Funzioni Hash

Paolo D'Arco
pdarco@unisa.it

Universitá di Salerno

Elementi di Crittografia

Contenuti

Funzioni Hash

Una funzione hash offre un modo per mappare una stringa di input lunga in una stringa di output piú corta, chiamata *digest* (impronta digitale)

Requisito primario: evitare collisioni

Le funzioni hash hanno innumerevoli applicazioni

- estensione del dominio dei Mac
- modellate come "funzioni impredicibili" sono ampiamente usate nella progettazione di protocolli crittografici

Da un punto di vista teorico stanno "nel mezzo" tra:

Critt. a chiave privata $\rightarrow$ funzioni hash $\leftarrow$ Critt. a chiave pubblica

Funzioni Hash

Nel mezzo perché?

- nella pratica vengono costruite utilizzando tecniche della crittografia a chiave privata
- da un punto di vista teorico l'esistenza di funzioni hash resistenti a collisioni sembra un'assunzione piú forte dell'esistenza di PRF (ma piú debole della Critt. a chiave pubblica)

Nelle strutture di dati le funzioni hash vengono usate per costruire tabelle hash

$$h : U \rightarrow \{1, \ldots, N\}$$

$h(x)$, con $x \in U$, é la posizione nella tabella T in cui x viene memorizzato.

Le funzioni hash rendono possibili ricerche in tempo $O(1)$.

Funzioni Hash

Una "buona" funzione hash riduce le "collisioni", cioé gli x, x', tali che $h(x) = h(x')$.

Poiché $|U| >> N \quad \Rightarrow \quad$ le collisioni esistono.

Nota che le collisioni implicano extra-spazio/extra-lavoro (chaining/open addressing)

Le funzioni hash **crittografiche** resistenti a collisioni sono simili nello spirito ma

Funzioni Hash

Nelle strutture dati

Evitare collisioni é un **desiderio**
al fine di ottenere efficienza maggiore

L'insieme dei dati é quasi sempre
indipendente dalla funzione hash
e **non ha** il fine di produrre collisioni

In Crittografia

Evitare collisioni é un
requisito fondamentale

Adv puó scegliere elementi
del dominio **con l'obiettivo**
esplicito di causare collisioni

Resistenza a collisioni

H é resistente a collisioni (collision-resistant) se é impraticabile per qualsiasi Adv ppt trovare una collisione in H.

Consideriamo soltanto $H : D \rightarrow R$ tali che $|D| > |R|$.

Funzioni hash con chiave: H é una funzione a due input

$$s, x \qquad \rightarrow \qquad H(s, x) \stackrel{def}{=} H^s(x)$$

Deve essere difficile trovare una collisione in $H^s(\cdot)$ per un s generato a caso da $Gen(1^n)$.

Nota che rispetto agli schemi di cifratura a chiave privata alcuni valori di s possono non essere generati da $Gen(1^n)$ (non tutti gli s corrispondono a chiavi valide).

Funzioni Hash

Inoltre, s (generalmente) **non** é segreto

$\Rightarrow$ useremo la notazione H^s invece di H_s per ricordarci di ció.

Definizione 5.1. Una funzione hash (con output di lunghezza ℓ) é una coppia di algoritmi ppt (Gen, H) tali che:

- Gen é un algoritmo ppt che prende in input 1^n e dá in output s
- H prende in input s ed una stringa $x \in \{0,1\}^*$ e dá in output $H^s(x) \in \{0,1\}^{\ell(n)}$

Se H^s é definita solo per $x \in \{0,1\}^{\ell'(n)}$, con $\ell'(n) > \ell(n)$, allora (Gen, H) é una funzione hash a lunghezza fissa per input di lunghezza ℓ'

H é detta **funzione di compressione**

Nota: senza "compressione" la resistenza a collisioni é facile $\rightarrow H^s(x) = x$!

Sicurezza

Al solito, definiamo la sicurezza attraverso un esperimento.

Siano $\Pi = (Gen, H)$, Adv A, ed n par. di sicurezza.

The collision-finding experiment $\mathsf{Hash\text{-}coll}_{\mathcal{A},\Pi}(n)$**:**

1. *A key s is generated by running* $\mathsf{Gen}(1^n)$.
2. *The adversary $\mathcal{A}$ is given s and outputs x, x'. (If Π is a fixed-length hash function for inputs of length $\ell'(n)$, then we require $x, x' \in \{0,1\}^{\ell'(n)}$.)*
3. *The output of the experiment is defined to be 1 if and only if $x \neq x'$ and $H^s(x) = H^s(x')$. In such a case we say that $\mathcal{A}$ has found a collision.*

Sicurezza

Definizione 5.2. Una funzione hash $\Pi = (Gen, H)$ é resistente a collisioni se, per ogni Adv ppt A, $\exists$ una funzione trascurabile $negl$ tale che

$$Pr[\text{Hash-coll}_{A,\Pi}(n) = 1] \leq negl(n).$$

Nota: deviazione dal modello. Nella pratica vengono usate funzioni hash "senza chiave", con lunghezza dell'output fissata. Cioé:

$$H : \{0,1\}^* \rightarrow \{0,1\}^{\ell}.$$

Tuttavia, coppie che "collidono" sono ancora non note e difficili da trovare

$\Rightarrow$ Le prove di sicurezza per funzioni hash "del mondo reale" sono ancora **significative** fino a quando la prova mostra che un avversario efficiente che rompe lo schema in esame puó essere usato per trovare esplicitamente collisioni in H

Nozioni di sicurezza piú deboli

Nozioni piú deboli sono:

- Second pre-image resistance: data s ed un x scelto unif. a caso, é impraticabile per ogni Adv ppt trovare un $x' \neq x$ tale che $H^s(x') = H^s(x)$
- Pre-image resistance: data s ed un y scelto unif. a caso in $\{0,1\}^\ell$, é impraticabile per ogni Adv ppt trovare un x tale che $H^s(x) = y$.

É facile vedere che:

- Collision resistance $\Rightarrow$ Second pre-image resistance

  Infatti se, data s ed un x uniforme, fosse possibile trovare un x' tale che $H^s(x') = H^s(x) \Rightarrow$ la coppia (x, x') sarebbe una collisione

Funzioni Hash

É facile anche vedere che:

- Second pre-image resistance $\Rightarrow$ Pre-image resistance

  Infatti se, data s ed un y uniforme, fosse possibile trovare un x tale che $H^s(x) = y$

  $$\Downarrow$$

  Adv sceglierebbe x' unif. a caso, calcolerebbe $y' = H^s(x')$, calcolerebbe una pre-imagine di y', chiamiamola x, e con alta probabilitá risulterebbe $x \neq x'$

  $$\Downarrow$$

  x sarebbe una seconda pre-immagine di y'

Estensione del dominio

Progettazione di funzioni hash:

- prima progettiamo una funzione di compressione
- poi ne estendiamo il dominio per input di lunghezza arbitraria

Senza perdita di generalitá, supponiamo che:

$$(Gen, h) \quad \text{sia tale che comprima} \quad 2n \text{ bit} \quad \rightarrow \quad n \text{ bit}$$

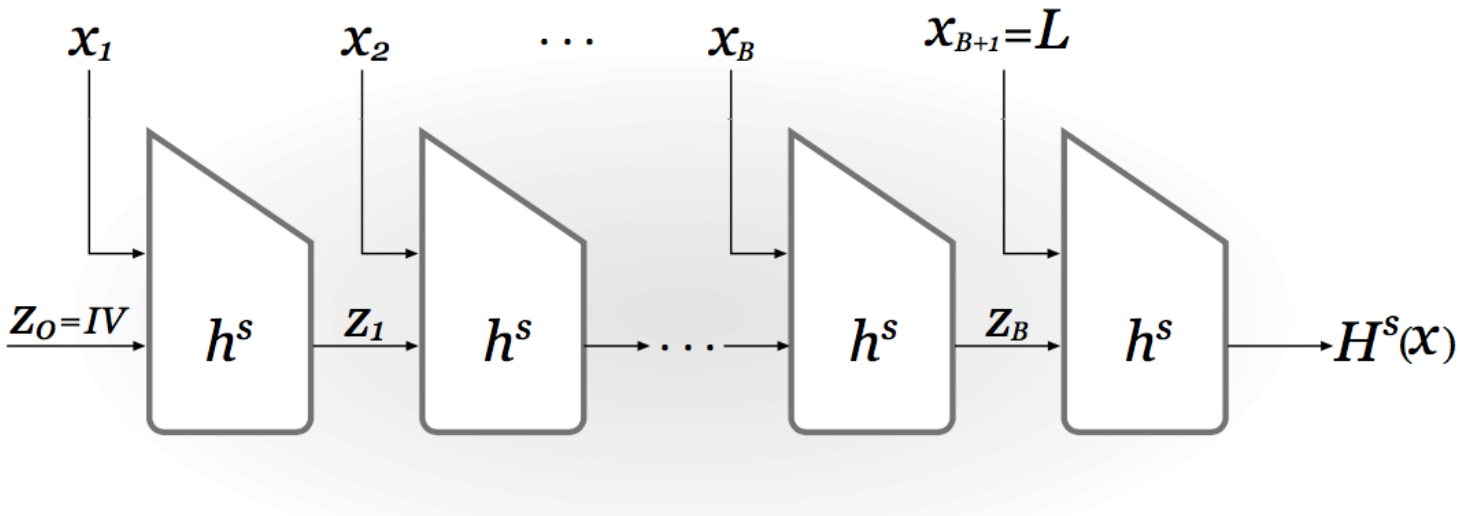
$$\Downarrow$$

$$(Gen, H), \text{ usando la trasformazione di Merkle-Damgard,}$$

$$x \in \{0,1\}^* \quad \rightarrow \quad y \in \{0,1\}^n$$

Trasformazione Merkle-Damgard

Graficamente, la trasformazione opera come segue:



Il messaggio x é diviso in blocchi $x_1, \ldots, x_B$ di n bit.

L'extra blocco x_{B+1} codifica la lunghezza di x con una stringa di n bit.

Trasformazione Merkle-Damgard

CONSTRUCTION 5.3

Let (Gen, h) be a fixed-length hash function for inputs of length $2n$ and with output length n. Construct hash function (Gen, H) as follows:

- Gen: remains unchanged.
- H: on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^n$, do the following:
  1. Set $B := \lceil \frac{L}{n} \rceil$ (i.e., the number of blocks in x). Pad x with zeros so its length is a multiple of n. Parse the padded result as the sequence of n-bit blocks $x_1, \ldots, x_B$. Set $x_{B+1} := L$, where L is encoded as an n-bit string.
  2. Set $z_0 := 0^n$. (This is also called the IV.)
  3. For $i = 1, \ldots, B+1$, compute $z_i := h^s(z_{i-1} \| x_i)$.
  4. Output z_{B+1}.

Trasformazione Merkle-Damgard

Teorema 5.4. Se (Gen, h) é resistente rispetto a collisioni, allora anche (Gen, H) lo é.

Prova. Mostriamo che, per *qualsiasi* s, una collisione in H^s dá una collisione in h^s.

Siano x ed x' due stringhe differenti di lunghezza L ed L' tali che $H^s(x) = H^s(x')$

$$x = x_1 \dots x_B x_{B+1} \qquad x' = x'_1 \dots x'_{B'} x'_{B'+1}$$

Ci sono due casi da considerare.

Trasformazione Merkle-Damgard

Caso 1. $L \neq L'$. Gli ultimi passi nel calcolo di $H^s(x)$ e di $H^s(x')$ sono

$$z_{B+1} = h^s(z_B||x_{B+1}) \quad \text{e} \quad z'_{B'+1} = h^s(z'_{B'}||x'_{B'+1})$$

$$\text{Ma } H^s(x) = H^s(x') \quad \Rightarrow \quad z_{B+1} = z'_{B'+1}$$

$$\Rightarrow \quad w = z_B||x_{B+1} \quad \text{e} \quad w' = z'_{B'}||x'_{B'+1}$$

sono una collisione per h^s, essendo $w \neq w'$ dato che $x_{B+1} \neq x'_{B'+1}$.

Trasformazione Merkle-Damgard

Caso 2. $L = L'$. Quindi $B = B'$ e $x_{B+1} = x'_{B+1}$.

Siano:

$z_1, \ldots, z_{B+1}$ i valori prodotti dal calcolo di $H^s(x)$.

$I_1, \ldots, I_{B+1}$ gli input per h^s, cioé $I_i = z_{i-1}||x_i$, per $i = 1, \ldots, B+1$.

$z'_1, \ldots, z'_{B+1}$ i valori prodotti dal calcolo di $H^s(x')$.

$I'_1, \ldots, I'_{B+1}$ gli input per h^s, cioé $I'_i = z'_{i-1}||x'_i$, per $i = 1, \ldots, B+1$.

Poniamo inoltre $I_{B+2} = z_{B+1}$ e $I'_{B+2} = z'_{B+1}$.

Sia N il **piú grande** indice per cui risulta $I_N \neq I'_N$.

Poiché $x \neq x'$, deve esistere un i tale che $x_i \neq x'_i \quad \Rightarrow \quad N$ certamente esiste.

Trasformazione Merkle-Damgard

D'altra parte, dato che

$$I_{B+2} = z_{B+1} = H^s(x) = H^s(x') = z'_{B+1} = I'_{B+2}.$$

deve essere $N \leq B + 1$.

Per definizione, N é l'indice piú grande per cui $I_N \neq I'_N$. Quindi:

$$I_{N+1} = I'_{N+1} \quad \Rightarrow \quad z_N = z'_N$$

$$\Downarrow$$

$$h^s(I_N) = z_N = z'_N = h^s(I'_N)$$

e quindi le stringhe I_N ed I'_N sono una collisione per h^s.

Paradigma Hash-and-Mac

Al momento sappiamo autenticare messaggi di lunghezza arbitraria in diversi modi

- Costruzione generica
- CBC-Mac
- GCM e Poly1305

Un'altra modalitá fa uso delle funzioni hash.

$$\text{Idea: } m \in \{0,1\}^*, \qquad y = H^s(m), \qquad Mac_k(y)$$

H^s é collision-resistant. Il Mac viene calcolato su $H^s(m)$ invece che su m.

Paradigma Hash-and-Mac

CONSTRUCTION 5.5

Let $\Pi = (\mathsf{Mac}, \mathsf{Vrfy})$ be a MAC for messages of length $\ell(n)$, and let $\Pi_H = (\mathsf{Gen}_H, H)$ be a hash function with output length $\ell(n)$. Construct a MAC $\Pi' = (\mathsf{Gen}', \mathsf{Mac}', \mathsf{Vrfy}')$ for arbitrary-length messages as follows:

- Gen': on input 1^n, choose uniform $k \in \{0,1\}^n$ and run $\mathsf{Gen}_H(1^n)$ to obtain s; the key is $k' := \langle k, s \rangle$.
- Mac': on input a key $\langle k, s \rangle$ and a message $m \in \{0,1\}^*$, output $t \leftarrow \mathsf{Mac}_k(H^s(m))$.
- Vrfy': on input a key $\langle k, s \rangle$, a message $m \in \{0,1\}^*$, and a MAC tag t, output 1 if and only if $\mathsf{Vrfy}_k(H^s(m), t) \stackrel{?}{=} 1$.

Paradigma Hash-and-Mac

La costruzione é sicura se Π é un Mac sicuro per messaggi di lunghezza fissa e Π_H é collision-resistant.

Intuizione: Π_H collision-resistant $\Rightarrow$ autenticare $H^s(m)$ é "essenzialmente uguale" ad autenticare m.

Precisamente: supponiamo che un mittente usi la costruzione per autenticare un insieme di messaggi Q, ed A riesca a produrre una falsificazione per $m^* \notin Q$.

Ci sono due casi:

Paradigma Hash-and-Mac

1. $\exists$ un messaggio $m \in Q$ tale che $H^s(m) = H^s(m^*)$

   $\Rightarrow$ A ha trovato una collisione per H^s

2. $\forall m \in Q$, $H^s(m) \neq H^s(m^*)$.

   $\Rightarrow$ A ha trovato un tag valido rispetto a Π per $H^s(m^*)$.

Paradigma Hash-and-Mac

Teorema 5.6. Se Π é un Mac sicuro per messaggi di lunghezza ℓ e Π_H é resistente a collisioni, allora la Costruzione 5.5 é un Mac sicuro per messaggi di lunghezza arbitraria.

Dim. Sia Π' la Costruzione 5.5 e A' un Adv che attacca Π'.

In una esecuzione di $Mac\text{-}forge_{A',\Pi'}(n)$ sia $k' =< k, s >$ e sia Q l'insieme dei messaggi di cui A' chiede i tag.

Sia $m^* \notin Q$. Indichiamo con $Coll$ l'evento in $Mac\text{-}forge_{A',\Pi'}(n)$

"c'é un messaggio $m \in Q$ per cui $H^s(m) = H^s(m^*)$."

Risulta $Pr[Mac\text{-}forge_{A',\Pi'}(n) = 1]$

$$
\begin{aligned}
&= Pr[Mac\text{-}forge_{A',\Pi'}(n) = 1 \wedge Coll] + Pr[Mac\text{-}forge_{A',\Pi'}(n) = 1 \wedge \overline{Coll}] \\
&\leq Pr[Coll] + Pr[Mac\text{-}forge_{A',\Pi'}(n) = 1 \wedge \overline{Coll}]
\end{aligned}
$$

Paradigma Hash-and-Mac

Mostreremo che entrambi i termini sono trascurabili.

Intuitivamente, il primo é trascurabile per via di Π_H.

L'algoritmo C che segue usa A' che attacca Π' per trovare collisioni per Π_H

Algorithm C:
The algorithm is given s as input (with n implicit).

- Choose uniform $k \in \{0,1\}^n$.
- Run $\mathcal{A}'(1^n)$. When $\mathcal{A}'$ requests a tag on the ith message $m_i \in \{0,1\}^*$, compute $t_i \leftarrow \mathsf{Mac}_k(H^s(m_i))$ and give t_i to $\mathcal{A}'$.
- When $\mathcal{A}'$ outputs (m^*, t), then if there exists an i for which $H^s(m^*) = H^s(m_i)$, output (m^*, m_i).

Paradigma Hash-and-Mac

Analisi: C computa in tempo polinomiale.

Quando s viene generato da $Gen_H(1^n)$, la "vista" di A' quando eseguito come subroutine di C é distribuita *identicamente* alla "vista" che A' ha quando esegue in $Mac\text{-}forge_{A',\Pi'}(n)$.

Poiché C dá in output una collisione **esattamente** quando essa occorre, risulta

$$Pr[Hash\text{-}Coll_{C,\Pi_H}(n) = 1] = Pr[Coll].$$

Dall'assunzione che Π_H é collision-resistant, segue che $\exists$ *negl* tale che

$$Pr[Hash\text{-}Coll_{C,\Pi_H}(n) = 1] \leq negl(n)$$

$$\Downarrow$$

$$Pr[Coll] \leq negl(n).$$

Paradigma Hash-and-Mac

Il secondo termine é trascurabile per via della sicurezza dello schema Mac Π.

L'algoritmo A attacca Π in $\mathit{Mac\text{-}forge}_{A,\Pi}(n)$.

Adversary $\mathcal{A}$:
The adversary is given access to a MAC oracle $\mathsf{Mac}_k(\cdot)$.

- Compute $\mathsf{Gen}_H(1^n)$ to obtain s.
- Run $\mathcal{A}'(1^n)$. When $\mathcal{A}'$ requests a tag on the ith message $m_i \in \{0,1\}^*$, then: (1) compute $\hat{m}_i := H^s(m_i)$; (2) obtain a tag t_i on $\hat{m}_i$ from the MAC oracle; and (3) give t_i to $\mathcal{A}'$.
- When $\mathcal{A}'$ outputs (m^*, t), then output $(H^s(m^*), t)$.

Analisi: A computa in tempo polinomiale.

Paradigma Hash-and-Mac

La "vista" di A' quando eseguito come subroutine di A é distribuita *identicamente* alla "vista" che A' ha quando esegue in $\mathit{Mac\text{-}forge}_{A',\Pi'}(n)$.

Quando entrambi gli eventi "$\mathit{Mac\text{-}forge}_{A',\Pi'}(n) = 1$" e "$\overline{\mathit{Coll}}$" si verificano, A dá in output una falsificazione (t é un tag valido per $H^s(m^*)$ rispetto a Π). Infatti, poiché Coll **non** si verifica, $H^s(m^*)$ **non** é stata una query di A all'oracolo $\mathit{Mac}_k(\cdot)$. Quindi:

$$Pr[\mathit{Mac\text{-}forge}_{A,\Pi}(n) = 1] = Pr[\mathit{Mac\text{-}forge}_{A',\Pi'}(n) = 1 \wedge \overline{\mathit{Coll}}].$$

Dall'assunzione che Π é un Mac sicuro, segue che $\exists$ negl tale che

$$Pr[\mathit{Mac\text{-}forge}_{A,\Pi}(n) = 1] \leq \mathit{negl}(n)$$

$$\Downarrow$$

$$Pr[\mathit{Mac\text{-}forge}_{A',\Pi'}(n) = 1 \wedge \overline{\mathit{Coll}}] \leq \mathit{negl}(n).$$

HMAC

É possibile costruire uno schema Mac sicuro per messaggi di lunghezza arbitraria, basandosi direttamente su una funzione hash?

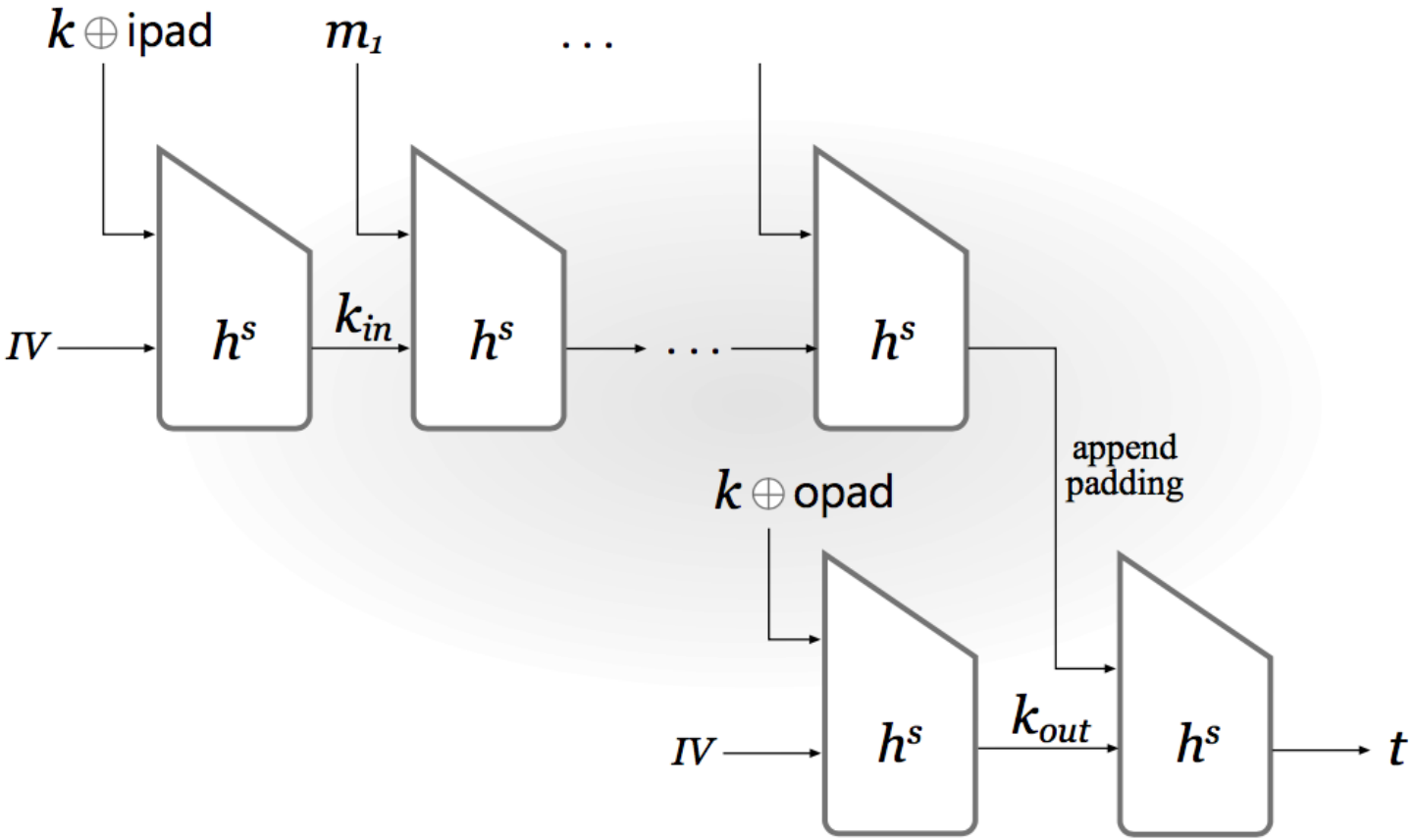
$$Mac_k(m) = H(k||m) \qquad \text{non é una buona scelta}$$

Si puó dimostrare che é completamente insicura se H é progettata usando la trasformazione di Merkle-Damgard.

Idea: usare "due livelli" di hash: $H^s(k_1, H^s(k_2, m))$

- un primo livello per creare il "digest"
- un secondo per autenticare

HMAC


$k \oplus$ ipad
m_1
...
IV
h^s
k_{in}
h^s
...
h^s
append padding
$k \oplus$ opad
IV
h^s
k_{out}
h^s
t

CONSTRUCTION 5.7

Let (Gen_H, H) be a hash function constructed by applying the Merkle–Damgård transform to a compression function (Gen_H, h) taking inputs of length $n + n'$. (See text.) Let opad and ipad be fixed constants of length n'. Define a MAC as follows:

- Gen: on input 1^n, run $\mathsf{Gen}_H(1^n)$ to obtain a key s. Also choose uniform $k \in \{0,1\}^{n'}$. Output the key $\langle s, k \rangle$.
- Mac: on input a key $\langle s, k \rangle$ and a message $m \in \{0,1\}^*$, output

$$t := H^s\Big((k \oplus \mathsf{opad}) \,\|\, H^s\big((k \oplus \mathsf{ipad}) \,\|\, m\big)\Big).$$

- Vrfy: on input a key $\langle s, k \rangle$, a message $m \in \{0,1\}^*$, and a tag t, output 1 if and only if $t \stackrel{?}{=} H^s\big((k \oplus \mathsf{opad}) \,\|\, H^s\big((k \oplus \mathsf{ipad}) \,\|\, m\big)\big)$.

HMAC

Osservazioni: la funzione di compressione

$$h : \{0,1\}^{n+n'} \rightarrow \{0,1\}^{n}$$

corrisponde alla funzione $h : \{0,1\}^{2n} \rightarrow \{0,1\}^{n}$ nell'analisi della trasformazione di Merkle-Damgard ($n' = n$).

La lunghezza del messaggio nella trasformazione viene codificata con un blocco extra x_{B+1}. In realtá, in pratica viene codificata in una porzione di blocco, usando ℓ bit.

Il messaggio x viene completato con zeri fino ad ottenere una lunghezza *multiplo* di n' *a meno* di ℓ bit. Poi viene aggiunta $L = |x|$, codificata con ℓ bit (assumiamo che $n + \ell < n'$ nella costruzione).

Perché dovremmo convincerci che HMAC é sicuro?

HMAC

Puó essere vista come una specifica istanza del paradigma Hash-and-Mac.

Precisamente, HMAC opera come segue:

- associa una stringa corta ad un messaggio di lunghezza arbitraria

$$y := H^s((k \oplus ipad)||m)$$

- calcola con una funzione (a chiave segreta)

$$t := H^s((k \oplus opad)||\tilde{y})$$

Ma possiamo essere piú precisi. Sia $\tilde{H}^s(m) \stackrel{def}{=} H^s((k \oplus ipad)||m)$

$\Rightarrow$ $\tilde{H}^s$ é collision-resistant se h lo é, per *qualsiasi* valore di $k \oplus ipad$.

HMAC

Inoltre, il primo passo nel calcolo di $t := H^s((k \oplus opad)|| \widetilde{y})$ consiste nel calcolare il valore

$$k_{out} \stackrel{def}{=} h^s(IV \,||\, k \oplus opad)$$

per poi calcolare

$$t := h^s(k_{out} \,||\, \widetilde{y})$$

Il valore $\widetilde{y}$ é y "con pad", i.e., include la lunghezza che é $n + n'$, codificata con ℓ bit. Pertanto, se trattiamo k_{out} come una stringa uniforme e assumiamo che

$$\widetilde{Mac}_k\,(y) \stackrel{def}{=} h^s(k||y)$$

sia un Mac sicuro a lunghezza fissa, allora HMAC é un'instanziazione di Hash-and-Mac.

HMAC

Precisamente:

$$HMAC_{s,k} = \widetilde{Mac}_{k_{out}}\left(\widetilde{H}^{s}\left(m\right)\right)$$

A cosa servono le costanti *ipad* ed *opad*? A derivare efficientemente due chiavi da una sola.

Perché incorporare k nella computazione "piú interna"? Occorre che la funzione hash sia collision-resistant e una chiave non é necessaria ma ...

Rende possibile provare la sicurezza della costruzione su una assunzione "piú debole", la *weak collision-resistance* (resistenza a collisioni debole).

Nell'esperimento $Hash\text{-}Coll_{A,\Pi}$, l'Adv A interagisce con un oracolo $H^{s}_{k_{in}}(\cdot)$ che restituisce $H^{s}_{k_{in}}(m)$ su m come richiesta.

$H^{s}_{k_{in}}(\cdot)$ usa la trasformata di Merkle-Damgard applicata ad h^{s}, ma usando come *IV* la *chiave segreta* k_{in}.

HMAC

Osservazione: H collision-resistant $\Rightarrow$ H weakly collision-resistant

La seconda é potenzialmente piú semplice da soddisfare.

Usare $\Pi = (Gen_H, H)$ weakly collision-resistant é una buona strategia difensiva.

Caso reale: MD5, funzione hash usata ampiamente in passato.

Primi attacchi mostrarono MD5 non collision-resistant ... ma ancora weakly collision resistant

Gli sviluppatori che usavano MD5 in HMAC ebbero "tempo" per sostituirla con un'altra funzione hash.

Usare assunzioni piú deboli é sempre preferibile.

HMAC

Due chiavi indipendenti -interna ed esterna - ed uniformi dovrebbero essere usate.

Definiamo:

$$G^s(k) = h^s(IV||(k \oplus opad)) \,||\, h^s(IV||(k \oplus ipad)) = k_{out} \,||\, k_{in}$$

Se assumiamo che G^s sia un PRG per qualsiasi valore di s, allora k_{out} e k_{in} possono essere considerate chiavi *indipendenti ed uniformemente distribuite*.

Teorema 5.8. Sia G^s un PRG per qualsiasi s, sia $\tilde{Mac}_{k_{out}}(\cdot)$ un Mac sicuro per messaggi di lunghezza fissa n, e sia (Gen_H, H) una funzione hash weakly collision-resistant. Allora HMAC é un MAC sicuro per messaggi di lunghezza arbitraria.

HMAC in pratica: ampiamente usato, piú efficiente di CBC-MAC.

Esercizi

Si formalizzi lo sketch della prova di sicurezza per la trasformata di Merkle-Damgard esibendo una riduzione rigorosa.