Codici per l'autenticazione di messaggi

Paolo D'Arco

Codici per l'autenticazione di messaggi

Comunicazione *sicure* su un canale pubblico:

- La cifratura puó essere usata per prevenire che ascoltatori non autorizzati carpiscano informazioni sul contenuto dei messaggi inviati sul canale sprotetto

$$\Downarrow$$

Confidenzialitá

- Ogni parte peró dovrebbe essere in grado anche di identificare il mittente del messaggio, esser certo che ha inviato il messaggio e capire se il messaggio é integro o é stato modificato

$$\Downarrow$$

Autenticitá, Integritá

Codici per l'autenticazione di messaggi

Esempi dal mondo reale:

- Utente che comunica con la banca. Richiede il trasferimento di 1000 $ da X a Y
  - é la richiesta autentica?
  - é stato alterato il totale?
  - le tecniche di rilevamento e correzione degli errori standard non sono utili qui: affrontano un altro problema.
- Cookies usati da un server web
  - *HTTP* é un protocollo senza stato. Aiutano ad implementare una "sessione"
  - l'utente non dovrebbe essere in grado di modificarli
  - il server web dovrebbe essere in grado di verificarne autenticitá ed integritá

Codici per l'autenticazione di messaggi

In generale, l'autenticitá di una comunicazione non puó essere basata su normali informazioni tipo

- identificatori degli SMS
- ID del chiamante
- indirizzo email
- ...

$\Rightarrow$ non rappresentano una *prova* dell'origine.

Inoltre, non possiamo evitare che le informazioni vengano alterate. Ma possiamo far sí che *ogni* alterazione venga riconosciuta.

Vedremo come ottenere autenticitá ed integritá usando tecniche crittografiche.

Cifratura: strumento per autenticitá e integritá?

La cifratura **non** risolve il problema dell'autenticitá e dell'integritá.

Esempio. Cifratura attraverso:

$$Enc_k(m) = G(k) \oplus m$$

dove G é un PRG costruito a partire da uno stream cipher.

Il cifrato é molto facile da manipolare:

- l'*inversione* di un qualsiasi bit nel cifrato risulta nell'inversione dello stesso bit nel messaggio in chiaro

$$\Downarrow$$

Le implicazioni possono essere nefaste. In un trasferimento bancario:

- inversione del bit meno significativo $\Rightarrow$ nessun impatto
- inversione dell'11-esimo bit $\Rightarrow$ impatto significativo!

Cifratura usando cifrari a blocchi?

Nota: tutti gli schemi di cifratura studiati fino ad ora sono tali che:

ogni cifrato corrisponde ad un qualche messaggio in chiaro

$\Downarrow$

Adv puó scegliere una stringa c (cifrato arbitrario) e inviarlo ad una parte

$\Downarrow$

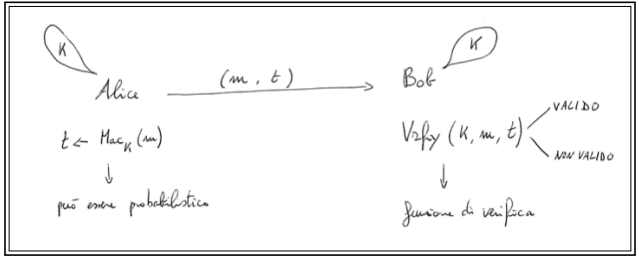
Un meccanismo aggiuntivo é necessario

MAC

I *codici per l'autenticazione dei messaggi* (*message authentication code*)

- permettono di identificare ed autenticare il mittente
- permettono di controllare l'integritá del messaggio

Entrambe le parti devono condividere un segreto che Adv non conosce.



MAC

Definizione 4.1. Un codice per l'autenticazione di messaggi (MAC, in breve) consiste di tre algoritmi ppt $(Gen, Mac, Vrfy)$ dove

1. $k \leftarrow Gen(1^n), \quad |k| \geq n$, genera le chiavi
2. $t \leftarrow Mac_k(m), \quad m \in \{0,1\}^*$, dove t é il *tag* associato ad m
3. $b := Vrfy_k(m,t), \quad b \in \{0,1\}$

tali che, per ogni n, per ogni k dato in output da $Gen(1^n)$ e per ogni $m \in \{0,1\}^*$

$$Vrfy_k(m, Mac_k(m)) = 1.$$

Se esiste una funzione $\ell(n)$ e lo schema é definito solo per $m \in \{0,1\}^{\ell(n)}$, allora parleremo di MAC a lunghezza fissa per messaggi di lunghezza $\ell(n)$.

MAC

Solitamente $k \leftarrow Gen(1^n)$ é uniforme.

Una classe importante é quella degli schemi deterministici.

- $Mac_k(\cdot)$ é deterministica
- La verifica si dice *canonica*

$$\Downarrow$$

Data la coppia (m, t), ricalcola $Mac_k(m) = t'$ e verifica che $t' = t$

In generale, usare una funzione $Vrfy(\cdot)$ é utile per distinguere le semantiche di due operazioni diverse: autenticazione e verifica.

Sicurezza dei codici di autenticazione di messaggi

"Nessun Adv PPT dovrebbe essere in grado di generare un tag su qualsiasi messaggio nuovo, che non sia stato autenticato ed inviato in precedenza da una delle parti."

Potere Adv

- Adv: algoritmo PPT, puó influenzare i contenuti da autenticare
- Useremo un oracolo $Mac_k(\cdot)$: Adv puó chiedere tag di messaggi m a propria scelta

Cos'é una rottura di un MAC

- Adv dá in output (m, t) tale che:
  1. il tag t é un tag valido su m, i.e., $Vrfy(m, t) = 1$
  2. Adv non ha ottenuto t per m con una precedente richiesta all'oracolo

La 1. implica che m viene accettato come valido. La 2. che Adv ha generato t da solo

Sicurezza dei codici di autenticazione di messaggi

Osservazioni:

- Schemi MAC che non permettono ad Adv di avere successo con probabilitá piú che trascurabile sono detti *non falsificabili esistenzialmente rispetto ad un attacco adattivo di tipo chosen-message* (existentially unforgeable under an adaptive chosen-message attack)
- Gli schemi MAC *non* considerano gli *attacchi di replay*, che vanno affrontati con altri mezzi.

Codici non falsificabili

Sia $\Pi = (Gen, Mac, Vrfy)$, A un Adv generico ed n il parametro di sicurezza

$Mac - forge_{A,\Pi}(n)$

1. Il challenger genera $k \leftarrow Gen(1^n)$
2. $A^{Mac_k(\cdot)}(1^n)$ invia all'oracolo un certo numero di query. Sia Q l'insieme delle query richieste. Alla fine $A^{Mac_k(\cdot)}(1^n)$ dá in output (m, t).
3. $A^{Mac_k(\cdot)}(1^n)$ ha successo se e solo se
   - $Vrfy_k(m, t) = 1$
   - $m \notin Q$

   In questo caso, l'output dell'esperimento é 1; altrimenti é 0.

Codici non falsificabili

Definizione 4.2. Un codice per l'autenticazione di messaggi $\Pi = (Gen, Mac, Vrfy)$ é non falsificabile esistenzialmente rispetto ad un attacco adattivo di tipo chosen-message (existentially unforgeable under an adaptive chosen-message attack) se, per ogni A PPT, esiste una funzione trascurabile negl, tale che:

$$Pr[Mac - forge_{A,\Pi}(n) = 1] \leq negl(n).$$

É la definizione troppo forte?

- Non falsificabile esistenzialmente $\Rightarrow$ anche un messaggio *senza significato* non é autenticabile da Adv
- Rispetto ad attacchi chosen-message $\Rightarrow$ Adv nella vita reale potrebbe non avere la possibilitá di scegliere ...

Avendo una definizione piú forte possibile, non dobbiamo preoccuparci della specifica applicazione.

Mac forti

Definizione 4.2. Un codice per l'autenticazione di messaggi $\Pi = (Gen, Mac, Vrfy)$ é fortemente sicuro (o é un Mac forte) se, per ogni A PPT, esiste una funzione trascurabile negl, tale che:

$$Pr[Mac - sforge_{A,\Pi}(n) = 1] \leq negl(n),$$

dove $Mac - sforge_{A,\Pi}(n)$ é come $Mac - forge_{A,\Pi}(n)$ ma l'insieme delle query Q contiene coppie (m, t).

Nota che:

- In $Mac - forge_{A,\Pi}(n)$: A produce (m, t') con $m \in Q \quad \Rightarrow$ non é un successo per A
- In $Mac - sforge_{A,\Pi}(n)$: A produce (m, t') con $(m, t) \in Q \quad \Rightarrow$ é un successo per A

Timing Attack

Un potenziale attacco basato sul calcolo del tempo

Adv invia coppie (m, t) ad un ricevitore, che funziona come un oracolo di verifica, e acquisisce info tipo

- se il ricevitore accetta/rifiuta
- il tempo che spende per decidere

Adv puó calcolare un tag corretto per un *nuovo* messaggio.

Per esempio, supponiamo che uno schema Mac usi la verifica canonica.

Per verificare t su m, il ricevitore

- calcola $t' := Mac_k(m)$ e confronta $t' = t$,
- dá in output 1 (uguale) o 0 (diversi).

Timing Attack

Supponiamo che il confronto $t' = t$ sia implementato con la funzione C *strcmp*.

strcmp confronta un byte alla volta e rifiuta appena trova due byte che differiscono.

Il tempo di rifiuto dipende dalla posizione del primo byte in cui t' e t differiscono.

Supponiamo Adv conosca i primi i byte di t per m.

$\Rightarrow$ Adv puó calcolare il prossimo byte del tag corretto t per m inviando al ricevitore

$$(m, t_0), \quad (m, t_1), \quad , \ldots , \quad (m, t_{255})$$

dove in t_j

- i primi i byte sono corretti (quelli che Adv conosce)
- il byte $i + 1$ é uguale a j
- i byte rimanenti sono uguali a 0

Timing Attack

Tutti gli elementi della sequenza saranno rifiutati ma la coppia (m, t_j) con il j *giusto* richiederá piú tempo!

Applicando l'attacco a partire da $i = 0$ e ripetendo per ogni byte, Adv calcola il tag corretto t.

Adv ha bisogno al piú di 256 query per calcolare ciascun byte

$$\Downarrow$$

Per un tag di 16 byte, 4096 query sono sufficienti per il calcolo dell'intero tag

$$\Downarrow$$

L'attacco é efficiente

Side-channel attack

L'attacco descritto é un attacco di tipo *side-channel* ed é di tipo fisico.

Attacchi come il precedente che possono essere sferrati nel mondo reale *non* sono modellati dalle definizioni usuali.

Il precedente é stato applicato alla playstation Xbox 360 qualche anno fa.

Conclusione: la procedura di verifica del Mac dovrebbe essere implementata usando una funzione time-independent, che confronta *sempre tutti* i byte delle stringhe.

Costruzioni di codici sicuri

Le funzioni pseudocasuali sono uno strumento naturale per la costruzione di codici per l'autenticazione dei messaggi

Intuitivamente $t = F_k(m) \Rightarrow$ probabilitá di individuare il valore é $\approx 1/2^n$

CONSTRUCTION 4.5

Let F be a pseudorandom function. Define a fixed-length MAC for messages of length n as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, output the tag $t := F_k(m)$. (If $|m| \neq |k|$ then output nothing.)
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^n$, and a tag $t \in \{0,1\}^n$, output 1 if and only if $t \stackrel{?}{=} F_k(m)$. (If $|m| \neq |k|$, then output 0.)

Lunghezza arbitraria

Come possiamo estendere la costruzione a messaggi *di lunghezza arbitraria*?

Sia $\Pi' = (Mac', Vrfy')$ un mac sicuro a lunghezza fissa per messaggi di lunghezza n.

Sia $m = m_1 m_2 \dots m_d$, messaggio di d blocchi di lunghezza n

Come usare Π' per costruire Π? Alcune idee

1. $m_1 m_2 \dots m_d \to t_1 t_2 \dots t_d$, dove $t_i = Mac'_k(m_i)$
   - previene l'invio di blocchi non autenticati
   - non previene attacchi di riordino $m_2 m_1 \to t_2 t_1$
2. per evitare il riordino, usiamo un indice di blocco
   - non previene attacchi "per troncamento" (A butta via la fine del msg)

Lunghezza arbitraria

3. aggiungiamo in ogni blocco la lunghezza del messaggio

$$t_i = Mac'_k(\ell||i||m_i)$$

   - non previene attacchi del tipo "mix-and-match"

$$m_1 \dots m_d \to t_1 \dots t_d \qquad \text{e} \qquad m'_1 \dots m'_d \to t'_1 \dots t'_d$$

$$\downarrow$$

$$m_1 m'_2 \dots m_d \to t_1 t'_2 \dots t_d$$

4. aggiungiamo un "identificatore casuale del messaggio" in ogni blocco.

CBC-MAC

Abbiamo visto che usando funzioni pseudocasuali che prendono input di lunghezza fissa n, é possibile costruire codici di autenticazione per messaggi di lunghezza arbitraria

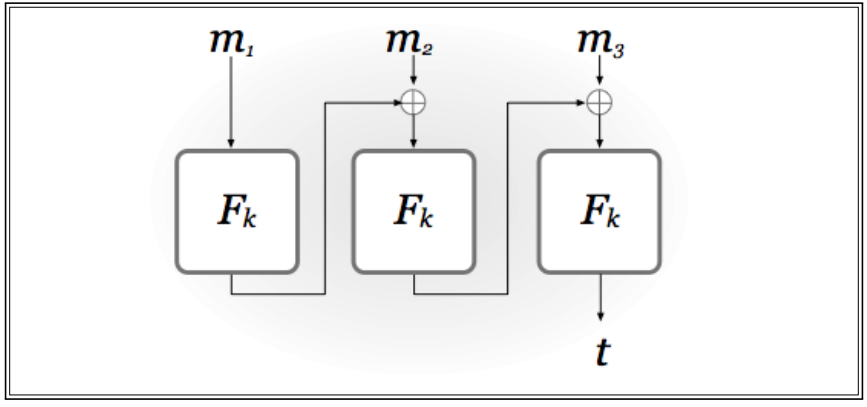
La costruzione **non** é efficiente.

$$m \quad : \quad |m| = dn \qquad \Rightarrow \qquad \text{tag } t \quad : \quad |t| = 4dn$$

Inoltre, F_k viene valutata $4d$ volte.

Possiamo far meglio usando CBC-MAC, un codice per l'autenticazione di messaggi standardizzato.

CBC-MAC



CBC-MAC

CONSTRUCTION 4.11

Let F be a pseudorandom function, and fix a length function $\ell > 0$. The basic CBC-MAC construction is as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message m of length $\ell(n) \cdot n$, do the following (we set $\ell = \ell(n)$ in what follows):

  1. Parse m as $m = m_1, \ldots, m_\ell$ where each m_i is of length n.
  2. Set $t_0 := 0^n$. Then, for $i = 1$ to ℓ:
     Set $t_i := F_k(t_{i-1} \oplus m_i)$.

  Output t_ℓ as the tag.

- Vrfy: on input a key $k \in \{0,1\}^n$, a message m, and a tag t, do: If m is not of length $\ell(n) \cdot n$ then output 0. Otherwise, output 1 if and only if $t \stackrel{?}{=} \mathsf{Mac}_k(m)$.

CBC-MAC

Teorema 4.12 Sia $\ell(\cdot)$ un polinomio. Se F é una PRF, allora la Costruzione 4.11 é un Mac sicuro per messaggi di lunghezza $\ell(n) \cdot n$.

Attenzione: sicura **soltanto** quando la lunghezza dei messaggi é fissata e convenuta in precedenza.

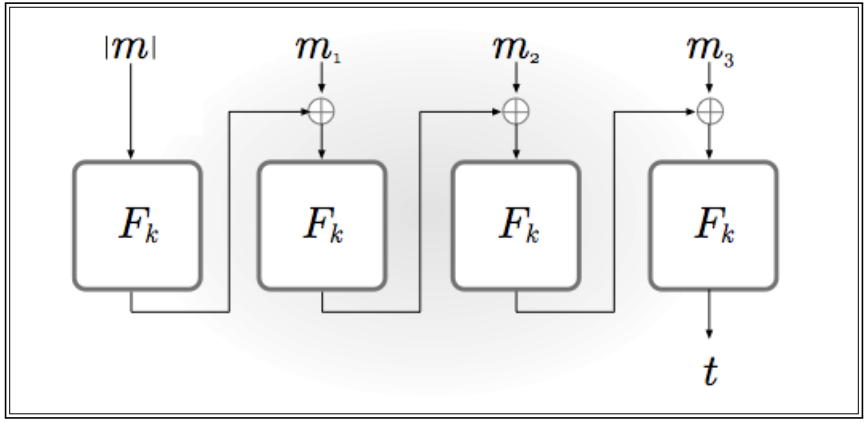
Vantaggi: fornisce un Mac a lunghezza fissa molto piú efficiente rispetto alla costruzione precedente.

Confronto con la modalitá di cifratura CBC

1. CBC-MAC **non** usa un vettore di inizializzazione IV. Questo aspetto é cruciale. CBC-MAC che usa un IV casuale **non** é sicuro.

2. La modalitá di cifratura CBC dá in output **tutti** i valori t_i intermedi. CBC-MAC solo t_ℓ. Se viene modificata per dare in output tutti gli $\{t_i\}_i$, CBC-MAC **non** é piú sicuro.

CBC-MAC



Attacchi di tipo chosen ciphertext

L'Adv ha anche l'abilitá di ottenere i messaggi in chiaro corrispondenti a cifrati di propria scelta. Modelliamo questa abilitá dando ad Adv accesso ad un oracolo per la decifratura

$PrivK^{cca}_{A,\Pi}(n)$

1. Il Challenger sceglie $k \leftarrow Gen(1^n)$
2. A riceve in input 1^n ed ha accesso agli oracoli $Enc_k(\cdot)$ e $Dec_k(\cdot)$. Dá in output m_0 ed m_1 della stessa lunghezza
3. Il Challenger sceglie $b \leftarrow \{0,1\}$ e calcola $c^* \leftarrow Enc_k(m_b)$
4. $A(1^n)$ riceve c^*, continua ad accedere agli oracoli $Enc_k(\cdot)$ e $Dec_k(\cdot)$. Non puó chiedere la decifratura di c^*. Alla fine dá in output $b' \in \{0,1\}$.
5. Se $b = b'$, l'output dell'esperimento é 1 ($A(1^n)$ vince); altrimenti, 0.

Definizione

Definizione 3.33. Uno schema di cifratura a chiave privata $\Pi = (Gen, Enc, Dec)$ ha cifrature indistinguibili rispetto ad un attacco di tipo chosen ciphertext (CCA-sicuro) se, per ogni Adv A PPT, esiste una funzione trascurabile $negl$ tale che:

$$Pr[PrivK^{cca}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n),$$

dove la probabilitá é calcolata su

- randomness usata da A
- randomness usata nell'esperimento
  - scelta della chiave
  - scelta del bit b
  - random bit usati da $Enc_k(\cdot)$

CCA: Attacchi plausibili

Si puó dimostrare che:

Sicurezza CCA per un singolo messaggio $\Rightarrow$ Sicurezza CCA per messaggi multipli

Sono gli attacchi CCA realistici?

1. US NAVY 1942: "manda messaggi cifrati ed osserva il comportamento del nemico"
2. Server on-line: "manda messaggi cifrati ed osserva la reazione"
   e.g., servizio rifiutato $\Rightarrow$ cifrato scorretto
3. Protocolli crittografici: le parti oneste in alcuni casi possono inconsapevolmente agire come oracoli di decifratura

Schemi di cifratura introdotti e attacchi CCA

Tutti gli schemi che abbiamo studiato sono insicuri rispetto ad attacchi di tipo CCA

$$Enc_k(m) =< r, F_k(r) \oplus m >$$

$$F \text{ Pseudocasuale} \Rightarrow \text{CPA-sicuro}$$

L'Adv A che segue, di tipo CCA, é in grado di distinguere le cifrature con prob. 1.

- A sceglie $m_0 = 0^\ell$ ed $m_1 = 1^\ell$.
- riceve il cifrato $c =< r, s >$
- inverte il primo bit di s e chiede all'oracolo $Dec_k(\cdot)$ di decifrare $c' =< r, s' >$ (nota che $c' =< r, s' > \quad \neq \quad c =< r, s >$, quindi la query é lecita)
- Se $Dec_k(c') = 10^{\ell-1}$ allora $b' = 0$; altrimenti ($Dec_k(c') = 01^{\ell-1}$), $b' = 1$
- Dá in output b'.

Schemi di cifratura introdotti e attacchi CCA

Osservazione: ogni schema di cifratura in cui i cifrati possono essere "manipolati" in modo controllato *non* possono essere CCA-sicuri.

$$\text{Sicurezza CCA} \Rightarrow \text{"Non malleabilitá"}$$

Non malleabilitá = Informalmente significa che un cifrato modificato, quando decifrato, comporta

- la restituzione di un messaggio di errore $\perp$, oppure
- la restituzione di un messaggio m' *scorrelato* dal messaggio m originariamente cifrato

Costruire schemi di cifratura CCA-sicuri richiede strumenti aggiuntivi.

Padding-Oracle attack

Un attacco CCA utilizzato contro protocolli reali

L'attacco richiede soltanto l'abilitá di determinare se un cifrato modificato viene decifrato correttamente oppure no

Nella cifratura CBC un msg deve avere lunghezza multipla della lunghezza del blocco. Pertanto, uno *schema di padding* (completamento) viene usato per ottenere il requisito.

Sia L la lunghezza del blocco.

Sia b il numero di byte che devono essere aggiunti al messaggio. Risulta $1 \leq b \leq L$.

Se il messaggio é giá "multiplo" di L, allora $b = L$

Strategia di padding: aggiungiamo al messaggio la stringa contenente b (un byte) esattamente b volte.

Padding-Oracle attack

1 byte necessario $\Rightarrow$ aggiungiamo il byte con valore $0x01$ (rappr. esadecimale)

2 byte necessari $\Rightarrow$ aggiungiamo la stringa $0x0202$ (rappr. esadecimale)

4 byte necessari $\Rightarrow$ aggiungiamo la stringa $0x04040404$ (rappr. esadecimale)

...

In fase di decifratura, il pad viene controllato e rimosso, prima di restituire il plaintext.

Se il pad *non* é corretto $\Rightarrow$ viene generato e restituito un errore $\perp$

$\Downarrow$

Usando questa informazione, Adv riesce a recuperare *completamente* il plaintext

Padding-Oracle attack

Consideriamo l'attacco su un cifrato di 3 blocchi

$$c =< c_0, c_1, c_2 >=< IV, c_1, c_2 >$$

Avendo applicato il CBC-mode e lo schema di padding descritto,

$$m_2 = F_k^{-1}(c_2) \oplus c_1 \qquad \text{dove } m_2 \text{ termina con } 0xbb\ldots b \ (b \text{ volte})$$

Sia c_1' identico a c_1 a parte una modifica nel byte finale. Nota che:

$$c' =< IV, c_1', c_2 > \qquad \Rightarrow \qquad m_2' = F_k^{-1}(c_2) \oplus c_1'$$

Risulta m_2' uguale ad m_2 eccetto che per una modifica nel byte finale.

Padding-Oracle attack

Lo stesso ragionamento vale per qualsiasi byte scelto, i.e., se modifichiamo l'i-esimo byte di c_1, allora m_2' differisce da m_2 nell'i-esimo byte soltanto.

In generale,

$$c_1' = c_1 \oplus \Delta \qquad \Rightarrow \qquad m_2' = m_2 \oplus \Delta$$

Come possiamo usare questa relazione per calcolare il valore di b?

Un Adv A puó agire come segue:

A modifica il *primo* byte di c_1 e chiede la decifratura di $c' =< IV, c_1^1, c_2 >$

Se la decifratura

- *fallisce* significa che, verificando il pad, l'algoritmo di decifratura é arrivato al primo byte e l'ha trovato scorretto $\Rightarrow b = L$
- *non fallisce* significa che il primo byte é del messaggio (non del pad) $\Rightarrow b < L$

Padding-Oracle attack

A ripete l'attacco modificando il *secondo* byte di c_1 e chiedendo la decifratura di $c'' =< IV, c_1^2, c_2 >$

L'attacco viene ripetuto al piú L volte ed alla fine A ottiene il valore di b.

Disponendo di b, A puó calcolare i byte del messaggio uno dopo l'altro al modo seguente:

A sa che m_2 termina con $0xB0xb \dots b$ dove

- B é il byte sconosciuto del messaggio in chiaro che A vuole conoscere
- $0xb \dots b$ é il pad, con il byte b ripetuto b volte

A definisce, per ogni i tale che $0 \leq i < 2^8$,

$$\Delta_i \stackrel{def}{=} \begin{array}{cccc} 0x00 \dots 0 & 0x\mathbf{i} & 0x(b+1) \dots & (b+1) \\ \oplus\, 0x00 \dots 0 & 0x0 & 0xb \quad \dots & b \end{array}$$

Padding-Oracle attack

Si noti che gli ultimi $b+1$ byte di Δ_i contengono

$$\text{l'intero } i \text{ seguito da } (b+1) \oplus b \ldots (b+1) \oplus b$$

A questo punto, quando A chiede la decifratura di

$$c_i = < IV, c_1 \oplus \Delta_i, c_2 >$$

il messaggio sottostante al cifrato é

$$0x(B \oplus i)0x(b+1)\ldots(b+1).$$

Nota che il pad $0xb \ldots b$ é stato sostituito con $0x(b+1)\ldots(b+1)$.

Padding-Oracle attack

Pertanto, se la decifratura é corretta

$$0x(B \oplus i) = 0x(b+1) \qquad \Rightarrow \qquad B = (b+1) \oplus i.$$

A ha quindi calcolato il byte B del messaggio!

L'attacco viene ripetuto per $i = 0, \ldots, 2^8 - 1$, fino a quando la decifratura non risulta corretta e B viene calcolato

L'attacco puó essere facilmente esteso per recuperare gli altri byte del messaggio in chiaro.

Obiettivo: cifratura autenticata

Abbiamo visto che:

- schemi di cifratura $\Rightarrow$ confidenzialitá/riservatezza
- codici per l'autenticazione di messaggi $\Rightarrow$ integritá/autenticitá

Vorremmo ottenere entrambi gli obiettivi, confidenzialitá ed autenticitá, insieme.

Sia $\Pi = (Gen, Enc, Dec)$ uno schema di cifratura a chiave privata.

- la nozione di segretezza a cui siamo interessati é rispetto ad attacchi CCA
- la nozione di integritá a cui siamo interessati é "non falsificabile esistenzialmente rispetto ad attacchi di tipo chosen message"

Purtroppo Π non soddisfa la sintassi di un codice per l'autenticazione di messaggi. Pertanto, occorre una definizione specifica.

Cifratura non falsificabile

Sia $\Pi = (Gen, Enc, Dec)$, A un avversario, n parametro di sicurezza.

The unforgeable encryption experiment $\mathsf{Enc\text{-}Forge}_{\mathcal{A},\Pi}(n)$**:**

1. *Run* $\mathsf{Gen}(1^n)$ *to obtain a key* k.
2. *The adversary* $\mathcal{A}$ *is given input* 1^n *and access to an encryption oracle* $\mathsf{Enc}_k(\cdot)$. *The adversary outputs a ciphertext* c.
3. *Let* $m := \mathsf{Dec}_k(c)$, *and let* $\mathcal{Q}$ *denote the set of all queries that* $\mathcal{A}$ *asked its encryption oracle. The output of the experiment is* 1 *if and only if (1)* $m \neq \perp$ *and (2)* $m \notin \mathcal{Q}$.

Cifratura non falsificabile

Definizione 4.16. Uno schema di cifratura a chiave privata Π é non falsificabile (unforgeable) se, per ogni avversario A ppt, esiste una funzione trascurabile *negl* tale che

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

Nota: possiamo considerare una definizione piú forte in cui A ha anche accesso all'oracolo di decifratura. La costruzione proposta nel seguito la soddisfa.

Definizione 4.17. Uno schema di cifratura a chiave privata Π é uno schema di cifratura autenticata se é CCA-sicuro e non falsificabile.

Come possiamo costruire schemi di cifratura autenticata?

Cifratura autenticata

Ogni generica combinazione di uno schema di cifratura e di un codice per l'autenticazione di messaggi sicuri funziona?

... purtroppo no!

Siano:

- $\Pi_E = (Gen, Enc, Dec) \rightarrow$ schema di cifratura CPA-sicuro
- $\Pi_M = (Mac, Vrfy) \rightarrow$ codice per l'autenticazione di messaggi sicuro

Esistono tre combinazioni naturali:

1. **Cifra ed autentica**. Il mittente trasmette il cifrato $< c, t >$ dove

$$c \leftarrow Enc_{k_E}(m) \quad \text{e} \quad t = Mac_{k_M}(m)$$

Cifratura autenticata

Il ricevente decifra c per recuperare m. Poi verifica il tag t. Restituisce m se non si verificano errori, $\perp$ altrimenti.

2. **Autentica e poi cifra**. Il mittente trasmette il cifrato $< c >$ calcolato come

$$t = Mac_{k_M}(m) \quad \text{e} \quad c \leftarrow Enc_{k_E}(m||t)$$

Il ricevente decifra c per recuperare $m||t$. Poi verifica il tag t. Restituisce m se non si verificano errori, $\perp$ altrimenti.

3. **Cifra e poi autentica**. Il mittente trasmette il cifrato $< c, t >$ calcolato come

$$c \leftarrow Enc_{k_E}(m) \quad \text{e} \quad t = Mac_{k_M}(c)$$

Cifratura autenticata

CONSTRUCTION 4.18

Let $\Pi_E = (\mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme and let $\Pi_M = (\mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code, where in each case key generation is done by simply choosing a uniform n-bit key. Define a private-key encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

- Gen': on input 1^n, choose independent, uniform $k_E, k_M \in \{0,1\}^n$ and output the key (k_E, k_M).
- Enc': on input a key (k_E, k_M) and a plaintext message m, compute $c \leftarrow \mathsf{Enc}_{k_E}(m)$ and $t \leftarrow \mathsf{Mac}_{k_M}(c)$. Output the ciphertext $\langle c, t \rangle$.
- Dec': on input a key (k_E, k_M) and a ciphertext $\langle c, t \rangle$, first check whether $\mathsf{Vrfy}_{k_M}(c, t) \stackrel{?}{=} 1$. If yes, then output $\mathsf{Dec}_{k_E}(c)$; if no, then output $\perp$.

Cifratura autenticata

La sicurezza forte del Mac implica due cose:

1. A non puó produrre t' su qualsiasi nuovo c'
   - $\Rightarrow (Gen', Enc', Dec')$ é non falsificabile
2. A non puó produrre $< c, t >$ di cui chiedere la decifratura all'oracolo in $Priv^{cca}_{A,\Pi}$
   - $\Rightarrow$ l'oracolo di decifratura $Dec'(\cdot)$ é inutile
   - $\Rightarrow A$ agisce esattamente come in $Priv^{cpa}_{A,\Pi}$

Formalmente, vale il seguente:

Teorema 4.19. Sia Π_E uno schema di cifratura CPA-sicuro a chiave privata e sia Π_M un codice di autenticazione di messaggi fortemente sicuro. Allora la Costruzione 4.18 produce uno schema di cifratura autenticata.

Necessitá di chiavi indipendenti

Homework

"Istanze differenti di primitive crittografiche dovrebbero sempre usare chiavi indipendenti"

Consideriamo un esempio in cui applichiamo l'approccio "cifra e poi autentica" in cui la **stessa** chiave viene usata

$$\text{Sia } F \text{ una PRP forte} \quad \Rightarrow \quad F^{-1} \text{ é una PRF forte}$$

Si consideri lo schema definito da

- $F_k : \{0,1\}^n \to \{0,1\}^n$
- $Enc_k(m) = F_k(m||r)$, dove $m \leftarrow \{0,1\}^{n/2}, r \leftarrow \{0,1\}^{n/2}$
- $Mac_k(c) = F_k^{-1}(c)$

Necessitá di chiavi indipendenti

Homework

É possibile mostrare che:

1. lo schema di cifratura é CPA-sicuro
2. il Mac é fortemente sicuro
3. la combinazione usando la stessa chiave k **non** lo é!

Infatti,

$$[Enc_k(m) = F_k(m||r), Mac_k(Enc_k(m)) = F_k^{-1}(F_k(m||r))] \quad = \quad [F_k(m||r), m||r]$$

Nella costruzione 4.18 si richiede che le chiavi siano scelte indipendentemente l'una dall'altra ed in modo uniforme.

Sessioni di comunicazione sicure

Homework

"Un periodo di tempo durante il quale le parti che comunicano mantengono uno stato e desiderano comunicare in modo sicuro"

Uno schema di cifratura autenticata da solo **non** é sufficiente.

- **Attacco di riordino.**
  - *Alice* trasmette c_1, c_2, Adv intercetta e trasmette c_2, c_1
- **Attacco di replay.**
  - Adv invia piú volte c
- **Attacco di riflessione.**
  - Adv invia ad *Alice* un cifrato c che in precedenza *Alice* ha inviato a *Bob*

Soluzione

Homework

Bit di direzionalitá: nelle comunicazioni, il bit $b_{X,Y}$ indica il verso, da X verso Y

Contatori: il valore $ctr_{X,Y}$ conta i messaggi che X ha inviato a Y.

- $A \rightarrow B \qquad c \leftarrow Enc_k(b_{A,B}||ctr_{A,B}||m)$
- $B \rightarrow A \qquad c \leftarrow Enc_k(b_{B,A}||ctr_{B,A}||m)$
- $\text{Stato}_A = \text{Stato}_B = (ctr_{A,B}, ctr_{B,A})$

Cifratura CCA-sicura

Uno schema di cifratura autenticata é CCA-sicuro

Esistono schemi di cifratura a chiave privata che risultano CCA-sicuri ma non infalsificabili?

Sí!

$$F_k\{0,1\}^n \leftarrow \{0,1\}^n$$

$$Enc_k(m) = F_k(m||r), \quad m \in \{0,1\}^{\frac{n}{2}}, \quad r \leftarrow \{0,1\}^{\frac{n}{2}}$$

Sono utili? In alcune applicazioni in cui l'integritá non é una preoccupazione, potrebbero.

In realtá, poiché con le costruzioni attualmente note non c' é alcun guadagno in termini di efficienza, conviene usare uno schema di cifratura autenticata.

Da un punto di visto concettuale é peró utile distinguere.

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

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

$h(x)$, 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

Nelle strutture dati	In Crittografia
Evitare collisioni é un **desiderio** al fine di ottenere efficienza maggiore	Evitare collisioni é un **requisito** fondamentale
L'insieme dei dati é quasi sempre indipendente dalla funzione hash e **non ha** il fine di produrre collisioni	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 $H : D \to R$ tali che $|D| > |R|$.

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

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

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:

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

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 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

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 in dominio per input di lunghezza arbitraria

Senza perdita di generalitá, supponiamo che:

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

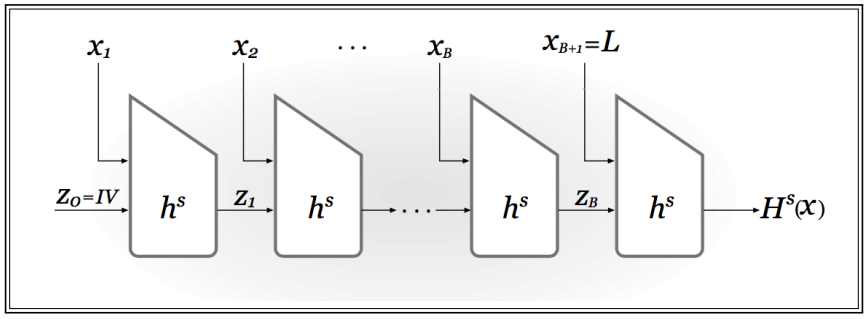
$$\Downarrow$$

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

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

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.

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

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

Ma $H^s(x) = H^s(x') \quad \Rightarrow \quad z_{B+1} = z'_{B+1} \quad \Rightarrow \quad w = z_B || x_{B+1}$ e $w' = z'_B || x'_{B+1}$ 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 due modi

- Costruzione generica
- CBC-Mac

Una terza 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 riesce a produrre una falsificazione per $m^* \notin Q$.

Ci sono due casi:

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. $\forall m \in Q$, $H^s(m) \neq H^s(m^*)$. Sia $H^s(Q) = \{H^s(m) \mid m \in Q\}$. $H^s(m^*) \notin H^s(Q)$.
   $\Rightarrow A$ ha trovato un tag valido rispetto a Π per il nuovo messaggio $H^s(m^*)$.

HMAC

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

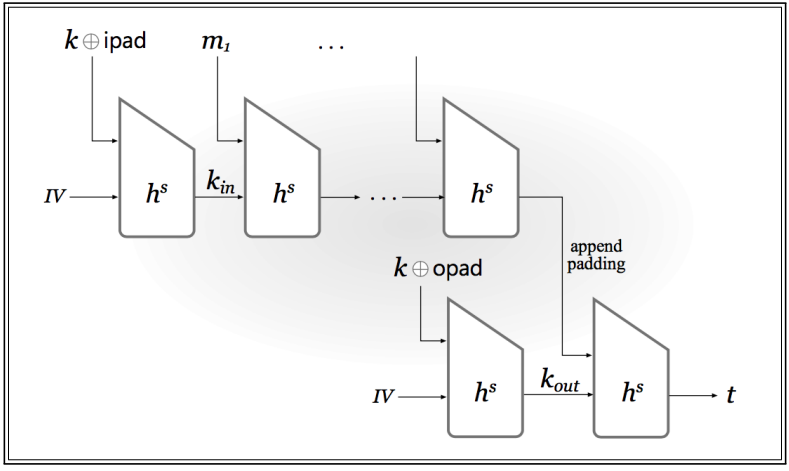
$$Mac_k(m) = H(k||m) \qquad \text{non é una buona scelta}$$

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

Idea: usare "due livelli" di hash

- 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

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

corrisponde alla funzione $h : \{0,1\}^{2n} \to \{0,1\}^n$ 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)

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

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

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

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.

Precisamente:

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

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.

La discussione suggerisce che due chiavi indipendenti -interna ed esterna - 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 $\widetilde{Mac}_{k_{out}}(\cdot)$ 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.

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)

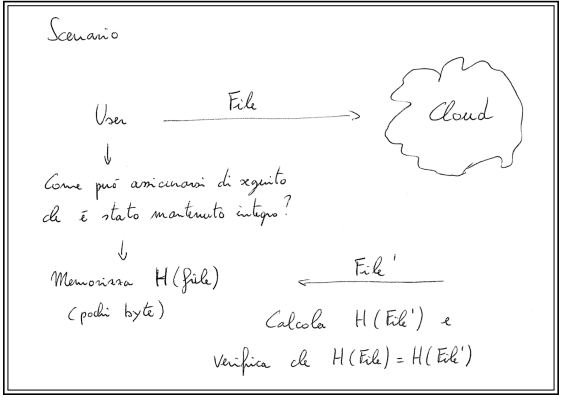
  Idea applicata in:

  - **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

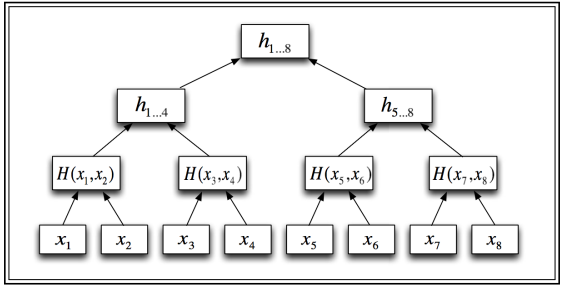


Merkle Tree

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 $h_{1\ldots8}$. 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 $h_{1...8}$ e.g., per verificare x_3,

l'utente riceve $x_4, h_{1,2}$, e $h_{5...8}$.

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.

Derivazione delle chiavi

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

Le parti possono tentatare 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

Intuitivamente: 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

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

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:

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

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

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

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

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

Homework

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

Homework

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

Homework

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

Homework

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

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