UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Funciones Hash y HMAC

Alan Reyes-Figueroa

Criptografía y Cifrado de Información (Aula 11) 02.septiembre.2021

MACs

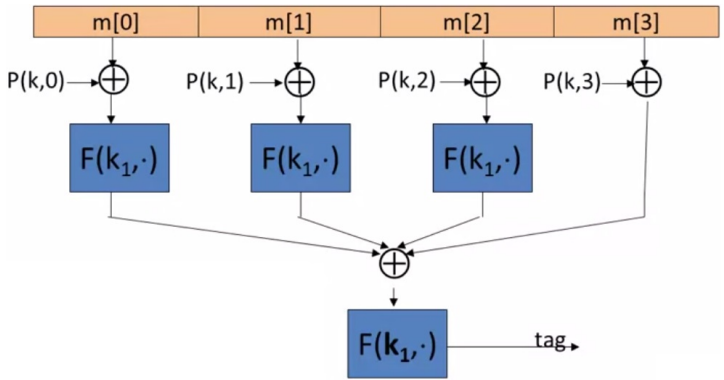
Construcciones de MACs

- **eCBC-MAC** y **CMAC**: usados comunmente con AES (por ejemplo: 802.11i)
- **NMAC (*nested MAC*)**: base para el HMAC.
- **PMAC (*Parallel MAC*)**: esquema en paralelo.
- **Carter-Wegman MAC**: consturido a partir de un One-time MAC.

Parallel MAC

Sea $F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ una PRF. Definimos una nueva PRF de la forma

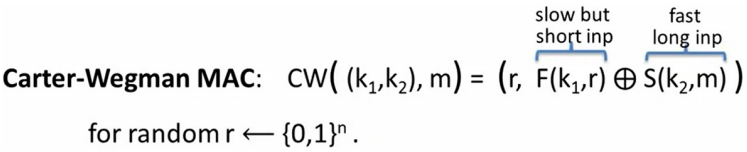
$$F_{PMAC} : \mathcal{K} \times \mathcal{K} \times \mathcal{X}^{\leq L} \to \mathcal{X}.$$



Carter-Wegman MACs

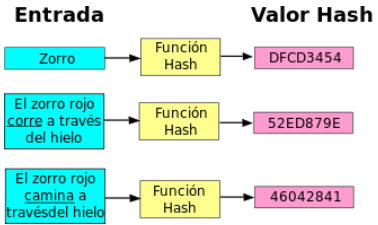
Sea (S, V) un one-time MAC seguro, sobre $(\mathcal{K}_I, \mathcal{M}, \{0,1\}^n)$, y sea $F : \mathcal{K}_F \times \{0,1\}^n \to \{0,1\}^n$ una PRF segura.

Construimos el esquema



Definición

*Una **función hash** es básicamente cualquier función* $H : \mathcal{M} \rightarrow \mathcal{T}$*, donde* $|\mathcal{K}| >> |\mathcal{T}|$.



Se considera que una función hash realiza tres funciones:

1. Convertir claves de longitud variable en cadenas de longitud fija (e.g. PRFs o XOR).
2. Mezclar bits de la clave para que los valores resultantes se distribuyan de manera uniformeme sobre el espacio de claves.
3. Asignar los valores clave en códigos o cadenas de menor longitud.

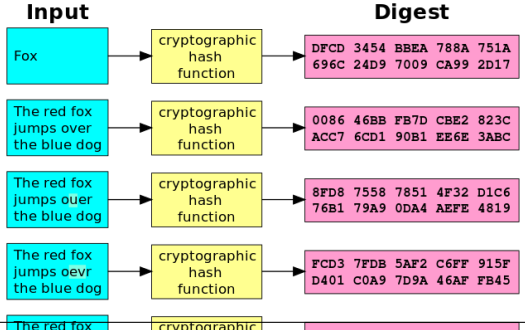
Funciones Hash

Ejemplos de uso:

- Construcción de estructuras de datos complejas: (indexación y búsquedas, tablas hash, árboles de Merkle, listas resumen).
- Construcción de esquemas de compromiso.
- Construcción de algoritmos de cifrado/descifrado.
- Construcción de cadenas pseudo-aleatorias.
- Herramientas para verificar y proteger la integridad de firmas digitales.
- Sumas de verificación (*checksum*).
- Prueba de integridad de contenidos.
- Autenticación de entidades (MAC).
- Herramientas para autenticación y control de acceso (*repositorios de passwords*).
- Protección de claves, derivación de claves.
- Comparación de datos, detección de virus.

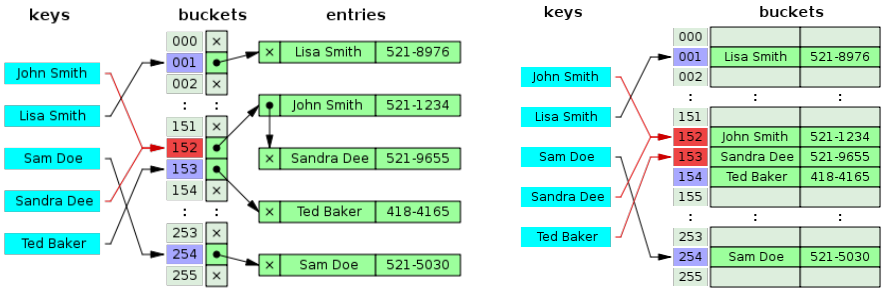
Funciones Hash

Muchas de las aplicaciones anteriores son relativas al campo de la criptografía (cifrado, firma digital, passwords, protocolos de autenticación, etc.). No toda función hash es apta para estos propósitos, se requiere cumplir una serie de propiedades que permiten a las utilidades criptográficas que las utilizan ser resistentes frente ataques. A las funciones hash que cumplen estas propiedades se les llama **funciones hash criptográficas**.



Definición

*Dada una función hash $H : \mathcal{M} \to \mathcal{T}$, una **colisión** para H es un par de mensajes $\mathbf{m}_0, \mathbf{m}_1 \in \mathcal{M}$, con $\mathbf{m}_0 \neq \mathbf{m}_1$, tales que $H(\mathbf{m}_0) = H(\mathbf{m}_1)$.*



Estrategias para tablas hash cuando ocurre colisión.

Funciones Hash

Desde el punto de vista criptográfico, una función hash que produzca fácilmente colisiones, no es segura.
(Esto es porque si $\mathbf{m}_0, \mathbf{m}_1 \in \mathcal{M}$ son tales que $H(\mathbf{m}_0) = H(\mathbf{m}_1)$, un atacante o algoritmo A que esté probando generar pares (mensaje,hash), puede fácilmente hacer falsificación, enviando $(\mathbf{m}_1, H(\mathbf{m}_0))$ como par nuevo).

Definición

*Una función hash $H : \mathcal{M} \to \mathcal{T}$ es **resistente a colisiones** si para todo algoritmos A eficiente*

$$\text{Adv}_{CR}(A, H) = \mathbb{P}(A \text{ produzca una colisión para } H) = \varepsilon$$

es negligible.

De nuevo, la idea es que H es resistente a colisiones, si es muy muy improbable que se produzcan colisiones con entradas aleatorias.

Ejemplo: SHA-256.

MACs a partir de Funciones Hash

Supongamos que $I = (S, V)$ es una MAC para mensajes pequeños, sobre $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, y sea $H : \mathcal{M}^{big} \to \mathcal{M}$ una función hash resistente a colisiones.

Vamos a definir un MAC para cadenas grandes de la sigiuiente forma: Definimos $I^{big} = (S^{big}, V^{big})$, sobre $(\mathcal{K}, \mathcal{M}^{big}, \mathcal{T})$, donde

$$S^{big}(\mathbf{k}, \mathbf{m}) = S(\mathbf{k}, H(\mathbf{m})), \qquad \text{y} \qquad V^{big}(\mathbf{k}, \mathbf{m}, \mathbf{t}) = V(\mathbf{k}, H(\mathbf{m}), \mathbf{t}).$$

Teorema

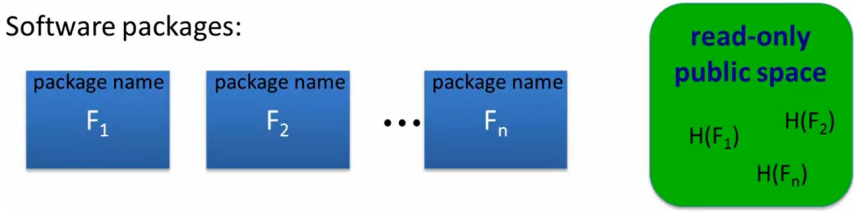
Si $I = (S; V)$ es un MAC seguro, y H es resistente a colisiones, entonces, $I^{big} = (S^{big}, V^{big})$ es un MAC seguro.

Por ejemplo, combinando un AES MAC con un SHA-256.

De nuevo, la longitud del tag no debe ser muy corta. (Veremos en un momento que ahora la longitud se requiere mayor a 180 bits).

Aplicación

Descarga de paquetes o archivos:
Supongamos que un usuario descarga paquetes de archivos desde un sitio web público. El creador del contenido quiere asegurarse que el usuario descarga exactamente los paquetes que no, y no otros. (*e.g.* gcc, gnu, emacs, ...) Al momento de la descarga, puede verificar a través de un sitio de consuta los hash de cada paquete (sólo de lectura).



SI *H* es resistente a colisiones, ningún atacante puede modificar el contenido de la información sin ser detectado.

- **Ventajas**: no requiere clave (es público), requiere espacio de almacenaje.

Ataque a Funciones Hash

Sea $H : \mathcal{M} \to \{0,1\}^n$ una función hash (aquí $|\mathcal{M}| >> 2^n$).

Definimos un algoritmo genérico para hallar colisiones de H en tiempo $O(2^{n/2})$.

Algoritmo: (Ataque a funciones hash)

1. Elegir $2^{n/2}$ mensajes aleatorios $\mathbf{m}_1, \ldots, \mathbf{m}_{2^{n/2}}$ en $\mathcal{M}$.
2. Para cada $i = 1, 2, \ldots, 2^{n/2}$: calcular $\mathbf{t}_i = H(\mathbf{m}_i) \in \{0,1\}^n$.
3. Buscar por colisiones $\mathbf{t}_i = \mathbf{t}_j$, con $i \neq j$.

Si no hallamos colisiones, repetimos los pasos 1, 2 y 3, hasta encontrar una colisión.

Pregunta: ¿Cuántas veces, en promedio, hay que repetir estos pasos, para obtener una colisión?

Para calcular este número, necesitamos hablar de la *paradoja del cumpleaños*.

Teorema (Paradoja del Cumpleaños)

Sean $r_1, r_2, \ldots, r_n \in \{1, 2, \ldots, B\}$ variables aleatorias enteras, idénticamente distribuidas. Cuando $n \approx 1.2B^{1/2}$, entonces la probabilidad de obtener una colisión de estos enteros es $\geq \frac{1}{2}$. Esto es:

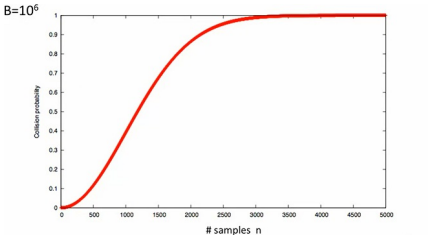
$$\mathbb{P}(\textit{existe } i \neq j \textit{ con } r_i = r_j) \geq \tfrac{1}{2}.$$

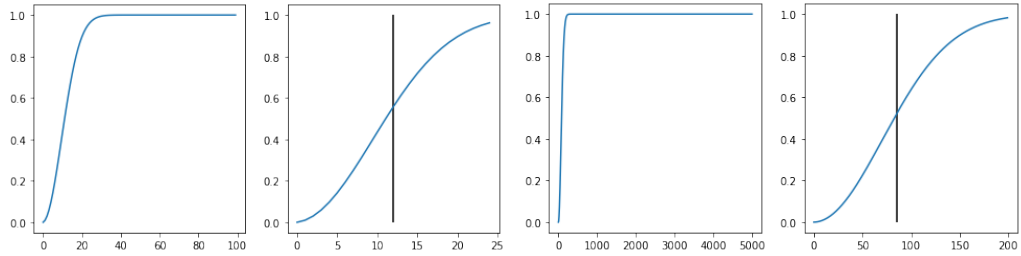
Prueba: En el caso de la distribución uniforme

$$\begin{aligned}
\mathbb{P}(\exists\, i \neq j :\ r_i = r_j) &= 1 - \mathbb{P}(r_i \neq r_j,\ \forall i \neq j) = 1 - \Big(\frac{B-1}{B}\Big)\Big(\frac{B-2}{B}\Big) \cdots \Big(\frac{B-(n-1)}{B}\Big) \\
&= 1 - \prod_{i=1}^{n-1}\Big(1 - \frac{i}{B}\Big) \geq 1 - \prod_{i=1}^{n-1} e^{-i/B} = 1 - \exp\Big(-\frac{1}{B}\sum_{i=1}^{n-1} i\Big) \\
&\geq 1 - e^{-n^2/2B} \approx 1 - e^{-0.72} \approx 0.5132\ldots \geq \tfrac{1}{2}.
\end{aligned}$$

(Recordar que $1 - x \leq 1 - x + \left(\frac{x^2}{2} - \frac{x^3}{6} + \ldots\right) = e^{-x}$.)

Ataque a Funciones Hash





UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Ataque a Funciones Hash

Algoritmo: (Ataque a funciones hash)

1. Elegir $2^{n/2}$ mensajes aleatorios $\mathbf{m}_1, \ldots, \mathbf{m}_{2^{n/2}}$ en $\mathcal{M}$.
2. Para cada $i = 1, 2, \ldots, 2^{n/2}$: calcular $\mathbf{t}_i = H(\mathbf{m}_i) \in \{0,1\}^n$.
3. Buscar por colisiones $\mathbf{t}_i = \mathbf{t}_j$, con $i \neq j$.

Si no hallamos colisiones, repetimos los pasos 1, 2 y 3, hasta encontrar una colisión.

Pregunta: ¿Cuántas veces, en promedio, hay que repetir estos pasos, para obtener una colisión?

La probabilidad de hallar una colisión en una iteración es $\mathbb{P}\frac{1}{2}$.

$\Rightarrow$ el número esperado de iteraciones es 2.

$\Rightarrow$ tiempo de ejecución $= O(2^{n/2})$.

En conclusión: un atacante que conozca alrededor de $\sqrt{n}$ datos $(\mathbf{n}, \mathbf{t})$, con $2^n = |\mathcal{T}|$, tiene alta probabilidad de hallar una colisión.

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Ataque a Funciones Hash

Ejemplos:

	function	digest size (bits)	Speed (MB/sec)	generic attack time
NIST standards	SHA-1	160	153	2^{80}
	SHA-256	256	111	2^{128}
	SHA-512	512	99	2^{256}
	Whirlpool	512	57	2^{256}

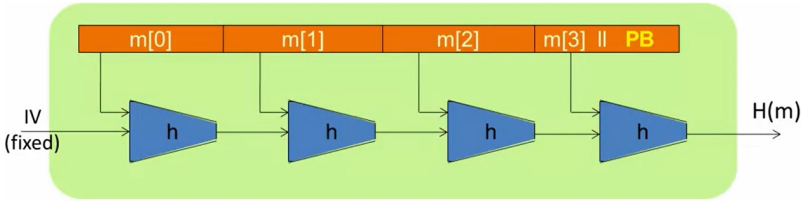
Funciones hash criptográficas seguras.

- SHA = *Secure Hash Algorithm* (Estandar NIST)
- Whirlpool = Creado por los mismos autores de AES (Rijndael).

Paradigma Merkle-Dåmgard

Definimos una construcción de funciones hash grandes, resistente a colisiones, en dos pasos:

- Paso 1: Construir una función hash resistente a colisiones para mensajes pequeños
- Paso 2: Dada *h* pequeña, construimos otra función hash resistente *H*, para mensajes grandes.



Esquema Merkle-Dåmgard.

Dada $h : \mathcal{T} \times \mathcal{X} \to \mathcal{T}$ (función de compresión), obtenemos $H : \mathcal{X}^{\geq L} \to \mathcal{T}$.

Construcción de Funciones Hash

Obs! Funciona similar el NMAC (*nested MAC*).

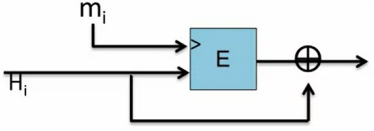
Teorema

Si h es resistente a colisiones $\implies$ *H es resistente a colisiones.*

La pregunta que queda por resolver es ¿Cómo construir funciones hash pequeñas *h*, resistentes a colisiones? Hay varios esquemas

Sea $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ un cifrado por bloques. Definimos

The **Davies-Meyer** compression function: **h(H, m) = E(m, H)⊕H**



Resultado: Si *E* es cifrado seguro, se requieren $O(2^{n/2})$ intentos para hallar una colisión.

Construcción de Funciones Hash

Sea $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ un cifrado por bloques. Definimos

Miyaguchi-Preneel: $h(H, m) = E(m, H) \oplus H \oplus m$ **(Whirlpool)**

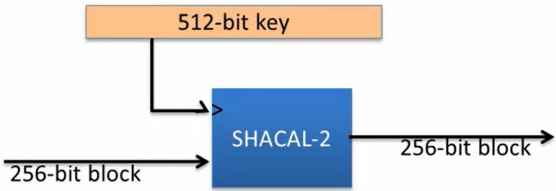
$h(H, m) = E(H \oplus m, m) \oplus m$

total of 12 variants like this

Construcción de Funciones Hash

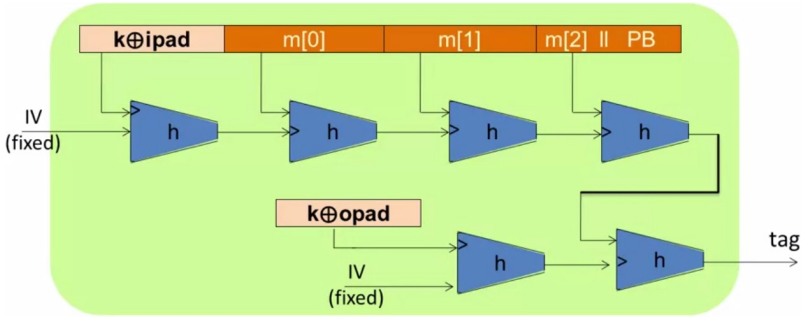
Estudio de Caso : **SHA-256**.

- Merkle-Damgard function
- Davies-Meyer compression function
- Block cipher: SHACAL-2



HMAC

Estudio de Caso : HMAC basado en SHA-256.



- Similar al NMAC.
- Diferencia principal: las claves $\mathbf{k}_1$ y $\mathbf{k}_2$ no son independientes.

Construcción de Funciones Hash

Otra lase de funciones de compresión (hash) viene de la teoría de números:

Logaritmo discreto:

$$a^k \equiv b \pmod{p}.$$

Llamamos k el **logaritmo discreto** de b base a módulo p.

- Hallar potencias $a^k \pmod{p}$ es simple.
- Resolver la ecuación $x^k \equiv b \pmod{p}$ es un problema difícil (cuando p es un primo muy grande).

Funciones de compresión probables: Elegir un primo p grande (digamos de 2000 bits o más), y elegimos $1 \leq u, v \leq p-1$.
Para $m, H \in \{1, 2, \ldots, p-1\}$, definimos

$$h(m, H) = u^H v^m \pmod{p}.$$

Hallar colisiones para h es tan difícil como resolver el problema del logaritmo discreto.