CÓDIGOS Y CRIPTOGRAFÍA

Funciones Hash. Función resumen MD5.

Funciones resumen

Función resumen

Una función resumen, *hash* o *digest* es una función computable mediante un **algoritmo** tal que

$$\begin{array}{rccc} h: & U & \to & M \\ & x & \mapsto & h(x) \end{array}.$$

Funciones resumen

Función resumen

Una función resumen, *hash* o *digest* es una función computable mediante un **algoritmo** tal que

$$\begin{array}{rccc} h: & U & \to & M \\ & x & \mapsto & h(x) \end{array}.$$

- Tiene como entrada un conjunto de elementos, cadenas, y los convierte normalmente en cadenas de longitud fija.

Funciones resumen

Función resumen

Una función resumen, *hash* o *digest* es una función computable mediante un **algoritmo** tal que

$$\begin{array}{rccc} h: & U & \to & M \\ & x & \mapsto & h(x) \end{array}.$$

- Tiene como entrada un conjunto de elementos, cadenas, y los convierte normalmente en cadenas de longitud fija.
- La idea básica de un valor hash es que sirva como una representación compacta, o resumen, de la cadena de entrada.

Funciones resumen

Función resumen

Una función resumen, *hash* o *digest* es una función computable mediante un **algoritmo** tal que

$$\begin{aligned} h: \quad U &\to M \\ x &\mapsto h(x) \end{aligned} .$$

- Tiene como entrada un conjunto de elementos, cadenas, y los convierte normalmente en cadenas de longitud fija.
- La idea básica de un valor hash es que sirva como una representación compacta, o resumen, de la cadena de entrada.
- Hay una colisión cuando dos entradas distintas de la función resumen producen la misma salida. U puede tener infinitos elementos, pero M tiene un número finito de elementos debido a que el tamaño de sus cadenas es fijo. Por tanto, la posibilidad de existencia de colisiones es intrínseca a la definición de función hash.

Funciones resumen

Función resumen

Una función resumen, *hash* o *digest* es una función computable mediante un **algoritmo** tal que

$$\begin{aligned} h: \quad U &\to M \\ x &\mapsto h(x) \end{aligned} .$$

- Tiene como entrada un conjunto de elementos, cadenas, y los convierte normalmente en cadenas de longitud fija.
- La idea básica de un valor hash es que sirva como una representación compacta, o resumen, de la cadena de entrada.
- Hay una colisión cuando dos entradas distintas de la función resumen producen la misma salida. U puede tener infinitos elementos, pero M tiene un número finito de elementos debido a que el tamaño de sus cadenas es fijo. Por tanto, la posibilidad de existencia de colisiones es intrínseca a la definición de función hash.
- En una buena función resumen se desea que la probabilidad de colisión sea muy baja.

Propiedades

* Función resumen de un sólo sentido: dada una cadena x se quiere que sea muy fácil calcular $h(x)$, pero que dado un posible resumen y sea difícil calcular x de modo que $y = h(x)$. Además, debe ser difícil encontrar una pareja (x, y) con $x \neq y$ de modo que $h(x) = h(y)$.

Propiedades

* Función resumen de un sólo sentido: dada una cadena x se quiere que sea muy fácil calcular $h(x)$, pero que dado un posible resumen y sea difícil calcular x de modo que $y = h(x)$. Además, debe ser difícil encontrar una pareja (x, y) con $x \neq y$ de modo que $h(x) = h(y)$.

* Determinista: Un mensaje siempre tiene el mismo valor *hash* asociado.

Propiedades

* Función resumen de un sólo sentido: dada una cadena x se quiere que sea muy fácil calcular $h(x)$, pero que dado un posible resumen y sea difícil calcular x de modo que $y = h(x)$. Además, debe ser difícil encontrar una pareja (x, y) con $x \neq y$ de modo que $h(x) = h(y)$.

* Determinista: Un mensaje siempre tiene el mismo valor *hash* asociado.

* De bajo coste computacional.

Propiedades

* Función resumen de un sólo sentido: dada una cadena x se quiere que sea muy fácil calcular $h(x)$, pero que dado un posible resumen y sea difícil calcular x de modo que $y = h(x)$. Además, debe ser difícil encontrar una pareja (x, y) con $x \neq y$ de modo que $h(x) = h(y)$.

* Determinista: Un mensaje siempre tiene el mismo valor *hash* asociado.

* De bajo coste computacional.

* Uniforme: Todos los posibles resúmenes son igualmente probables.

Propiedades

* Función resumen de un sólo sentido: dada una cadena x se quiere que sea muy fácil calcular $h(x)$, pero que dado un posible resumen y sea difícil calcular x de modo que $y = h(x)$. Además, debe ser difícil encontrar una pareja (x, y) con $x \neq y$ de modo que $h(x) = h(y)$.

* Determinista: Un mensaje siempre tiene el mismo valor *hash* asociado.

* De bajo coste computacional.

* Uniforme: Todos los posibles resúmenes son igualmente probables.

* Con efecto avalancha: una modificación minúscula en la cadena de entrada ocasiona cambios en el valor hash comparables a un cambio de cualquier otro tipo.

Propiedades

* Función resumen de un sólo sentido: dada una cadena x se quiere que sea muy fácil calcular $h(x)$, pero que dado un posible resumen y sea difícil calcular x de modo que $y = h(x)$. Además, debe ser difícil encontrar una pareja (x, y) con $x \neq y$ de modo que $h(x) = h(y)$.

* Determinista: Un mensaje siempre tiene el mismo valor *hash* asociado.

* De bajo coste computacional.

* Uniforme: Todos los posibles resúmenes son igualmente probables.

* Con efecto avalancha: una modificación minúscula en la cadena de entrada ocasiona cambios en el valor hash comparables a un cambio de cualquier otro tipo.

- Las funciones hash permiten la autenticación de mensajes.

Algoritmo MD5

* El algoritmo MD5 (*Message-Digest Algorithm 5*), es un algoritmo de reducción criptográfica de 128 bits.

Algoritmo MD5

* El algoritmo MD5 (*Message-Digest Algorithm 5*), es un algoritmo de reducción criptográfica de 128 bits.

* Fue desarrollado por el profesor Ronald Rivest del MIT (*Massachusetts Institute of Technology*) en 1991.

Algoritmo MD5

* El algoritmo MD5 (*Message-Digest Algorithm 5*), es un algoritmo de reducción criptográfica de 128 bits.
* Fue desarrollado por el profesor Ronald Rivest del MIT (*Massachusetts Institute of Technology*) en 1991.
* La codificación del MD5 de 128 bits es representada típicamente como un número de 32 símbolos hexadecimales.

Algoritmo MD5

- El algoritmo MD5 (*Message-Digest Algorithm 5*), es un algoritmo de reducción criptográfica de 128 bits.
- Fue desarrollado por el profesor Ronald Rivest del MIT (*Massachusetts Institute of Technology*) en 1991.
- La codificación del MD5 de 128 bits es representada típicamente como un número de 32 símbolos hexadecimales.

$$MD5(\text{Jónatan}) = f08e4e01632a06331957d09b4db2753e$$

Algoritmo MD5

* El algoritmo MD5 (*Message-Digest Algorithm 5*), es un algoritmo de reducción criptográfica de 128 bits.
* Fue desarrollado por el profesor Ronald Rivest del MIT (*Massachusetts Institute of Technology*) en 1991.
* La codificación del MD5 de 128 bits es representada típicamente como un número de 32 símbolos hexadecimales.

$$MD5(\text{Jónatan}) = f08e4e01632a06331957d09b4db2753e$$

$$MD5(Jonatan) = 33838aa89eaf37488a31e65220c35388$$

Algoritmo MD5

* El algoritmo MD5 (*Message-Digest Algorithm 5*), es un algoritmo de reducción criptográfica de 128 bits.
* Fue desarrollado por el profesor Ronald Rivest del MIT (*Massachusetts Institute of Technology*) en 1991.
* La codificación del MD5 de 128 bits es representada típicamente como un número de 32 símbolos hexadecimales.

$$MD5(\text{Jónatan}) = f08e4e01632a06331957d09b4db2753e$$

$$MD5(Jonatan) = 33838aa89eaf37488a31e65220c35388$$

$$MD5(\textit{No conozco a la mitad de ustedes}) = cd3b52cd869ced5b1467efeac5afcb0e$$

Algoritmo MD5

* El algoritmo MD5 (*Message-Digest Algorithm 5*), es un algoritmo de reducción criptográfica de 128 bits.
* Fue desarrollado por el profesor Ronald Rivest del MIT (*Massachusetts Institute of Technology*) en 1991.
* La codificación del MD5 de 128 bits es representada típicamente como un número de 32 símbolos hexadecimales.

$$MD5(\text{Jónatan}) = f08e4e01632a06331957d09b4db2753e$$

$$MD5(Jonatan) = 33838aa89eaf37488a31e65220c35388$$

$$MD5(No\ conozco\ a\ la\ mitad\ de\ ustedes) = cd3b52cd869ced5b1467efeac5afcb0e$$

$$MD5(\ ) = d41d8cd98f00b204e9800998ecf8427e$$

Cómo preparamos el mensaje

- Queremos obtener bloques de 512 bits con los que trabajar.

Cómo preparamos el mensaje

- Queremos obtener bloques de 512 bits con los que trabajar.
- Para ello, primero transformamos nuestro mensaje a binario usando ASCII.

$$\text{Hola} = 01001000\ 01101111\ 01101100\ 01100001$$

Cómo preparamos el mensaje

- Queremos obtener bloques de 512 bits con los que trabajar.
- Para ello, primero transformamos nuestro mensaje a binario usando ASCII.

$$\text{Hola} = 01001000\ 01101111\ 01101100\ 01100001$$

- Luego incluimos un 1 y seguido de tantos ceros hasta llegar a 448 bits.

$$0100100001101111011011000110000\mathbf{1}000\ldots0$$

Cómo preparamos el mensaje

- Queremos obtener bloques de 512 bits con los que trabajar.
- Para ello, primero transformamos nuestro mensaje a binario usando ASCII.

$$\text{Hola} = 01001000\ 01101111\ 01101100\ 01100001$$

- Luego incluimos un 1 y seguido de tantos ceros hasta llegar a 448 bits.

$$01001000011011110110110001100001\mathbf{1}000\ldots0$$

- Finalmente utilizamos los 64 bits restantes (64=512-448) para poner la longitud del mensaje en *little endian*. En este caso tenemos 24 bits, que en binario se expresan como:

$$00011000$$

Cómo preparamos el mensaje

- Queremos obtener bloques de 512 bits con los que trabajar.
- Para ello, primero transformamos nuestro mensaje a binario usando ASCII.

$$\text{Hola} = 01001000\ 01101111\ 01101100\ 01100001$$

- Luego incluimos un 1 y seguido de tantos ceros hasta llegar a 448 bits.

$$01001000011011110110110001100001\mathbf{1}000\ldots0$$

- Finalmente utilizamos los 64 bits restantes (64=512-448) para poner la longitud del mensaje en *little endian*. En este caso tenemos 24 bits, que en binario se expresan como:

$$00011000$$

- Por lo que nos queda:

$$01001000011011110110110001100001\mathbf{1}000\ldots\mathbf{0}\ \mathbf{00011000}0\ldots0$$

un bloque de 512 bits.

Cómo preparamos el mensaje

- ¿Y si el mensaje pasado a bits tiene entre 448 y 512?

  Incluimos el 1 y rellenamos el bloque de 512 con ceros. Creamos un nuevo bloque de 512 con 448 ceros, y donde los últimos 64 bits son la longitud del mensaje.

Cómo preparamos el mensaje

- ¿Y si el mensaje pasado a bits tiene entre 448 y 512?

  Incluimos el 1 y rellenamos el bloque de 512 con ceros. Creamos un nuevo bloque de 512 con 448 ceros, y donde los últimos 64 bits son la longitud del mensaje.

- ¿Y si son más de 512?

  Dividimos el mensaje en bloques de 512, y rellenamos "donde falte".

Cómo preparamos el mensaje

- ¿Y si el mensaje pasado a bits tiene entre 448 y 512?

  Incluimos el 1 y rellenamos el bloque de 512 con ceros. Creamos un nuevo bloque de 512 con 448 ceros, y donde los últimos 64 bits son la longitud del mensaje.

- ¿Y si son más de 512?

  Dividimos el mensaje en bloques de 512, y rellenamos "donde falte".

- En definitiva, vamos a tener $BL^1, BL^2, \ldots, BL^m$ bloques de 512 bits.

Valores iniciales y bucles del algoritmo

- El algoritmo actúa reiteradamente sobre cada bloque de 512 bits o 16 palabras[1], utilizando como valores iniciales cuatro palabras. Al inicio estas cuatro palabras tienen los siguientes valores hexadecimales:

$$A_0 = 67452301 \qquad B_0 = EFCDAB89$$

$$C_0 = 98BADCFE \qquad D_0 = 10325476$$

[1] 1 Palabra = 4 bytes= 32 bits

Valores iniciales y bucles del algoritmo

- El algoritmo actúa reiteradamente sobre cada bloque de 512 bits o 16 palabras[1], utilizando como valores iniciales cuatro palabras. Al inicio estas cuatro palabras tienen los siguientes valores hexadecimales:

$$A_0 = 67452301 \qquad B_0 = EFCDAB89$$

$$C_0 = 98BADCFE \qquad D_0 = 10325476$$

- Se recorren las 16 palabras de cada bloque mediante cuatro rondas.

[1] 1 Palabra = 4 bytes= 32 bits

Valores iniciales y bucles del algoritmo

- El algoritmo actúa reiteradamente sobre cada bloque de 512 bits o 16 palabras[1], utilizando como valores iniciales cuatro palabras. Al inicio estas cuatro palabras tienen los siguientes valores hexadecimales:

$$A_0 = 67452301 \qquad B_0 = EFCDAB89$$

$$C_0 = 98BADCFE \qquad D_0 = 10325476$$

- Se recorren las 16 palabras de cada bloque mediante cuatro rondas.

- Al final de estas cuatro rondas se actualizan las palabras "A, B, C y D".

[1] 1 Palabra = 4 bytes= 32 bits

Valores iniciales y bucles del algoritmo

- El algoritmo actúa reiteradamente sobre cada bloque de 512 bits o 16 palabras[1], utilizando como valores iniciales cuatro palabras. Al inicio estas cuatro palabras tienen los siguientes valores hexadecimales:

$$A_0 = 67452301 \qquad B_0 = EFCDAB89$$

$$C_0 = 98BADCFE \qquad D_0 = 10325476$$

- Se recorren las 16 palabras de cada bloque mediante cuatro rondas.
- Al final de estas cuatro rondas se actualizan las palabras "A, B, C y D".
- Se pasa al siguiente bloque, y así hasta finalizar.

[1] 1 Palabra = 4 bytes= 32 bits

Elementos de los bucles: Funciones

- En cada ronda actua una función distinta. Las cuatro funciones que vamos a necesitar son:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

donde $\wedge = AND$, $\vee = OR$, $\neg = NOT$, $\oplus = SUMA$.

Elementos de los bucles: Funciones

- En cada ronda actua una función distinta. Las cuatro funciones que vamos a necesitar son:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

  donde $\wedge = AND$, $\vee = OR$, $\neg = NOT$, $\oplus = SUMA$.

- Una forma más conveniente de escribirlo:

$$\mathcal{F}(X,Y,Z,i) = \begin{cases} F(X,Y,Z), & i = 1,\ldots 16. \\ G(X,Y,Z), & i = 17,\ldots,32. \\ H(X,Y,Z), & i = 33,\ldots,48. \\ I(X,Y,Z), & i = 49,\ldots,64. \end{cases}$$

Elementos de los bucles: Rotación de bits

- Transformación expresada de la forma $<<< n$ e indica una reordenación de bits en una palabra.

Elementos de los bucles: Rotación de bits

- Transformación expresada de la forma $<<< n$ e indica una reordenación de bits en una palabra.
- La transformación coge los n-primeros bits y los mueve al final de la palabra.

Elementos de los bucles: Rotación de bits

- Transformación expresada de la forma $<<< n$ e indica una reordenación de bits en una palabra.
- La transformación coge los n-primeros bits y los mueve al final de la palabra.
- Veámoslo con un ejemplo:

$$10010001001010101010111100011011 <<< 5$$
$$10010001001010101010111100011011$$
$$00100101010101011110001101110010$$

Elementos de los bucles: Rotación de bits

- Transformación expresada de la forma $<<< n$ e indica una reordenación de bits en una palabra.
- La transformación coge los n-primeros bits y los mueve al final de la palabra.
- Veámoslo con un ejemplo:

$$10010001001010101010111100011011 <<< 5$$
$$10010001001010101010111100011011$$
$$00100101010101011110001101110010$$

- Asociada a la rotación necesitaremos la siguiente función:

$$Rot(i) = \begin{cases} (7 \;\; 12 \;\; 17 \;\; 22) & i = 1, \ldots, 16 \\ (5 \;\; 9 \;\; 14 \;\; 20) & i = 17, \ldots, 32 \\ (4 \;\; 11 \;\; 16 \;\; 23) & i = 33, \ldots, 48 \\ (6 \;\; 10 \;\; 15 \;\; 21) & i = 49, \ldots, 64 \end{cases}$$

Elementos de los bucles: Dos funciones adicionales

- Necesitaremos también dos funciones adicionales. Una primera que nos dará constantes:

$$T(i) = E(|\sin(i)2^{32}|), \quad i = 1, \ldots, 64$$

Elementos de los bucles: Dos funciones adicionales

- Necesitaremos también dos funciones adicionales. Una primera que nos dará constantes:

$$T(i) = E(|\sin(i)2^{32}|), \quad i = 1, \ldots, 64$$

- Y otra que nos dará una lista de "posiciones":

$$\mathrm{Sel}(i) = \begin{cases} (i-1) \pmod{16} & i = 1, \ldots, 16. \\ 5(i-1)+1 \pmod{16} & i = 1, \ldots, 16. \\ 3(i-1)+5 \pmod{16} & i = 1, \ldots, 16. \\ 7(i-1) \pmod{16} & i = 1, \ldots, 16. \end{cases}$$

¿Cómo funciona el algoritmo?

- Vamos a hacer 64 iteraciones por cada bloque M de 512 bits.

$$M = M_1 \quad M_2 \quad \dots \quad M_{16}$$

¿Cómo funciona el algoritmo?

- Vamos a hacer 64 iteraciones por cada bloque M de 512 bits.

$$M = M_1 \quad M_2 \quad \dots \quad M_{16}$$

- Todas las iteraciones siguen el mismo patrón:
$(A_i \quad B_i \quad C_i \quad D_i) \rightarrow (A_{i+1} \quad B_{i+1} \quad C_{i+1} \quad D_{i+1})$
donde $C_{i+1} = B_i, D_{i+1} = C_i$ y $A_{i+1} = D_i$.

¿Cómo funciona el algoritmo?

- Vamos a hacer 64 iteraciones por cada bloque M de 512 bits.

$$M = M_1 \quad M_2 \quad \dots \quad M_{16}$$

- Todas las iteraciones siguen el mismo patrón:
$(A_i \quad B_i \quad C_i \quad D_i) \rightarrow (A_{i+1} \quad B_{i+1} \quad C_{i+1} \quad D_{i+1})$
donde $C_{i+1} = B_i, D_{i+1} = C_i$ y $A_{i+1} = D_i$.

- Por su lado, $B_{i+1} = \mathrm{Transf}(A_i, B_i, C_i, D_i, i)$, donde:

$$Transf(X, Y, Z, W, i) = X + \mathcal{F}(Y, Z, W, i) + T(i) + M_{Sel(i)+1} <<< Rot(i).$$

¿Cómo funciona el algoritmo?

- Vamos a hacer 64 iteraciones por cada bloque M de 512 bits.

$$M = M_1 \quad M_2 \quad \dots \quad M_{16}$$

- Todas las iteraciones siguen el mismo patrón:
$(A_i \quad B_i \quad C_i \quad D_i) \rightarrow (A_{i+1} \quad B_{i+1} \quad C_{i+1} \quad D_{i+1})$
donde $C_{i+1} = B_i, D_{i+1} = C_i$ y $A_{i+1} = D_i$.

- Por su lado, $B_{i+1} = \mathrm{Transf}(A_i, B_i, C_i, D_i, i)$, donde:

$$Transf(X, Y, Z, W, i) = X + \mathcal{F}(Y, Z, W, i) + T(i) + M_{Sel(i)+1} <<< Rot(i).$$

- El resultado obtenido por el primer bloque es $(A_{64} \quad B_{64} \quad C_{64} \quad D_{64})$.

¿Cómo funciona el algoritmo?