H_3^i = H_3^{i-1} + D \qquad H_7^i = H_7^{i-1} + H$$

→Las sumas se realizan en módulo 2^{64}

} // Fin del FOR

SHA-512 Funcionamiento (4)

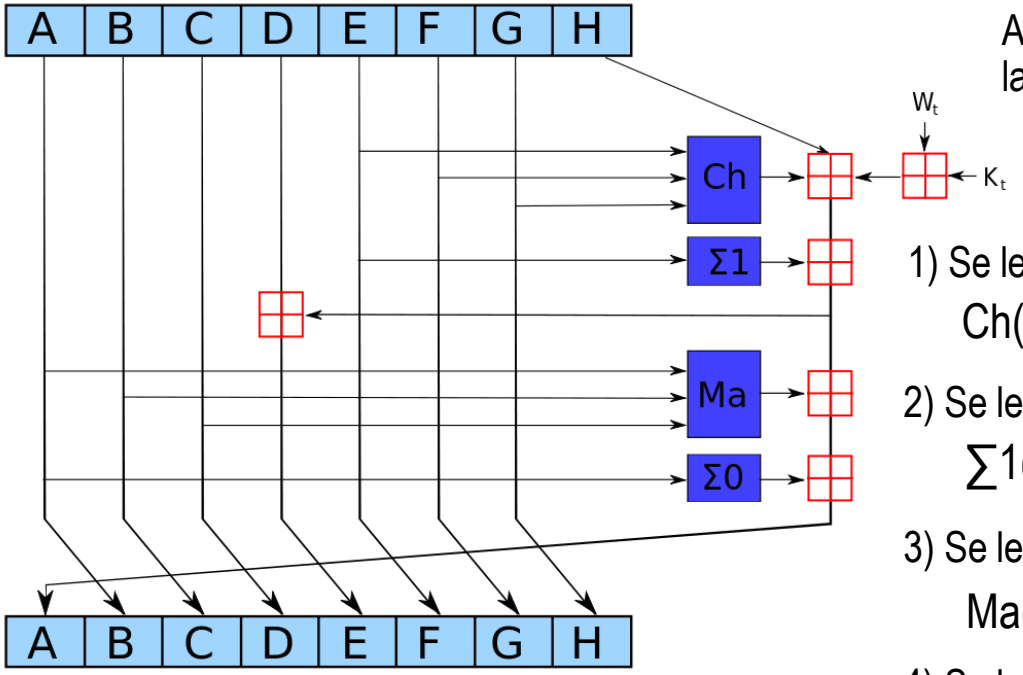
Descripción de una iteración de transformación del hash

Se explica el cuerpo del bucle: 3) FOR t = 0 to 79

En cada iteración t el hash inicial se transforma en un nuevo hash así:

En cada iteración t la palabra de 512 bits que contiene el hash es rotada circularmente a la derecha 64 bits

Pero NO se hace una rotación simple con las palabras D y H que son transformadas durante la rotación



A l**a palabra D** se le suma (en mod 2^{64}) la mitad de las transformaciones que sufre la palabra H

La palabra H sufre 4 transformaciones:

1) Se le suma (en mod 2^{64}) la salida de Ch, W_t y K_t

$Ch(E,F,G) = (E \text{ AND } F) \text{ XOR } (\text{NOT}(E) \text{ AND } G)$

2) Se le suma $\sum 1(E)$ (en mod 2^{64})

$\sum 1(E) = (E >>> 14) \text{ XOR } (E >>> 18) \text{ XOR } (E >>> 41)$

3) Se le suma Ma (en mod 2^{64})

$Ma(A,B,C) = (A \text{ AND } B) \text{ XOR } (A \text{ AND } C) \text{ XOR } (B \text{ AND } C)$

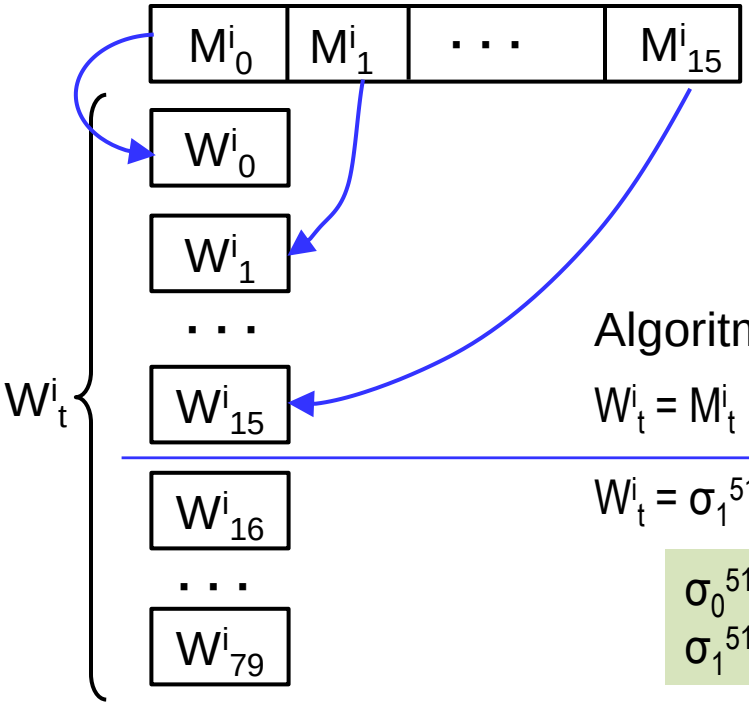
4) Se le suma $\sum 0(A)$ (en mod 2^{64})

$\sum 0(A) = (A >>> 28) \text{ XOR } (A >>> 34) \text{ XOR } (A >>> 39)$

SHA-512 Funcionamiento (5)

Preparación de la secuencia de palabras W^i_t

Cada bloque M^i formado por 16 palabras de 64 bits se expande en 80 palabras



Expansión en 80 palabras W_t de 64 bits

Algoritmo de expansión:

$W^i_t = M^i_t \quad (0 \le t \le 15)$

$W^i_t = \sigma_1^{512}(W^i_{t-2}) + W^i_{t-7} + \sigma_0^{512}(W^i_{t-15}) + W^i_{t-16} \quad (16 \le t \le 79)$

$\sigma_0^{512}(X) = (X>>>1) \oplus (X>>>8) \oplus (X>>7)$

$\sigma_1^{512}(X) = (X>>>19) \oplus (X>>>61) \oplus (X>>6)$

Constantes a sumar en las iteraciones K^i_t

La constante depende del número de iteración →

$K_0 = 428A2F98D728AE22$

$K_1 = 7137449123EF65CD$

. . .

$K_{79} = 6C44198C4A475817$

Son palabras de 64 bits y se suman en mod 2^{64}