TEMA
Funciones HASH.

Las funciones criptográficas hash juegan un papel fundamental en la criptografía moderna. Hay muchas funciones hash, comúnmente usadas en aplicaciones no criptográficas. Nosotros nos referiremos siempre a las que admiten el adjetivo de criptográficas, y las llamaremos habitualmente simplemente funciones hash.

Una función hash, o función resumen, toma un mensaje como entrada y produce una salida que llamamos código hash, o resultado hash, o valor hash, o simplemente hash. La idea básica de las funciones criptográficas hash es que los valores hash obtenidos con ellas sirven como una imagen representativa y compactada de una cadena de entrada, y pueden usarse como un posible identificador único de esa cadena de entrada: ese valor hash obtenido del mensaje de entrada suele llamarse resumen del mensaje o huella digital del mensaje.

Las funciones hash se emplean en criptografía junto con los criptosistemas de firma digital para otorgar integridad a los datos. A la hora de firmar digitalmente un documento o mensaje, es práctica habitual hacer la firma sobre la huella digital del mensaje y no sobre la totalidad del mensaje a firmar.

1. FUNCIONES HASH DE USO CRIPTOGRÁFICO.

Entendemos por una FUNCIÓN HASH una aplicación $h: \Sigma^* \to \Sigma^n$, con $n \in \mathbb{N}$. Son funciones que transforman una cadena de longitud arbitraria en una cadena de longitud fija. También se les llama en castellano función RESUMEN.

Un ejemplo muy sencillo de estas funciones podría ser aquella que transforma una cadena de bits $b_1 b_2 \dots b_k \in \{0,1\}^*$ en un único dígito binario $b_1 \oplus b_2 \oplus \dots \oplus b_k$: es la función de paridad.

Las funciones Hash pueden generarse mediante funciones de compresión. Una FUNCIÓN DE COMPRESIÓN es una transformación $h: \Sigma^m \to \Sigma^n$, $n, m \in \mathbb{N}$, $m > n$. Es decir, es una aplicación que transforma cadenas de longitud fija en cadenas de longitud fija menor.

Sea la función hash $h: \Sigma^* \to \Sigma^n$, o la función de compresión $h: \Sigma^m \to \Sigma^n$. En adelante llamaremos al conjunto de argumentos de h con el nombre de D. Si h es una función de hash entonces $D = \Sigma^*$; si h es una función de compresión, entonces $D = \Sigma^m$.

Aquellas funciones h que verifiquen que el cálculo de su inversa es computacionalmente muy complicado, las llamamos FUNCIONES DE UNA VÍA. Es decir, dado $s \in \Sigma^n$ es computacionalmente inasequible hallar $x \in D$ tal que $h(x) = s$. Realmente no queda perfilado o cuantificado el concepto "inasequible"; entendemos, de forma intuitiva, que serán funciones donde el tiempo necesario para encontrar el inverso es tal que no lo hace viable. No se sabe a ciencia cierta si existen realmente funciones h que no sean invertibles. Se considera que una función h es de una sola vía cuando la dificultad computacional de su inversa sea alta.

Toda función de compresión o hash tendrá colisiones, porque son funciones no inyectivas. Entendemos por COLISIÓN la existencia de pares $(x, y) \in D^2$, tales que $x \neq y$ y $h(x) = h(y)$. Una función h diremos que es RESISTENTE A COLISIONES si conocido $h(x)$ para un $x \in D$, es computacionalmente inviable hallar $x' \in D$ con $h(x') = h(x)$.

Una función hash diremos que es FUERTEMENTE RESISTENTE A COLISIONES si resulta computacionalmente inviable encontrar cualquier par $(x, y) \in D^2$, $x \neq y$ tales que $h(x) = h(y)$.

Una vez presentamos estos conceptos previos, podemos determinar ahora cuando una función hash $h(x)$ o una función de compresión serán válidas para usos criptográficos. Eso ocurrirá si la función verifica las siguientes propiedades:

1. Fácil de computar para todo $x \in D$.
2. De una sola vía.
3. Resistente a colisiones.
4. Fuertemente resistente a colisiones.

En el siguiente epígrafe presentamos un ataque clásico contra las funciones hash, eficaz si ésta no es fuertemente resistente a colisiones. La posibilidad de

este ataque nos permitirá determinar una característica elemental que deben verificar las funciones criptográficas hash: $h: D \to \Sigma^n$ no será fuertemente resistente a colisiones si n no es suficientemente grande. A lo largo de siguiente epígrafe se verá qué significa aquí el adjetivo "suficientemente".

2. ATAQUE DEL CUMPLEAÑOS.

Presentamos un ataque bastante simple contra las funciones hash, llamado ATAQUE DEL CUMPLEAÑOS. Este ataque surte efecto si la función hash no es fuertemente resistente a las colisiones.

El ataque del cumpleaños se basa en la PARADOJA DEL CUMPLEAÑOS, que queda presentada como apéndice a este tema.

Conocida ya la paradoja, veamos ahora en qué consiste la estrategia de ataque a una función hash mediante la técnica del ataque del cumpleaños. Esta estrategia es eficaz cuando la longitud del valor hash no es suficientemente grande. Suponemos que esa longitud es n.

Las circunstancias que concurren en este ataque son las siguientes: Alguien (A) desea obtener una huella digital (h_m) de un determinado mensaje suyo (m). Si la función hash que emplea A no es fuertemente resistente a colisiones, entonces un atacante (B) puede aprovechar la ocasión y sustituir el mensaje de A por otro de igual significado (m') y de aspecto similar, con una huella ($h_{m'}$) igual a otro mensaje (f) fraudulento escrito por B, con un contenido diferente al mensaje inicial al mensaje de A (es decir, $h_{m'} = h_f$). Sólo queda entonces a B sustituir el mensaje m que A deseaba enviar por el mensaje sustitutivo m': A firmará "su" mensaje y guardará a buen recaudo su garantía de integridad (la huella digital) o lo enviará con la huella a un receptor. B puede ahora sustituir el mensaje m' por el suto fraudulento f: al tener ambos la misma huella, B habrá logrado burlar la garantía de integridad de datos que en principio ofrece la huella hash: ni A, ni un posible receptor, podrán verificar que el mensaje inicial ha sido modificado.

Veamos el procedimiento que sigue B para lograr sus fines:

1. A posee su mensaje m, del que desea calcular la huella digital o resumen.
2. El atacante B, que dispone de ese mensaje, genera $2^{n/2}$ variaciones de él, todos ellos sustancialmente con el mismo significado. (n es la longitud del

valor hash.) entre estos mensajes está el candidato a ser el mensaje sustituto, m'.

3. El atacante toma el mensaje fraudulento y prepara también $2^{n/2}$ mensajes parecidos, de similar significado al mensaje fraudulento inicial. Entre estos mensajes está el candidato a ser el mensaje fraudulento f.
4. El atacante compara los dos conjuntos de mensajes y busca un par (uno entre las copias del mensaje de A, y otro entre los mensajes fraudulentos) que produzcan la misma salida hash. La probabilidad de que esto ocurra, con $2^{n/2}$ mensajes generados es, por la paradoja del cumpleaños, mayor de 0.5. Si no consigue encontrar dos hash iguales, el atacante puede generar unos mensajes más en cada conjunto hasta lograrlo.
5. Sólo le resta al atacante sustituir el mensaje inicial m por su versión de igual significado m'.

Para una función hash que llegue a resúmenes o valores hash de 64 bits, el atacante puede sustituir un documento por otro con tan solo 2^{32} pruebas. Y la generación de un número grande de versiones similares del mismo texto no es tarea en absoluto difícil. Se requiere un software sencillo que lo realiza y cierta capacidad de memoria.

Para prever y prevenir un ataque de este estilo será necesario utilizar funciones hash que realicen un resumen de cierta longitud. Normalmente $n \geq 128$ es suficiente. Habitualmente sin embargo se emplean tamaños $n \geq 160$. Es el modo habitual de lograr funciones hash fuertemente resistentes a colisiones.

3. FUNCIONES DE COMPRESIÓN DESDE FUNCIONES DE CIFRADO.

No podemos afirmar con rotundidad que existan funciones hash resistentes a colisiones. Tampoco podemos asegurar que un determinado criptosistema es seguro y eficiente: siempre puede haber un modo de ataque desconocido.

Podemos suponer que si definimos una función hash como resultado de aplicar una función criptográfica, el resultado será resistente a colisiones. Tan resistente a colisiones como segura sea la función de cifrado del criptosistema. Digo que podemos suponer esta afirmación, pero de hecho hoy por hoy no la podemos demostrar. Sin embargo se utilizan a veces funciones criptográficas para hacer hash.

Supongamos que tanto el espacio de claves, como el espacio de texto plano como el de texto cifrado es $\{0,1\}^n$. Las funciones de cifrado son $e_k: \{0,1\}^n \longrightarrow \{0,1\}^n$, con $k \in \{0,1\}^n$. Nuestro valor hash tendrá longitud n. Para protegerse del ataque del cumpleaños, tomamos criptosistemas de cifrado con $n \geq 128$: DES queda descartado.

La función hash $h: \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}^n$ puede definirse de las siguientes formas:

$h(k,x) = e_k(x) \oplus x$.

$h(k,x) = e_k(x) \oplus x \oplus k$.

$h(k,x) = e_k(x \oplus k) \oplus x$.

$h(k,x) = e_k(x \oplus k) \oplus x \oplus k$.

4. FUNCIONES HASH DESDE FUNCIONES DE COMPRESIÓN.

La resistencia a colisión de las funciones de compresión pueden usarse para la construcción de funciones hash, resistentes también a colisiones. Se puede demostrar que si la función hash resultante no es resistente a colisiones, entonces tampoco es resistente a colisiones la función de compresión original. Lamentablemente, no sabemos demostrar la afirmación inversa.

Veamos un modo de lograr una función hash a partir de una función de comprensión.

Sea $g: \{0,1\}^m \longrightarrow \{0,1\}^n$, una función de compresión. Y sea $r = m - n$.

Como g es una función de compresión, tendremos que $r > 0$. Una elección habitual es $n = 128$ y $r = 512$.

A partir de g queremos construir una función hash $h: \{0,1\}^* \longrightarrow \{0,1\}^n$. Sea $x \in \{0,1\}^*$. Consideramos el caso $r > 1$. Realizamos los siguientes pasos:

1. Añadimos a x el número de ceros a la izquierda necesarios para que su longitud sea un entero divisible por r.
2. A esa cadena le añadimos r ceros por la derecha.
3. Volvemos a tomar la cadena original x. Expresamos su longitud inicial en código binario. Añadimos a esa expresión tantos ceros como sea necesario para que su longitud sea divisible por $r - 1$. Delante de esa nueva cadena

añadimos un 1, y lo mismo hacemos delante de cada porción de $r-1$ caracteres.

4. Añadimos esta segunda cadena resultante a la anterior creada. Tendremos la secuencia de palabras de longitud r: $x = x_1 x_2 \dots x_t$, donde $x_i \in \{0,1\}^r$, $1 \leq i \leq t$

Por ejemplo, si tenemos $r = 4$ y $x = 111011$, los pasos indicados son:

1. $x' \leftarrow 0011\ 1011$
2. $x' \leftarrow 0011\ 1011\ 0000$
3. La longitud inicial de x es $l(x) = 6$, que expresada en binario es $l(x) = 110$, que ya tiene un número de dígitos múltiplo de $r-1$, es decir, de 3. Añadimos a esta cadena un 1 delante y llegamos así a la cadena 1110.
4. Concatenamos: $x' \leftarrow 0011\ 1011\ 0000\ 1110$.

El valor hash $h(x)$ se calcula de forma iterativa. Inicialmente $H_0 = 0^n$: una cadena de n ceros. Las siguientes iteraciones son: $H_i = g(H_{i-1} \circ x_i)$, para $1 \leq i \leq t$. La última iteración la tomamos como resultado final: $h(x) = H_t$. La operación composición aquí indicada ($\circ$) es la concatenación de cadenas: Cada H_i es, por ser imagen de g, una cadena binaria de longitud n. H_0 también es, por definición una cadena binaria de esa longitud. Cada x_i es, por la propia construcción que hemos tomado, de longitud r. Cada concatenación $H_{i-1} \circ x_i$ es, por tanto, de longitud $n + r = m$, que es la longitud del conjunto inicial de la función g.

De todo lo dicho en los dos epígrafes anteriores, se comprende que, a partir de cualquier criptosistema de cifrado por bloques se puede definir una función de compresión, a partir de la cual a su vez se puede definir una función hash.

5. SHA – 1.

La familia **SHA** (*Secure Hash Algorithm*) es un sistema de funciones *hash* criptográficas relacionadas de la Agencia de Seguridad Nacional de los Estados Unidos y publicadas por el *National Institute of Standards and Technology* (NIST). El primer miembro de la familia fue publicado en 1993 es oficialmente llamado **SHA**. Sin embargo, hoy día, no oficialmente se le llama **SHA-0** para evitar confusiones con sus sucesores. Dos años más tarde el primer sucesor de SHA fue publicado con el nombre de **SHA-1**. Existen cuatro variantes más que se han publicado desde entonces cuyas diferencias se basan en un diseño algo

modificado y rangos de salida incrementados: **SHA-224**, **SHA-256**, **SHA-384**, y **SHA-512** (llamándose **SHA-2** a todos ellos).

SHA-1 ha sido examinado muy de cerca por la comunidad criptográfica pública, y no se ha encontrado ningún ataque efectivo. No obstante, en el año 2004, un número de ataques significativos fueron divulgados sobre funciones criptográficas de *hash* con una estructura similar a SHA-1; lo que ha planteado dudas sobre la seguridad a largo plazo de SHA-1.

SHA-0 y SHA-1 producen una salida resumen de 160 bits de un mensaje que puede tener un tamaño máximo de 2^{64} bits, y se basa en principios similares a los usados por el profesor Ronald L. Rivest del MIT en el diseño de los algoritmos de resumen de mensaje MD4 y MD5.

DESCRIPCIÓN DEL ALGORITMO. PREPARACIÓN DE LA CADENA DE ENTRADA.

Tomamos $x \in \{0,1\}^*$. Como ya ha quedado dicho, $|x| < 2^{64}$. El valor hash se compone mediante los siguientes pasos:

1. Añadimos un dígito 1 al final de la cadena de bits de x: $x \leftarrow x \circ 1$.
2. Añadimos la mínima cantidad de ceros a la derecha necesarios para que $|x| = k \cdot 512 - 64$.
3. Escribimos la longitud de x como un número de 64 bits.

Veamos un ejemplo:

0. `x  <-- 0110 0001  0110 0010  0110 0011  0110 0100  0100 0101.`
1. `x' <-- 0110 0001  0110 0010  0110 0011  0110 0100  0100 0101  1.`
2. Longitud(x') = 41: hay que añadir 407 ceros a la derecha: 512 – 64 = 448; tenemos ya 41, faltan 407.

   En hexadecimal, tenemos:

   ```
   x <-- 61626364 65800000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000
   ```

3. Longitud de x: 40; en hexadecimal, con 64 bits: `00000000 00000028`.
4. Valor final de x:

   ```
   x <-- 61626364 65800000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
   ```

00000000 00000000 00000000 00000028.

DESCRIPCIÓN DEL ALGORITMO. PROCESAMIENTO DE LA CADENA PREPARADA.

Para la computación del valor hash empleamos las funciones:

$f_t: \{0,1\}^{32} \times \{0,1\}^{32} \times \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$.

Se define como sigue:

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & para\ 0 \leq t \leq 19 \\ B \oplus C \oplus D & para\ 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & para\ 40 \leq t \leq 59 \\ B \oplus C \oplus D & para\ 60 \leq t \leq 79 \end{cases}$$

Donde $\wedge$ es la operación lógica a nivel de bit AND; $\vee$ es la operación a nivel de bit OR; $\oplus$ es la operación lógica a nivel de bit XOR.

Por otro lado, se emplean las siguientes constantes:

$$K_t = \begin{cases} 5A827999 & para\ 0 \leq t \leq 19 \\ 6ED9EBA1 & para\ 20 \leq t \leq 39 \\ 8F1BBCDC & para\ 40 \leq t \leq 59 \\ CA62C1D6 & para\ 60 \leq t \leq 79 \end{cases}$$

Queda ver ahora cómo se calcula el Hash. Tenemos el valor de x ya transformado según las reglas antes indicadas. Su longitud es divisible por 512. Tomamos $\boldsymbol{x = M_1 M_2 M_3 \dots M_n}$, secuencias de palabras de 512 bits.

Comenzamos inicializando valores:

$H_0 = 67452301$, $H_1 = EFCDAB89$, $H_2 = 98BADCFE$, $H_3 = 10325476$, $H_4 = C3D2E1F0$.

El siguiente procedimiento se repite para $i = 1, 2, \dots, n$ (es decir, para cada bloque M_i de 512 bits). Definimos la operación $S^k(w)$ como el desplazamiento circular a izquierda de k bits, sobre palabras de 32 bits. Definimos también la suma (+) para enteros que trabajan sobre 32 bits, es decir, suma mod 2^{32}.

Los pasos que se realizan para cada M_i son:

1. Escribir M_i como una secuencia $\boldsymbol{M_i = W_0 W_1 \dots W_{15}}$, cada uno de 32 bits.
2. Para $t = 16, 17, \dots, 79$, calcular $\boldsymbol{W_t = S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})}$.
3. Tomar $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$.
4. Para $t = 0, 1, \dots, 79$, calcular $T = S^5(A) + f_t(B,C,D) + E + W_t + K_t$, $E = D$, $D = C$, $C = S^{30}(B)$, $B = A$, $A = T$. (T es una variable auxiliar.)
5. Calcular $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$.

Al final de procesar todos los bloques M_i de 512 bits, tendremos que el valor hash está formado por la cadena SHA-1(x)= $H_0H_1H_2H_3H_4$. (160 bits)

La codificación hash vacía para SHA-1 corresponde a:

SHA1("") = DA39 A3EE 5E6B 4B0D 3255 BFEF 9560 1890 AFD8 0709

6. APÉNDICE AL TEMA. PARADOJA DEL CUMPLEAÑOS.

Supongamos un grupo de personas en una habitación. ¿Cuál es la probabilidad de que dos de ellos celebren el cumpleaños el mismo día? Otra forma de presentar esta paradoja es: ¿Cuál es el número mínimo de personas k que debe haber en la habitación para que la probabilidad de que en el grupo al menos dos celebren el mismo día sea mayor que 0.5? Para buscar la respuesta, ignoramos la fecha 29 de febrero de los años bisiestos: tomamos, pues, $n = 365$. La pregunta es, cuál es el mínimo valor de k para el cual se verifica que$P(365, k) \geq 0.5$.

De entrada es evidente que $k \leq 365$. Superado ese valor límite la probabilidad de hallar dos fechas duplicadas es uno, porque hay más personas que fechas posibles.

Para lograr tener personas todas ellas con fechas de cumpleaños diferentes, primero dejamos entrar a una en la habitación, sea cual sea su fecha de cumpleaños; luego dejamos entrar a una segunda, limitando sus fechas de cumpleaños a 364; a la siguiente le limitamos su cumpleaños a 363 días: si alguna de las sucesivas personas que quieren entrar en la habitación no verifica esa exigencia, sencillamente no la dejamos entrar, y pasamos a la siguiente. Así hasta llegar a tener dentro de la habitación a las k personas.

Nos preguntamos cuántas combinaciones de fechas de cumpleaños posibles tendremos en nuestra habitación. El primer hombre tendrá una cualquiera de las 365, el segundo una cualquiera de las 364 restantes, el tercero una cualquiera de las 363 restantes, y así hasta el hombre $k - ésimo$ que podrá tener cualquiera de las $(365 - k + 1)$ fechas restantes. Es decir, las diferentes combinaciones posibles son $N = 365 \times 354 \times \ldots \times (353 - k + 1) = 365!/(365 - k)!$

Si eliminámaros la restricción de que no haya fechas de cumpleaños duplicadas, entonces cada individuo podría tener su fecha de nacimiento entre cualquiera de

los 365 días del año, y entonces el número de posibilidades en las fechas de los personales de la habitación sería una cualquiera de las combinaciones de 365 elementos tomados en grupos de k, es decir, 365^k combinaciones diferentes.

Así las cosas, la probabilidad de que no hay duplicado alguno ($Q(365,k)$) es:

$$Q(365,k) = \frac{365!/(365-k)!}{365^k}$$

Y por tanto

$$P(365,k) = 1 - Q(365,k) = 1 - \frac{365!}{(365-k)! \cdot 365^k}$$

Esta expresión da como resultado que para $k = 23$ entonces $P(365,23) = 0.507$. Para $k = 100$ la probabilidad de que haya dos personas con la misma fecha de cumpleaños es de $P(365,100) = 0.9999997$.

La paradoja reside en que no es lo mismo preguntar por la probabilidad de que alguien celebre el mismo día que yo, que preguntar la probabilidad de que dos cualesquiera de los habitantes de la habitación celebren en el mismo día.