Applicazioni delle Funzioni Hash

Paolo D'Arco
pdarco@unisa.it

Universitá di Salerno

Elementi di Crittografia

Contenuti

Applicazioni aggiuntive delle funzioni hash

Le funzioni hash sono utili per molteplici scopi.

- *Fingerprinting e deduplication:*

  H collision resistant, $H(x)$ digest di x (e.g., file)

  Due digest uguali $\Rightarrow$ con alta prob. $x_1 = x_2$ (file uguali)

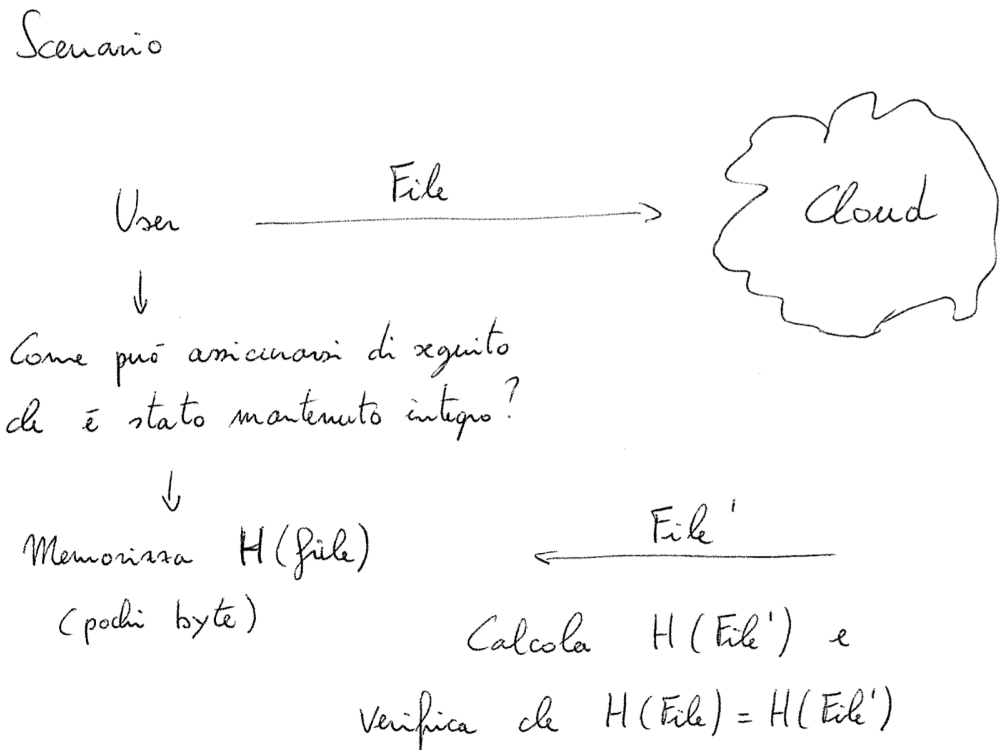
  Idea applicata in:

  - **Virus fingerprinting:** un database memorizza i digest calcolati sul codice dei virus. Viene usato come look-up table. L'hash di un file viene confrontato con i digest.
  - **Deduplication:** se in un servizio di tipo cloud (e.g., google drive) un file é giá presente, un nuovo upload viene evitato. Per controllare la presenza del file il digest viene confrontato con i digest giá presenti nel database
  - **Sistemi P2P:** contengono tabelle con i digest dei file disponibili, per velocizzare le ricerche

Merkle Tree

Tecnica che usa le funzioni hash per autenticare efficientemente gruppi di file.

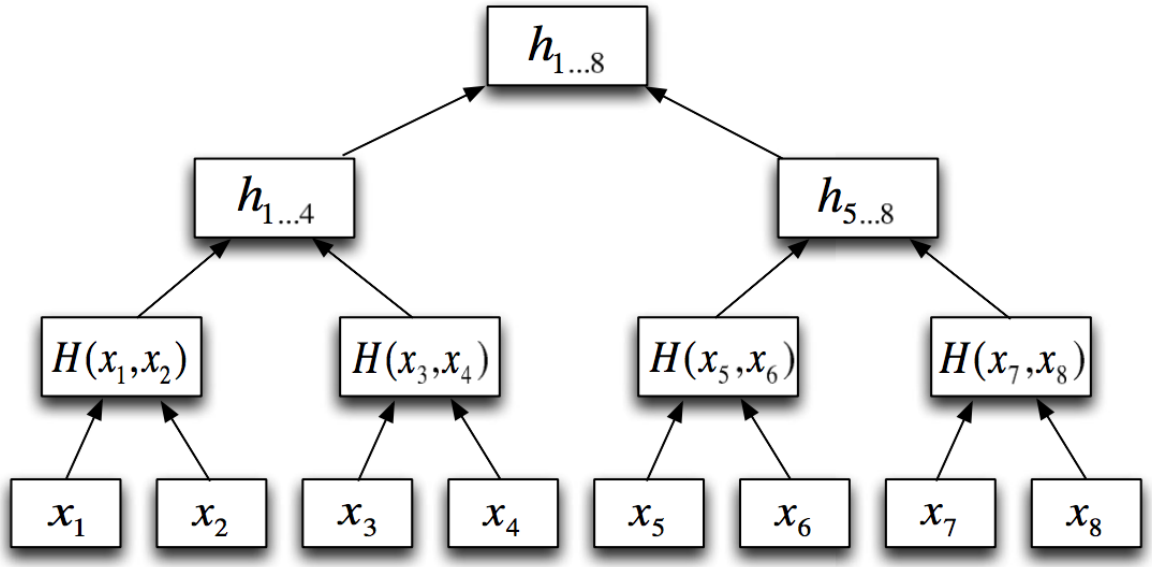
Offre anche un metodo alternativo alla trasformazione di Merkle-Damgard per estendere il dominio di una funzione hash collision-resistant



Merkle Tree

E in presenza di file multipli $x_1, \ldots, x_t$?

$\rightarrow$ Memorizza $H(x_1), \ldots, H(x_t)$ (num di hash lineare nel numero di file). Alternativa:



Calcola e memorizza localmente il valore $h_{1...8}$. Fa l'upload dei file $x_1, \ldots, x_8$ nel cloud.

Merkle Tree

Successivamente, per verificare l'integrita di x_i, riceve dal cloud x_i e tutti i valori hash associati ai nodi **adiacenti** alla path da x_i alla radice

$\Rightarrow$ una "prova" π_i che x_i é stato usato per calcolare $h_{1\ldots8}$ e.g., per verificare x_3, l'utente riceve $x_4, h_{1,2}$, e $h_{5\ldots8}$.

Nota sull'efficienza: per evitare di inviare x_4, il Merkle tree viene solitamente costruito a partire dagli hash dei file.

Teorema 5.11. Sia (Gen_H, H) collision-resistant. Allora (Gen_H, MT_t) é collision-resistant per t fissato.

Intuizione per la prova: se il cloud potesse provare che $x_j \notin \{x_1, \ldots, x_t\}$ é parte dell'albero

$\Rightarrow$ il cloud sarebbe in grado di calcolare collisioni di H efficientemente

Merkle Tree

Lo schema richiede:

- spazio di memorizzazione per l'utente costante
- complessitá di comunicazione $O(\log t)$
- complessitá di calcolo (verifiche hash) $O(\log t)$

I Merkle tree sono ampiamente usati nelle applicazioni reali.

Per esempio, sono il meccanismo che Bitcoin usa per raggruppare e autenticare efficientemente le transazioni.

Controllo delle password

Invece di memorizzare la *pwd* nel laptop, memorizza $H(pwd)$

$$\Downarrow$$

se viene rubato, grazie alla "preimage resistance" di H é difficile recuperare la pwd.
Se la $pwd \in D$ (Dizionario Lingua Inglese, circa 80000 termini) dove $|D|$ é piccola, una volta ottenuto il digest *hpwd* di *pwd*, puó essere applicata una ricerca esaustiva

$$\Rightarrow \text{provo tutte le } pwd \in D \text{ fino a quando risulta } H(pwd) = hpwd$$

$$\Downarrow$$

(strategia nota come "attacco del dizionario")

Obiettivo: vorremmo mostrare che contro questo schema di protezione delle *pwd*, l'attacco del dizionario é il **meglio** che *Adv* possa fare.

Controllo delle password

Purtroppo la proprietá "pre-image resistance" non é sufficiente

1. richiede che x sia scelto **uniformemente** in $\{0,1\}^n$ affinché $H(x)$ sia difficile da invertire, non dice nulla su spazi come D con distribuzioni arbitrarie

Se peró modelliamo H come un **oracolo casuale**, allora possiamo provare il risultato desiderato. Cioé, non c'é altra strada che valutare H per recuperare *pwd* da *hpwd* e, assumendo che *pwd* sia scelta in modo uniforme in D, tale attacco richiede $|D|/2$ valutazioni di H in media. Infatti, consideriamo l'evento

$$X_{val} = \text{"valutazioni richieste"}.$$

Essendo H totalmente casuale:

$$Pr[X_{val} = i] = Pr[hpwd \text{ corrisponda alla } i\text{-esima } pwd \text{ di D}] = \frac{1}{|D|}.$$

Controllo di password

Pertanto, risulta:

$$\begin{aligned} Exp(X_{val}) &= \sum_{i=1}^{|D|} i \cdot Pr(X_{val} = i) = \sum_{i=1}^{|D|} i \cdot \frac{1}{|D|} \\ &= \frac{1}{|D|} \cdot \sum_{i=1}^{|D|} i = \frac{1}{|D|} \cdot \frac{|D| \cdot (|D|+1)}{2} = \frac{|D|+1}{2} \end{aligned}$$

Osservazione: abbiamo mostrato prima che si puó *usare* un random oracle come un PRG e come una funzione collision resistant e che puó essere usato per costruire una PRF.

Qui abbiamo mostrato il primo esempio di come il ROM possa essere usato per provare un risultato che con le assunzioni standard non si riesce a provare.

Rafforzare lo schema delle *pwd*

- Rallentare il calcolo dell'hash

  Calcolare H^I, e.g., se $I = 1000$, H^I richiede mille iterazioni di H

  Non é un problema per l'utente. É un problema per *Adv*.

- Uso di un "salt"

  $$(s, hpwd = H(s||pwd)) \quad s \text{ unico per utente}$$

  Il salt é sconosciuto ad Adv, prima di leggere il file con i digest.

  Nessun pre-processing sul dizionario delle pwd puó essere effettuato.

  Se un salt diverso viene usato per ogni file, é richiesta una diversa ricerca esaustiva

Derivazione delle chiavi

I crittosistemi a chiave privata richiedono una chiave segreta che sia una stringa di bit uniformemente distribuita.

Le parti possono tentare di usare direttamente le informazioni che condividono

- troncando alla lunghezza giusta
- mappando l'informazione segreta condivisa su stringhe ad-hoc

Non é un approccio sicuro.

Esempio. Supponiamo che le due parti condividano una *pwd* di 28 lettere maiuscole casuali. Hanno bisogno di una chiave di 128 bit. Ci sono:

$$26^{28} > 2^{130} \qquad \text{chiavi possibili}$$

Se la *pwd* é in codice ASCII, allora dispongono di 224 bit condivisi.

Derivazione delle chiavi

Un'idea puó essere troncare la *pwd* per ottenere 128 bit. I primi 16 byte sono sufficienti.

Tuttavia, si noti che le *rappresentazioni* ASCII delle lettere A-Z vanno da $0x41$ a $0x5A$

$$\Downarrow$$

i primi tre bit di ogni carattere sono 010

$$\Downarrow$$

il $37,5\%$ dei bit della chiave ottenuta sono fissati

$$\Downarrow$$

ci sono soltanto 2^{75} chiavi possibili invece di 2^{128}

Derivazione delle chiavi

Pertanto, abbiamo bisogno di un buon metodo per *estrarre* dall'informazione condivisa una stringa uniforme.

Abbiamo cioé bisogno di una soluzione generica per derivare una chiave da un segreto condiviso - ad alta *entropia* - non necessariamente uniforme

Entropia $\rightarrow$ "il log della cardinalitá dello spazio delle possibili chiavi".

Nell'esempio precedente lo spazio delle chiavi ha 75 bit di entropia.

Esistono diverse misure dell'entropia.

Definizione. Una distribuzione di probabilitá X ha m bit di **min-entropy** se, per qualsiasi fissato valore di x, risulta

$$Pr_{x \leftarrow X}[X = x] \leq 2^{-m}$$

Derivazione delle chiavi

Esempi. La distribuzione uniforme S ha min-entropy $\log |S|$.

Se una distribuzione associa ad un elemento prob. $1/10$ e ai restanti 90 elementi prob. $1/100$, allora la distribuzione ha min-entropy:

$$1/10 \leq 2^{-m} \iff 2^m \leq 10 \iff m \leq \log 10.$$

La min-entropy di una distribuzione misura la probabilitá con cui *la strategia migliore* dell'attaccante indovina un valore campionato della distribuzione.

$$\text{min-entropy } m \Rightarrow \textit{Adv} \text{ ha una probabilitá di indovinare} \leq 2^{-m}$$

Funzioni hash H come funzioni di derivazione

Una *funzione di derivazione* delle chiavi fornisce un modo per ottenere una stringa (quasi) uniformemente distribuita da una distribuzione ad alta min-entropy.

Se modelliamo H come un oracolo casuale, allora H **puó essere utilizzata** come una buona funzione di derivazione.

Consideriamo l'incertezza di un Adv su $H(x)$ (X distribuzione con alta min-entropy).

Ogni query dell'attaccante puó essere vista come un tentativo di indovinare il valore.

Se l'attaccante **non** invia x come query all'oracolo, allora $H(x)$ é una stringa uniformemente distribuita ai suoi occhi (prima proprietá del random oracle).

Un Adv che effettua q query ha al piú probabilitá $q \cdot 2^{-m}$ di indovinare $H(x)$.

Funzioni hash H come funzioni di derivazione

Esistono funzioni di derivazione le cui proprietá di sicurezza possono essere argomentate senza far ricorso al ROM. Per esempio, possono essere utilizzate funzioni hash con chiavi chiamate *estrattori forti*.

Uno standard per la derivazione delle chiavi é HKDF (vedi riferimenti libro).

Commitment Scheme

Una parte puó "vincolarsi" (commit) ad un messaggio m, inviando un valore vincolante (commitment value) *com*, tale che valgono due proprietá:

- *Hiding*: il commitment non rivela nulla su m
- *Binding*: é impraticabile per la parte che fa il commitment dare in output un valore *com* che successivamente potrá "aprire" in due modi diversi (due messaggi diversi m ed m').

Il commitment value é una sorta di busta digitale.

Commitment Scheme

Formalmente, uno schema di commitment non interattivo é una tripla di algoritmi $\Pi = (Gen, Com, Decom)$ tale che:

- Gen, probabilistico, genera i parametri *params* pubblici
- $Com(params, r, m) \rightarrow com$ genera il commitment value *com*, su input *params*, una stringa random r ed il messaggio m
- $Decom(com) \rightarrow (r, m)$, dá in output la stringa casuale r ed il messaggio m usati per calcolare *com*

Nella fase di commitment, *Alice* invia a *Bob* *com*.

Nella fase di decommitment, *Alice* "apre" il commitment inviando a *Bob* (r, m).

Bob usa (r, m) per verificare che $com = Com(params, r, m)$

Proprietá di Hiding e di Binding

Le proprietá di Hiding e di Binding vengono formalizzate attraverso due esperimenti

The commitment hiding experiment $\mathsf{Hiding}_{\mathcal{A},\mathsf{Com}}(n)$**:**

1. *Parameters* $\mathsf{params} \leftarrow \mathsf{Gen}(1^n)$ *are generated.*
2. *The adversary* $\mathcal{A}$ *is given input* params*, and outputs a pair of messages* $m_0, m_1 \in \{0,1\}^n$*.*
3. *A uniform* $b \in \{0,1\}$ *is chosen and* $\mathsf{com} \leftarrow \mathsf{Com}(\mathsf{params}, m_b; r)$ *is computed.*
4. *The adversary* $\mathcal{A}$ *is given* com *and outputs a bit* b'*.*
5. *The output of the experiment is* 1 *if and only if* $b' = b$*.*

Proprietá di Hiding e di Binding

Similmente per la proprietá di binding

The commitment binding experiment $\mathsf{Binding}_{\mathcal{A},\mathsf{Com}}(n)$**:**

1. *Parameters* $\mathsf{params} \leftarrow \mathsf{Gen}(1^n)$ *are generated.*
2. $\mathcal{A}$ *is given input* params *and outputs* $(\mathsf{com}, m, r, m', r')$.
3. *The output of the experiment is defined to be* 1 *if and only if* $m \neq m'$ *and* $\mathsf{Com}(\mathsf{params}, m; r) = \mathsf{com} = \mathsf{Com}(\mathsf{params}, m'; r')$.

Definizione di sicurezza

Usando i due esperimenti precedenti, é possibile definire formalmente cosa si intende per schema di commitment sicuro

DEFINITION 5.13 *A commitment scheme* Com *is* secure *if for all* PPT *adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr\left[\mathsf{Hiding}_{\mathcal{A},\mathsf{Com}}(n)=1\right] \leq \frac{1}{2}+\mathsf{negl}(n)$$

and

$$\Pr\left[\mathsf{Binding}_{\mathcal{A},\mathsf{Com}}(n)=1\right] \leq \mathsf{negl}(n).$$

Commitment in ROM

Usando il random oracle model é facile costruire uno schema di commitment.

- per fare il commit di m, si sceglie $r \in \{0,1\}^n$ in modo uniforme e si dá in output $com := H(r||m)$ (*Gen* e *params* non sono necessari poiché la funzione hash H é pubblica)
- per fare il decommit, si invia (r, m) e il ricevente verifica che $com = H(r||m)$

Hiding: se A non invia query del tipo $H(r||\cdot)$, allora $H(r||m)$ non rivela nulla di m. E poiché r é uniforme, A lo indovina con probabilitá trascurabile

Binding: H é resistente a collisioni $\Rightarrow$ se A ha successo, allora A ha trovato una collisione.

Commenti finali

Nota: schemi di commitment possono essere costruiti senza far ricorso al ROM, ma le costruzioni sono piú complicate e meno efficienti.

In conclusione, possiamo dire che

- le funzioni hash hanno molteplici usi e sono un tool importante in crittografia
- il random oracle model *rende possibile* provare la sicurezza di costruzioni per cui risulta difficile nel modello standard e *rende facile* la realizzazione di diverse primitive crittografiche
- il random oracle model é un modello di progettazione ed analisi di protocolli controverso per le ragioni teoriche discusse
- le costruzioni usate nella pratica basate sulle funzioni hash sono efficienti