Funciones resúmenes o hash.

INTRODUCCIÓN

La criptografía es una rama inicial de las Matemáticas y en la actualidad también de la Informática y la Telemática, que hace uso de métodos y técnicas con el objetivo principal de cifrar, y por tanto proteger, un mensaje o archivo por medio de un algoritmo, usando una o más claves. Esto da lugar a diferentes tipos de sistemas de cifra, denominados criptosistemas, que permiten asegurar, al menos, tres de los cuatro aspectos básicos de la seguridad informática: la confidencialidad o secreto del mensaje, la integridad del mensaje y autenticidad del emisor, así como el no repudio mutuo entre emisor y receptor.

Una de las aplicaciones más interesantes de la criptografía es la posibilidad real de añadir en un mensaje una firma digital: la autenticación completa. Dado que los sistemas de clave pública son muy lentos, en vez de firmar

Mensaje = M Función Resumen = h(M)

Firma (digital): = $E_{dE}\{h(M)\}$ dE: clave privada del emisor

digitalmente el mensaje completo, se hará una operación de cifra con la clave privada del emisor sobre un resumen o hash de dicho mensaje, representado por sólo una centena de bits.

¿Pero qué es una función hash?

Una función hash es una función que transforma una cadena binaria o mensaje de longitud arbitraria (por ejemplo x) en otra cadena binaria (por ejemplo Z) de longitud constante, generalmente más pequeña, que es la salida de la función, llamada un valor hash y son empleados para la emisión de certificados, firmas digitales, generación de claves y verificación de password. En la figura 1 se grafica la función hash. [3]

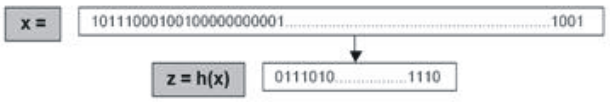

Fig. 1. Representación gráfica de la función hash [3]

Este resumen o función hash debe ofrecer ciertos niveles de seguridad para garantizar la integridad del mensaje y pueda ser empleado en criptografía, por lo que debe cumplir con las siguientes propiedades: [4]

1. Unidireccionalidad: conocido un resumen h(M), debe ser computacionalmente imposible encontrar M a partir de dicho resumen. (es decir, que la función h(M) no tenga función inversa o lo que es lo mismo no sea una función uno a uno)

2. Compresión: a partir de un mensaje de cualquier longitud, el resumen h(M) debe tener una longitud fija. Lo normal es que la longitud de h(M) sea menor que el mensaje M.

3. Facilidad de cálculo: sea computacionalmente fácil calcular h(M).

4. Difusión: el resumen h(M) debe ser una función compleja de todos los bits del mensaje M: si se modifica un sólo bit del mensaje M, el hash h(M) debería cambiar la mitad de sus bits aproximadamente. Por ejemplo:

hash ("Esto sí es una prueba de MD5")= e99008846853ff3b725c27315e469fbc

Cambio simple en el mensaje anterior:

hash ("Esto no es una prueba de MD5")= dd21d99a468f3bb52a136ef5beef5034

5. Colisión simple: será computacionalmente imposible conocido M, encontrar otro M' tal que h (M) = h (M'). Esto se conoce como resistencia débil a las colisiones.

6. Colisión fuerte: será computacionalmente difícil encontrar un par (M, M') de forma que h(M) = h(M'). Esto se conoce como resistencia fuerte a las colisiones.

La función hash emplea algoritmos específicos, entre los más conocidos están el MD5, el SHA-1, RIPEMD-160: [4]

- MD5: desarrollado por Ron Rivest en 1992. Es una mejora al MD4 y MD2 (1990), es más lento pero con mayor nivel de seguridad. Resumen de 128 bits. Aunque esta obsoleto desde 2005 su algoritmo constituye la base de otras funciones.

- SHA-1: Desarrollado por el NIST, National Institute of Standards and Technology, 1995. Similar a MD5 pero con resumen de 160 bits. Existen otras propuestas conocidas como SHA-256 y SHA-512, posibles estándares.

- RIPEMD-160: Comunidad Europea, RACE, 1992. Resumen de 160 bits.

Aunque existen otros como:

- N-Hash: Nippon Telephone and Telegraph, 1990. Resumen: 128 bits.

- Snefru: Ralph Merkle, 1990. Resúmenes entre 128 y 256 bits. Ha sido criptoanalizado y es lento.

- Tiger: Ross Anderson, Eli Biham, 1996. Resúmenes de hasta 192 bits. Optimizado para máquinas de 64 bits (Alpha).

- Panama: John Daemen, Craig Clapp, 1998. Resúmenes de 256 bits de longitud. Trabaja en modo función hash o como cifrador de flujo.

- Haval: Yuliang Zheng, Josef Pieprzyk y Jennifer Seberry, 1992. Admite 15 configuraciones diferentes. Hasta 256 bits.

A continuación se abordarán en detalles los algoritmos básicos de las funciones hash más usadas.

MD5 (Message-Digest Algorithm 5, Algoritmo de Resumen del Mensaje 5)

MD5 trabaja con mensajes de longitud variable para obtener un resumen de 128 bits. El mensaje es dividido en bloques de 512 bits los cuales son seccionados en 16 subloques de 32 bits, es decir, cada bloque de 512 bits queda formado por 16 palabras de 32 bits cada una. Para el procesamiento de dichos bloques (512 bits) se definen cuatro funciones y se emplea cuatro vectores ABCD de 32 bits cada uno. Para cada bloque se realizan cuatro vueltas en cada una de ellas se realiza una función diferente y se ejecutan 16 operaciones. Cada vuelta entrega una salida de 128 bits que forman el nuevo vector A,B,C,D que se utilizará para analizar el próximo bloque, realizando las mismas operaciones hasta procesar el mensaje completo.

Los valores resultantes del vector ABCD, después de culminar todas las operaciones con el mensaje completo, constituye el resumen del mismo o código hash de 128 bits representado típicamente por un número de 32 dígitos hexadecimal. [4]

MD5 ("Esto sí es una prueba de MD5")= e99008846853ff3b725c27315e469fbc

Para calcular el resumen del mensaje se realizan los siguientes pasos: [4] [5]

Paso 1. Adición de bits

Extender el mensaje hasta que su longitud en bits sea congruente con 448, módulo 512. La extensión se realiza como sigue: un sólo bit "1" se añade al mensaje, y después se añaden bits "0" hasta que la longitud en bits del mensaje extendido se haga congruente con 448, módulo 512. En todos los mensajes se añade al menos un bit y como máximo 512.

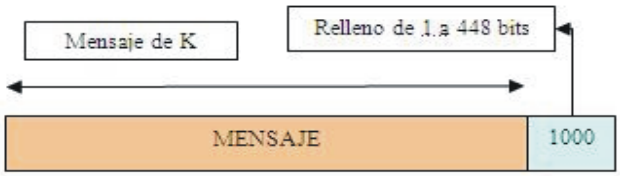

Fig. 2. Adición de bits en MD5 [4]

Paso 2. Longitud del mensaje

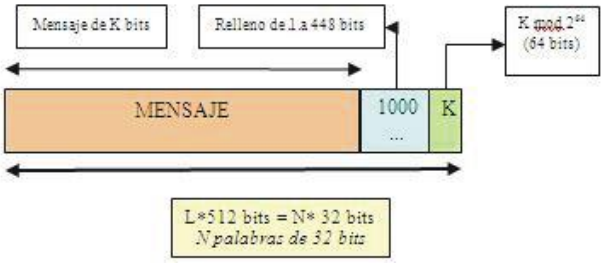

Fig 3. Conformación de la longitud del mensaje [4]

Como se muestra en la figura 2, al resultado del paso anterior, se añade un entero de 64 bits que representa la longitud 'k' del mensaje (longitud antes de añadir los bits). Si la longitud del mensaje original es mayor que 264, entonces sólo los 64 bits de menor peso de 'k' se usarán. El mensaje resultante (después de rellenar con los bits y con 'k') tiene una longitud que es un múltiplo exacto de 512 bits y es equivalentemente a un múltiplo de 16 palabras (32 bits por palabra), siendo M [0.....N-1] las palabras del mensaje resultante, donde N es múltiplo de 16.

Paso 3. Inicializar el búfer

Empleo de un vector inicial (A,B,C,D) de 32 bits cada uno (128 bits en total) para las operaciones con los bloques de mensajes y calcular el resumen del mensaje. Estos vectores se inicializan con los siguientes valores en hexadecimal (con los bits menos significativos primero):

palabra A16 = 01 23 45 67 palabra B16 = 89 ab cd ef palabra C16 = fe dc ba 98

palabra D16 = 76 54 32 10

Paso 4. Procesado del mensaje en bloques de 16 palabras (512 bits)

Se definen cuatro funciones (no lineales) auxiliares, F, G, H e I para las operaciones lógicas que se ejecutarán entre los bloques de información. Estas funciones toman como entrada tres palabras de 32 bits y su salida es una palabra de 32 bits. Las operaciones lógicas que ejecutarán F, G, H e I son las siguientes:

F (x, y, z) = (x AND y) OR (NOT x AND z) G (x, y, z) = (x AND z) OR (y AND NOT z)

H (x, y, z) = x XOR y XOR z I (x, y, z) = y XOR (x OR NOT z)

Siendo x,y,z = b,c,d

Estas funciones son empleadas para procesar cada bloque de mensaje de 512 bits. Para el procesamiento de dichos bloques se realizan cuatro vueltas, en cada una de ellas se realiza una función diferente no lineal en tres palabras de A, B, C y D, y se ejecutan 16 operaciones (debido a que cada bloque de 512 bits de mensaje es dividido en 16 subloques). Cada vuelta entrega una salida de 128 bits que forman el nuevo vector A, B, C y D que se utilizará para analizar el próximo bloque.

Las operaciones a realizar por cada función quedan definidas como sigue:

1ª Vuelta: FF(a,b,c,d,Mk,Ti,s) a = b + ((a + F(b,c,d) + Mk + Ti) <<< s)

2ª Vuelta: GG(a,b,c,d,Mk, Ti,s) a = b + ((a + G(b,c,d) + Mk + Ti) <<< s)

3ª Vuelta: HH(a,b,c,d,Mk, Ti,s) a = b + ((a + H(b,c,d) + Mk + Ti) <<< s)

4ª Vuelta: II(a,b,c,d,Mk, Ti,s) a = b + ((a + I(b,c,d) + Mk + Ti) <<< s)

Siendo k=0,1…,15; Mk los mensajes que conforman los subloques de mensajes; Ti i=1,2,…,64 los elementos de una tabla ya definida cuyos valores son construidos a partir de la función seno y <<< s representa los bits de desplazamiento a la izquierda.

La figura 4 muestra el esquema operacional en la primera vuelta para la función F, en las vueltas 2, 3 y 4 la secuencia es la misma para las funciones G, H, I respectivamente.[5]

El resumen final o hash del mensaje se obtiene de los últimos 128 bits, expresada en un número hexadecimal de 32 dígitos que se obtiene con una complejidad algorítmica de 264.

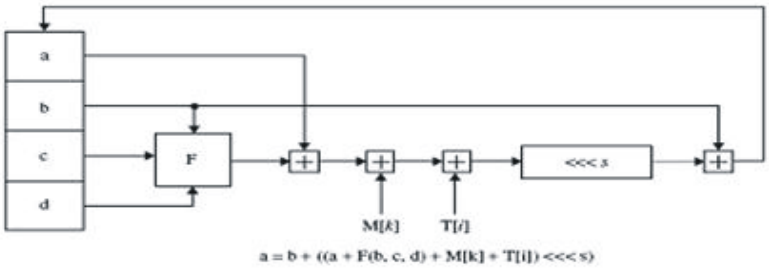

Fig. 4. Esquema operacional de la función MD5 para la 1ra vuelta [5]

Los resúmenes MD5 se utilizan extensamente en el mundo del software para proporcionar la seguridad de que un archivo descargado de Internet no ha sido alterado. Esto ofrece protección contra los 'Caballos de Troya' o 'Troyanos' y virus que pudieran estar incluidos en el archivo. La comprobación de un archivo descargado contra su suma MD5 no detecta solamente los archivos alterados de una manera maliciosa, también reconoce una descarga corrupta o incompleta. Para ello se puede emplear una herramienta MD5 por ejemplo MD5SUM disponible en los sistemas UNIX.

También es empleado para calcular el hash de las claves de los usuarios en los sistemas UNIX y GNU/Linux y poder autenticar al usuario. Otros usos de MD5 son en la comprobación de los correos electrónicos verificando que no han sido alterados usando claves públicas y privadas.

A pesar de haber sido considerado criptográficamente seguro en un principio, su uso actualmente es cuestionable ya que en el 2004 fueron reveladas sus vulnerabilidades o colisiones por Xiaoyun Wang, Dengguo Feng, Xuejia Lai y Hongbo mediante el empleo de un clúster IBM P690 cuyo ataque duró una hora de cálculo. El ataque demostró que es posible, computacionalmente, encontrar dos mensajes distintos cuyos hashes sean iguales. Aunque no son conocidos los detalles de cómo se seleccionan los mensajes, en el método empleado se tomaron los valores originales de inicialización de vectores (aunque funciona para cualquier valor inicial dado) y el mismo está basado en: [3] [6]

1. Uso de dos submensajes concatenados (M, Ni) de 512 bits cada uno (=16 palabras)

2. A partir de los submensajes M y Ni (organizados como un vector de 16 palabras) se obtienen los mensajes colisionantes M' y N'i sumando “máscaras” que poseen palabras distintas de cero en las posiciones 4, 11 y 14:

M' = M + (0, 0, 0, 0, 231,..., 215,...., 231, 0)

N'i = N + (0, 0, 0, 0, 231,..., -215,...., 231, 0)

3. Resulta finalmente que

MD5(M, Ni) = MD5(M', N'i)

La tabla 1 muestra dos pares de mensajes (M,N1) y (M,N2) (=1024 bits) a partir de los cuales se generan los pares de mensajes cuyos hashes son iguales, es decir, a partir de M se genera el mensaje M' mediante suma de máscaras como se explica anteriormente, de igual forma, a partir del mensaje Ni se obtiene N'i.. De esta forma (M,Ni) y (M,N'i) producen el mismo hash.

Tabla 1 Mensajes produciendo colisiones [6]

M	2dd31d1 c4eee6c5 69a3d69 5cf9af98 **87b5ca2f** ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 **5a417125** e8255108 9fc9cdf7 **f2bd1dd9** 5b3c3780
N_1	d11d0b96 9c7b41dc f497d8e4 d555655a **c79a7335** cfdebf0 66f12930 8fb109d1 797f2775 eb5cd530 baade822 **5c15cc79** ddcb74ed 6dd3c55f **d80a9bb1** e3a7cc35
M'	2dd31d1 c4eee6c5 69a3d69 5cf9af98 **7b5ca2f** ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 **5a41f125** e8255108 9fc9cdf7 **72bd1dd9** 5b3c3780
N'_1	d11d0b96 9c7b41dc f497d8e4 d555655a **479a7335** cfdebf0 66f12930 8fb109d1 797f2775 eb5cd530 baade822 **5c154c79** ddcb74ed 6dd3c55f **580a9bb1** e3a7cc35
H	9603161f f41fc7ef 9f65ffbc a30f9dbf
M	2dd31d1 c4eee6c5 69a3d69 5cf9af98 **87b5ca2f** ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 **5a417125** e8255108 9fc9cdf7 **f2bd1dd9** 5b3c3780
N_2	313e82d8 5b8f3456 d4ac6dae c619c936 **b4e253dd** fd03da87 6633902 a0cd48d2 42339fe9 e87e570f 70b654ce **1e0da880** bc2198c6 9383a8b6 **2b65f996** 702af76f
M'	2dd31d1 c4eee6c5 69a3d69 5cf9af98 **7b5ca2f** ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 **5a41f125** e8255108 9fc9cdf7 **72bd1dd9** 5b3c3780
N'_2	313e82d8 5b8f3456 d4ac6dae c619c936 **34e253dd** fd03da87 6633902 a0cd48d2 42339fe9 e87e570f 70b654ce **1e0d2880** bc2198c6 9383a8b6 **ab65f996** 702af76f
H	8d5e7019 6324c015 715d6b58 61804e08

SHA-1 (Secure Hash Algorithm, Algoritmo de Hash Seguro)

SHA-1 es una revisión técnica del SHA desarrollada por el NIST para uso con el DSA (Digital Signature Algorithm). SHA-1 funciona similar al MD5, trabaja con mensajes de longitud arbitraria menor que 264 bits, dividiendo estos en bloques de 512 bits que son procesados en 80 vueltas, empleando un vector inicial formado por 5 palabras de 32 bits cada una (en lugar de 4 como MD5) por lo que la salida será un resumen de 160 bits con una complejidad algorítmica de 280. [4] [5]

Para calcular el resumen SHA-1 los pasos a seguir son similares a los de MD5, es decir, se adicionan bits al mensaje hasta conformar uno que sea múltiplo de 512 para procesar el mismo en bloques de 512 bits, este proceso es exactamente igual a como ocurre en MD5. Posteriormente se inicializa un vector de 5 palabras con valores hexadecimales donde las cuatro primeras palabras son las mismas usadas en MD5 pero con el orden de los octetos expresados de manera diferente, es decir, SHA-1 utiliza los octetos de orden superior primero mientras que MD5 emplea los octetos de menor orden primero, así entonces quedaría el vector A, B, C, D y E de la siguiente manera: [5]

A16 = 67452301 D16 = 10325476

B16 = EFCDAB89 E16 = C3D2E1F0

C16 = 98BADCFE

Para obtener el resumen de 128 bits de salida, SHA-1 emplea cuatros funciones (F, G, H, I) o, lo que es lo mismo, una secuencia desde 0 hasta 80 de una de las funciones lógicas, definidas como sigue: [4]

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

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

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

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

Cada función lógica opera con tres palabras de 32 bits B, C, D y produce una salida de una palabra de 32 bits. Tal como ocurre en MD5, estas operaciones lógicas son no lineales sobre tres de las palabras definidas en el vector ABCD. La esencia del cálculo consiste en transformar cada bloque de 16 palabras de mensaje (M0…M15) en 80 palabras de 32 bits (representada por Wt, siendo t=0…79) de acuerdo al siguiente algoritmo: [4] [5] [8]

Para t=0,..,15 Wt = Mt

Para t=16,..,79 Wt = (Wt-3 XOR Wt-8 XOR Wt-14 XOR Wt-16) <<<1

y además: Kt = 5A827999 para t = 0, ..., 19

Kt = 6ED9EBA1 para t = 20, ..., 39

Kt = 8F1BBCDC para t = 40, ..., 59

Kt = CA62C1D6 para t = 60, ..., 79

Como muestra la figura 5, el ciclo operacional de la función SHA-1 para cada vuelta sería:

Siendo ABCDE el vector inicial, para t = 0 hasta 79 hacer:

TEMP = S5(A) + Ft (B,C,D) + E + Wt + Kt

a = e, e = d, d = c, c = b <<<30, b = a,

a = TEMP

donde:

Si : desplazamiento hacia la izquierda en los bits que indica i

Wt: palabra de 32 bits derivada del bloque de 512 bits de entrada

Kt : Constante aditiva

+: Adición módulo 232

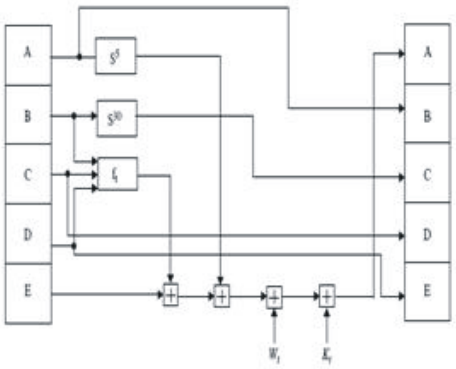

Fig 5. Operación SHA-1 [5]

El procesamiento de cada palabra incluye 80 operaciones a realizar en cada vuelta, para cada vuelta se genera un desplazamiento del registro de los valores del vector ABCDE, obteniéndose un nuevo vector ABCDE para la próxima vuelta. Después de procesados todos los bloques de mensajes la salida es un resumen de 160 bits representado por las cinco palabras expresadas en el vector ABCDE. [4] [5]

El algoritmo SHA-1 es el que emplean la mayoría de las autoridades de certificación actuales, también puede ser empleado en el correo electrónico para la autenticación de las partes con el uso de Algoritmos de firma Digital (DSA); para la transferencia electrónica de fondos, para la distribución de software, para el almacenamiento de datos y para otras aplicaciones que requieren asegurar la integridad de los datos y garantizar la autenticación del origen de los mismos. También esta función forma parte de algunos protocolos de seguridad como TLS y/o SSL v3 (empleado en algunas aplicaciones como por ejemplo los servidores de directorio basados en ldap para la autenticación de usuarios, aunque para ello es necesario el empleo de otros algoritmos que le añaden esta funcionalidad como por ejemplo MAC y HMAC; PGP, S/MIME e IPSec. También es usado por los sistemas UNIX para almacenar las contraseñas de los usuarios en forma segura. [3] [7]

A pesar de considerarse SHA-1 más fuerte ante MD5 debido a que computacionalmente es imposible encontrar un mensaje que corresponda a un resumen dado, o dos mensajes diferentes que produzcan el mismo resumen, su uso posterior ha sido cuestionado ya que en el 2005, el mismo equipo de investigadores chinos que demostraron las colisiones de MD5, demostraron también que son capaces de romper el SHA-1 en al menos 269 operaciones (2000 veces más rápido que un ataque de fuerza bruta) o lo que es lo mismo demostraron que la función SHA-1 no está libre de colisiones. La mayoría de los cálculos y el diseño de los métodos empleados fueron hechos manualmente. Otros investigadores australianos han demostrado que pueden lograr debilitar esta función en 253. Aunque este ataque es de particular importancia para aplicaciones que usan firmas digitales como marcas de tiempo y notarías,

para otras aplicaciones que incluyen además información sobre el contexto se dificulta llevar a la práctica el mismo. Por tal razón el NIST contempla el uso de funciones SHA de mayor tamaño por ejemplo, SHA-256 y SHA-512 bits de longitud. [9] [10]

De manera general y muy brevemente se abordarán otras funciones resúmenes, como por ejemplo: HMAC (Hashed Message Authentication Codes, Código de Autenticación de Mensaje), RIPEMD-160 y HAVAL.

Las funciones vistas anteriormente no han sido diseñadas para la autenticación al carecer de claves secretas; por tal razón la RFC 2104 propone el uso de una autenticación en entornos seguros como SSL mediante una operación MAC en la que intervenga una función hash: su nombre es HMAC, empleada para asegurar y chequear la integridad de la información y la autenticación de las partes en el intercambio de mensajes firmados digitalmente. HMAC es una herramienta que usa una función hash (MD5, SHA-1), una clave K en lo posible mayor que el tamaño del hash en bits, una función xor y operaciones de relleno de bits. En esencia HMAC lo que hace es aplicar una función hash a un mensaje con una clave secreta de 160 bits dando como resultado un resumen de un mensaje al que se le ha aplicado una función hash con una clave secreta (HMACK (M)).[3]. RIPMED-160 es una función criptográfica de 160 bits que surge como reemplazo del MD4, MD5 y RIPE. Existen otras versiones de RIPEMD que producen resúmenes de 320 bits para aplicaciones de funciones hash que requieren un resultado más grande sin requerir mayor nivel de seguridad. [8]. A diferencia de estas funciones que producen resúmenes de longitud fijo, HAVAL es una función hash one-way de longitud variable ya que puede ofrecer una salida de 128, 160, 192, 224 o 256 bits. Para ello HAVAL procesa los mensajes en bloques de 1024 bits y trabaja con 8 variables de 32 bits cada una. A diferencia de las funciones MD5 y SHA-1 cuyos algoritmos consta de un número fijo de vueltas, ésta tiene un número variable de vueltas (desde tres hasta cinco), emplea 7 funciones no lineales cada una de las cuales satisfacen un conjunto de criterios. [8]

Una de las aplicaciones de HAVAL y RIPEMD es en el envío de mensajes firmados y/o privados mediante el empleo de PGP (Pretty Good Privacy) [5]

Conclusiones.

En una emisión de certificado o en la creación de una firma digital, la fortaleza del algoritmo utilizado en la “función hash”, también llamada función resumen o huella digital, es crucial para garantizar la imposibilidad de falsificación, es decir la integridad de la información.

La importancia de la rotura de una función hash se debe interpretar en el siguiente sentido: Un hash permite crear una huella digital, teóricamente única, de un archivo. Una colisión entre hashes supondría la posibilidad de la existencia de dos documentos con la misma huella, aunque fuera trivial encontrar dos ficheros con el mismo resumen criptográfico ello no implicaría que los ficheros fueran congruentes en el contexto adecuado.

Debido a que las funciones hash son normalmente muchos a uno, no se pueden utilizar para determinar con certeza la igualdad de dos mensajes, pero sí para obtener una seguridad razonable de precisión.

Por el descubrimiento de métodos sencillos para generar colisiones de hash, muchos investigadores recomiendan el empleo de otros algoritmos alternativos más fuertes ante ataques criptoanalíticos.

Relacionado con el SHA-1, a pesar de que el NIST contempla funciones de SHA de mayor tamaño (por ejemplo, el SHA-512, de 512 bits de longitud), expertos de la talla de Bruce Schneier abogan por buscar una nueva función hash estandarizada que permita sustituir a SHA-1. Los nombres que se mencionan al respecto son Tiger, de los creadores de Serpent, y WHIRLPOOL, de los creadores de AES.

REFERENCIAS

1. Johner Heinz, Fujiwara Seiei, Sm Yeung Amelia, Stephanou Stephanou, Whitmore Jim: "Deploying a Public Key Infrastructure". Febrero 2000 SG24-5512-00 [Disponible on-line en: http://ps-2.kev009.com:8081/rs6000/redbook-cd/SG245512.pdf]

2. Esteban Marti José Ramón: "Seguridad en la Red - Criptografía" [Disponible on-line en: http://www.seguridadenlared.org/es/index25esp.html]

3. Scolnik Hugo Daniel1, Hecht Juan Pedro: "Impacto de recientes ataques de colisiones contra funciones de hashing de uso corriente". (Versión revisada 2. - 5 de septiembre de2004) [Disponible on-line en: http://www.criptored.upm.es/descargas/ataques_a_hashings.zip]

4. Ramió Aguirre Jorge: "Libro electrónico de Seguridad Informática y Criptografía". 6ta.ed (Versión v 4.1), ISBN: 84-86451-69-8 (2006) Depósito Legal: M-10039-2003, Pp. 711-743 [Disponible on-line en: http://www.criptored.upm.es/descarga/SegInfoCrip_41.zip]

5. Young Rhee Man: "Internet Security Cryptographic Principles, Algorithms and Protocols". ISBN 0-470-85285-2

6. Wang Xiaoyun, Feng Dengguo, Lai Xuejia, Yu Hongbo: "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD". August 17, 2004 [Disponible on-line en: http://eprint.iacr.org/2004/199.pdf]

7. Federal Information Processing Standards Publication 180-1: "Announcing the Standard for SECURE HASH STANDARD". 1995 [Disponible on-line en: http://www.itl.nist.gov/fipspubs/fip180-1.htm]

8. Schneier, Bruce: "Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C (cloth)". ISBN: 0471128457 Publication Date: 01/01/96 [Disponible on-line en:

http://spi2.nisu.org/recop/Applied_Cryptography-Protocols_Algorithms_and_Source_Code_in_C_found_via_www.fileDonkey.com.pdf]

9. Central News Agency: "Chinese Professor Cracks Fifth Data Security Algorithm-SHA-1 added to list of accomplishments". Jan 11, 2007. Artículo electrónico [Disponible on-line en: http://en.epochtimes.com/news/7-1-11/50336.html]

10. Acero Martín, Fernando: "Debilidad del SHA-1". Julio 2009 [Disponible on-line en: http://fernando-acero.livejournal.com/63217.html]