Criptografía y Seguridad Computacional **2016-01**

Clase 5: 30/03/2016

Profesor: Fernando Krell *Notas: Diego Peña*

1. Seguridad en Verificación

Tenemos el siguiente esquema donde se manda un mensaje con tag t de verificación:

$$\texttt{Alice} \xrightarrow{m,t=\mathsf{MAC}_k(m)} \texttt{Bob}$$

Bob acepta el mensaje m si $\mathsf{Vrfy}_k(m,t) = 1$. A partir de esto podemos obtener una nueva definición para seguridad para códigos de autentcación de mensajes mediante el experimento en figura 1.

Sea $\Pi = \langle \mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy} \rangle$ un código de auntenticación de mensajes

$\underline{\mathsf{Exp}^{\mathtt{fals}}_{\Pi,\mathcal{A}}(\lambda)}$**:**

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$
2. $m, t \leftarrow \mathcal{A}^{\mathsf{MAC}_k(\cdot)}(1^\lambda)$

output 1 si, y sólo si, $\mathsf{Vrfy}(m,t) = 1 \wedge m$ no fue consultado al oráculo $\mathsf{MAC}_k(\cdot)$

Figura 1: Experimento de falsificación de códigos de autenticación

Definición 1. *Decimos que* Π *es seguro (infalsificable bajo ataques adaptivos de texto plano escogido) si para todo adversario de tiempo polinomial* $\mathcal{A}$ *existe una funcion negligible* negl *tal que*

$$\Pr[\mathsf{Exp}^{fals}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

Construcción basada en PRF: Sea F es una función pseudoaleatoria $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$ con $n = \text{poly}\lambda$. Podemos construir un código the autenticación de mensajes $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ como:

- $\mathsf{Gen}(1^\lambda)$: output $k \sim U_\lambda$ (llave para F).

- $\mathsf{Mac}_k(m)$: output $F_k(m)$
- $\mathsf{Vrfy}_k(m,t)$: Output 1 si, y sólo si, $F_k(m) = t$.

Teorema 2. *La construcción anterior es un esquema de autenticación de mensajes que satisface definition 1.*

Demostración. Sea $\mathcal{F}$ el conjunto de todas las funciones de $\{0,1\}^n$ a $\{0,1\}^n$, y denotemos $f \sim \mathcal{F}$ el proceso de elegir una función f uniformemente en $\mathcal{F}$. Reduciremos el problema de falsificar un tag para un mensaje al problema de distinguir entre la función F y una función elegida uniformemente en $\mathcal{F}$. Para esto construiremos el siguiente distinguidor para F:

$\underline{\mathsf{D}^{O(\cdot)}}$**:**

1. Corremos $\mathcal{A}^{\mathsf{Mac}'}(\cdot)$ en donde Mac' es la simulación del oráculo Mac usado en el experimento $\mathsf{Exp}^{\mathtt{fals}}$. Este oráculo funciona de la siguiente manera: Inicialmente $Q = \emptyset$, y pára cada consulta m, retorna $O(m)$ y guardamos m en Q. El adversario $\mathcal{A}$ retorna el par m, t.
2. Output 1 si $O(m) = t \wedge m \notin Q$.

Figura 2: Algoritmo distinguidor $D^{O(\cdot)}$

Análisis. Si el oráculo corresponde a una función uniformemente aleatoria, entonces $O(m) = t$ es 2^n si m no fue consultado al oráculo (ya que el valor de $O(m)$ es uniformemente distribuído en $\{0,1\}^n$. Por lo tanto

$$\Pr_{f\sim\mathcal{F}}[D^{f(\cdot)} = 1] = \Pr[t = O(m)] = \mathsf{negl}(\lambda)$$

Si, en cambio, el oráculo $O(\cdot)$ corresponde a la función F, entonces el algoritmo distinguidor D esta simulando perfectamente el experimento $\mathsf{Exp}^{\mathtt{fals}}_{\Pi,\mathcal{A}}(\lambda)$, y por lo tanto:

$$\Pr_{k\sim U_\lambda}[D^{F_k(\cdot)} = 1] = \Pr[\mathsf{Exp}^{\mathtt{fals}}_{\mathcal{A},\Pi} = 1]$$

Deducimos entonces que

$$|\Pr[D^{f(\cdot)} = 1] - \Pr[D^{F_k(\cdot)} = 1]| \geqslant |\mathsf{negl}(\lambda) - Pr[\mathsf{Exp}^{fals}_{\mathcal{A},\Pi} = 1]|$$

Podemos concluir que si existiese un adversario $\mathcal{A}$ que rompe a Π con probabilidad no negligible, entonces F no sería pseudoaleatoria. □

2. MAC de largo variable

En la sección anterior presentamos un MAC para mensajes de largo fijo. ¿Cómo podemos construir un MAC para mensajes de largo variable?

1. Dividir mensaje m en bloques de largo fijo n y aplicamos Π_{LF} (un MAC de largo fijo) a cada bloque (para cada bloque m_i computamos $t_i = \mathsf{Mac}_k(m_i)$ y retornamos $t_1, t_2, ..., t_\ell$).

   ¿Es seguro? No. $\mathcal{A}$ obtiene tag t^* para un mensaje de un solo bloque m^* (vía el oráculo $\mathsf{Mac}_k(\cdot)$) y retorna $(m = m^*||m^*, t = t*||t*)$.

2. Segundo intento: $t_i \leftarrow \mathsf{Mac}_k(i||m_i)$. Cada bloque es concatenado con el índice del bloque.

   Aún no es seguro. Adversario $\mathcal{A}$ puede retornar un mensaje que es el prefijo de uno consultado al oráculo. Consulta por $m* = m_1||m_2$ obteniendo $t^* = t_1, t_2$, y retorna m_1, t_1.

3. Para prevenir el ataque anterior, cada bloque tamnbién es contatenado con el largo total del mensaje ℓ ($t_i \leftarrow \mathsf{Mac}_k(i||\ell||m_i)$).

   El adversario aún puede atacar llamando al orácuo con dos mensajes distintos del mismo largo, y entregar un mensaje nuevo haciendo una mezcla por bloque de los mensajes anteriores (por ejemplo, usar bloque 1 del primer mensaje, pero el bloque 2 del segundo mensaje).

4. Idea final: Aplicar aleatoriedad. Elegimos $r \sim U_\lambda$, y cada bloque se tagea como $t_i \leftarrow \mathsf{Mac}_k(r||i||\ell||m_i)$

   Por supuesto, se utiliza en el mismo r para todos los bloques de un mensaje, siendo la probabilidad de que el mismo r se repita en 2 consultas al orácuo negligible.

   Demostración. Bosquejo: Sea E el evento que $\mathsf{Exp}^{fals}_{\mathcal{A},\Pi} = 1$ para algún adversario $\mathcal{A}$. Entonces,

   $$\Pr[E] = \Pr[E \wedge \ r \ repite] + \Pr[E \wedge \ rr \ no \ repite]$$

   $\Pr[E \wedge \ r \ repite] \leq \Pr[r \ repite]$, que sabemos que es es negligible en λ si el adversario sólo utiliza el orácuo un número polinomial de veces (ver ataque de los cumpleaños más abajo). Para computar $\Pr[E \wedge \ rr \ no \ repite]$ analizamos los siguientes 2 casos:

1 $\exists i$ tal que m_i que no fue consultado al oráculo en el output del adversario.

2 $\forall m_i$ fue consultado al oráculo.

Por definición del experimento, la probabilidad de que ocurra el segundo caso es 0, pues no se permite output 1 si hubo una consulta al oráculo usando m_i. La primera posibilidad por otra parte es igual a falsificar Π_{LF}. □

¿Cuál es el problema con esta construcción? El tag es 4 veces mayor al mensaje, siendo problemático para mensajes muy largos.

3. CBC-MAC

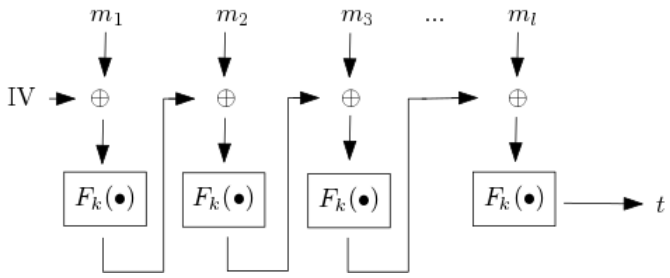

Figura 3: Diagrama de funcionamiento de CBC-MAC

Este algoritmo es seguro pero para mensajes del mismo largo l (queda como ejercicio la demostración). De lo contrario el adversario puede hacer lo siguiente:

1. Consultar por $m_1 \to t_1 = F_k(m_1)$
2. Consultar por $m_2 = t_1 \to t_2 = F_k(F_k(m_1))$

Cómo $m_2 \oplus F_k(m_1) = F_k(m_1) = 0^n$, el adversario puede retornar un nuevo mensaje $m3 = m_1||0^n$ con un tag válido $t_3 = t_1||t_2$.

Esto se puede arreglar utilizando F con una nueva llave al oputput de CBC-MAC.

4. Funciones de Hash

Las funciones de hash son funciones que encogen el input. Por ejemplo, $H : \{0,1\}^{2n} \to \{0,1\}^n$. Una función de hash la podriamos usar, por ejemplo, para computar MAC's de la siguiente manera: Usamos H para reducir el largo del input, y luego usamos una función psuedo aleatoria (o un MAC de largo fijo pequeño). Sin embargo, es claro que si un adversario encuentra una colisión ($m1, m2$ tales que $H(m1) = H(m2)$), entonces el esquema MAC no puede ser seguro ya que el adversario podria usar $m1$ para obtener un tag válido via una llamada al oráculo, y este MAC seria válido para $m2$.

Por lo tanto, para aplicaciones criptográficas queremos evitar colisiones. Lamentablemente, es garantizado que las colisiones ocurran (ya que el espacio de input es mayor al espacio de output. Sin embargo, podemos utilizar el enfoque computacional y requerir que encontrar colisiones es "difícil" para cualquier algoritmo de tiempo polinomial.

En particular tenemos las siguientes tres defuniciones para funciones de hash (de ás fuerte a más débil) Una función de hash se considera segura si:

1 Es resistente a colisiones. Es decir que dado un s aleatorio (conocido para el adversario) la probabilidad de encontrar un par de mensajes x y x' tales que $x \neq x'$ y $H^s(x) = H^s(x')$ en tiempo polinomial es negligible en el largo de los elementos del rango de H.

2 Es resistente a segunda preimagen. Dado un x uniforme en el dominio de H la probabilidad de cualquier algoritmo polinomial de encontrar un x' tal que $x \neq x'$ y $H^k(x) = H^k(x')$ es negligible en el largo de los elementos del rango de H.

3 Es resistente a primera preimagen. Dado un valor y en el rango de H,la probabilidad de encontrar x tal que $y = H^s(x)$ es negligible

En la tarea se les pide demostrar que si H es resistente a colisiones, entonces H resistente a segunda preimagen, y que si H resistente a segunda preimagen, entonces H resistente a preimagen.

Observación 3. *La llave s es pública (conocida por el adversario), ¿Por qué se necesita? Ninguna función sin llave puede ser resistente a colisiones. Esto es debido a que está garantizado de que existe, al menos en teoría, un algoritmo que retorna la colisión (imaginen el algoritmo que tiene como codigo x, x' tal que $H(x) = H(x')$, y retorna x, x'). Este algoritmo encuentra la colisión con probabilidad 1. Por este motivo introducimos una llave aleatoria, de manera que estos algoritmos ganen con probabilidad negligible. Sin embargo, en la práctica utilizamos funciones sin llaves como SHA1,SHA256,SHA3, etc. A pesar de que en teoría existen algoritmos que encuentren colisiones para estas funciones, estos son realmente difíciles de encontrar.*

En la práctica, las funciones de hash tienen como dominio $\{0,1\}^*$, y como output $\{0,1\}^n$ ($n = 160, 256$ por ejemplo). Sin embargo, para construir tal función de hash, se utiliza una de dominio de largo fijo $m > n$. Dado $h : \{0,1\}^m \to \{0,1\}^n$ resistente a colisiones, podemos construir $H : \{0,1\}^* \to \{0,1\}^n$ utilizando la transformada de Merkle-Damgärd:

Dividimos el input en bloques de largo $m - n$ cada uno (imaginar $m = 2n$), el último bloque se agregan 0's hasta que sea de largo $m - n$. Además se le agrega un bloque adicional que contiene el largo original del input (de manera que x, $x||0$, $x||00$, etc. tengan distinto hash).

Demostramos ahora que si h es resistente a colisiones, entonces H construida utilizando la transformada de Merkle-Damärd es resistente a colisiones.

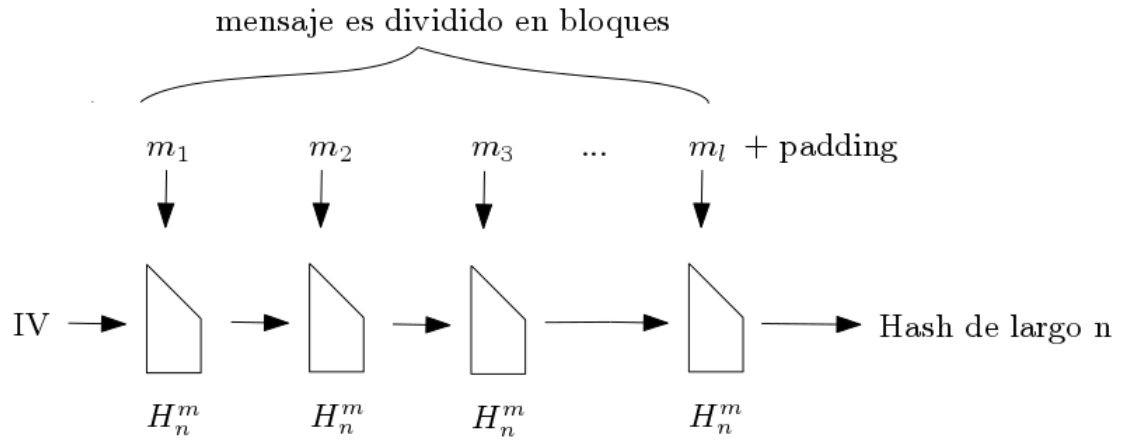

Figura 4: Diagrama de funcionamiento de Merkle-Damgärd. El IV suele ser 0^n

Sea $\mathcal{A}$ un adversario que encuentra una colisión en H, es decir, encuentra $x, x' \in \{0,1\}^*$ tales que $H(x) = H(x')$. Sea ℓ, ℓ' los largos de x y x', $x_1, x_2, ..., x_B$ los bloques de x, y $x'_1, x'_2, ..., x'_B$ la cantidad de bloques de largo $m - n$ de x' para aplicar la transformada sobre x y x' respectivamente. En la transformada a x se le agrega un ultimo bloque x_{B+1} con valor ℓ (lo mismo para x').

Sea z_i el output del i-ésimo paso en la transformada. Es decir, $z_0 = IV = 0^n$, $z_1 = h(z_0||m_1)$, ..., $z_i = h(z_{i-1}||m_i)$ y $z_{B+1} = h(z_B||L) = H(m)$. Analizaremos 2 casos. 1) Si $L \neq L'$, entonces la colisión para h es $z_B||L$ y $z'_{B'}||L'$. 2) Si $L = L'$, entonces $B = B'$. Sea $I_i = z_{i-1}||x_i$, $I'_i = z'_{i-1}||x'_i$, $I_{B+2} = z_{B+1}$, y $I'_{B+2} = z'_{B+1}$. Sea N el máximo tal que $I_N \neq I'_N$ (sabemos que $x \neq x'$, por lo que $x_i \neq x'_i$ para algun i y entonces N esta bien definido). Dado que $I_{B+2} = I_{B+2}$ (colisión en H), N es menor o igual a $B+1$. Como N es máximo, $I_{N+1} = I'_{N+1}$, por lo tanto I_N, I'_N es una colisión.

Aplicaciones de funciones de Hash

Sea H una función resistente a colisiones y un $\Pi = (\mathsf{Gen}, \mathsf{MacVrfy})$ un esquema de autenticación infalsificable para mensajes de largo fijo n, y $H = (\mathsf{Gen}, \mathsf{Hash})$ una función resistente a colisiones con input de largo variable. Podemos construir un $\Pi' = (\mathsf{Gen}', \mathsf{Mac}', \mathsf{Vrfy}')$ infalsificable para mensajes de largo no definido:

- $\mathsf{Gen}'(1^\lambda)$
  - $s \leftarrow \mathsf{Gen}_H(1^\lambda)$
  - $k \leftarrow \mathsf{Gen}_\Pi(1^\lambda)$

- $\mathsf{Mac}'_{\langle s,k\rangle}(m) = \mathsf{Mac}_k(H^s(m))$
- $\mathsf{Vrfy}'_{\langle s,k\rangle}(m,t)$
  1. $y = H^s(m)$
  2. $\mathtt{outputVrfy}_k(t,y)$

Sea $\mathcal{A}'$ un adversario que intenta quebrar Π'. Si $\mathcal{A}'$ gana entonces H no es resistente a colisiones o Π es falsificable. Informalmente, tenemos:

$$\Pr[\mathcal{A}' \ gana] = \Pr[\mathcal{A}' \ gana \wedge colisión] + \Pr[\mathcal{A}' \ gana \wedge !colisión]$$

Pero

$$\Pr[\mathcal{A}' \ gana \wedge colisión] \leq \Pr[colisión] = negl(\lambda)$$

y por otro lado,

$$\Pr[\mathcal{A}' \ gana \wedge !colisión] = \Pr[\ quiebra \ \Pi] = negl(\lambda)$$

Ejercicio: Formalizar el argumento anterior.

Otras aplicaciones pueden ser:

1. Huellas de virus: Antiviruses cuentan con hashes de viruses con la que comparan programas que se ejecutan en un equipo.
2. Encontrar duplicados: Evitar mal uso de memoria manteniendo hashes de archivos y evitando generación de copias.
3. Derivación de llaves: Passwords y huellas digitales almacenarlas como una sola llave uniforme.

Ataque generico a funciones de hash: El ataque de los cumpleaños

Supongamos $H(x)$ se distribuye uniformemente en $0,1^n$ (es decir, supongamos que H es una función aleatoria de). Entonces podemos analizar la probabilidad de colisión como la probabilidad de que en q valores $Y_1, ..., Y_q$ distribuidos uniformemente e independientement en $0,1^n$, exista i,j tal que $Y_i = Y_j$. Luego, es posible demostrar que

$$\Pr[\exists i,j | Y_i = Y_j] = \frac{1}{2} \ si \ q = 2^{\frac{n}{2}}$$

Por lo tanto cualquier función de hash puede ser quebrada utilizando $2^{\frac{n}{2}}$ valores de $H(\cdot)$. Podemos concluir que, para tener seguridad 2^λ, el output de una función de hash resistente a colisiones debe ser mayor a 2λ.

Árboles de Merkle

Supongamos que tenemos un conjunto de archivos, en una base de datos externa (Dropbox, por ejemplo). Para verificar que un archivo obtenido de una base de datos no se haya modificado, podemos comparar el hash del archivo obtenido con el valor hash almacenado. ¿Qué hacemos en el caso de tener muchos documentos? Es incómodo e ineficiente mantener un hash para cada documento. Una solución es armar un árbol de hashes para optimizar el proceso de verificación:

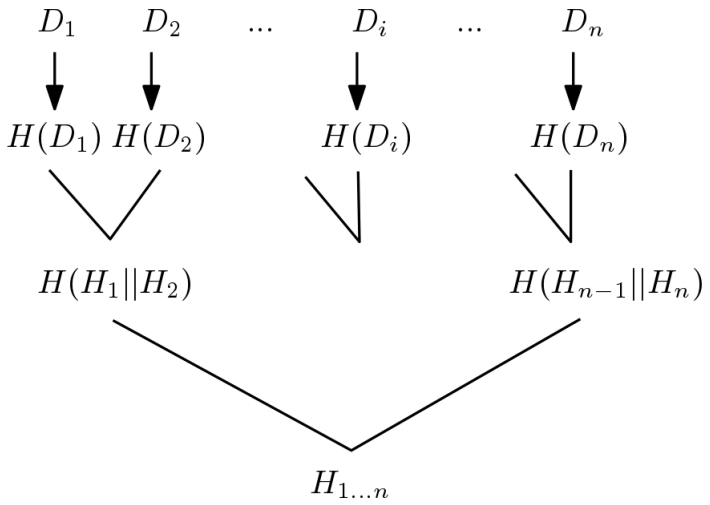

Figura 5: Construcción de árbol de Merkle

Formamos un árbol en donde las hojas estan asociadas con cada documento. Cada nodo del árbol contiene el hash de la concatenación de sus nodos hijos. El cliente, solo mantiene la raiz del árbol. Esto le permite verificar la autenticidad de cualquier documento requerido al servidor de la siguiente manera. Dado un indice i, el servidor provee el documento i junto a los $\log n$ hashes de los nodos hermanos del camino de la hija i hasta la raiz. El cliente, luego, puede verificar que el documento es autentico, pues si no lo fuera, el computo de la raiz no concidiria con el almacenado localmente o el servidor fue capaz de encontrar una colisión.

Por ejemplo, para calcular un D_5 en el siguiente árbol se requeriría D_5, H_6, $H_{7,8}$ y $H_{1,4}$: Si al final se obtiene un $H_{1..n}$ correspondiente al total del grupo de archivos,

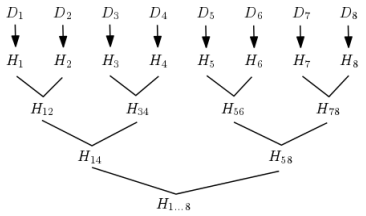

Figura 6: Ejemplo árbol de Merkle

entonces el D verificado en cuestión es el documento original. Si no, D no está correcto u ocurrió una colisión en medio de la operación.

Ejericio: Demostrar formalmente que el esquema es seguro.