Autenticazione

Paolo D'Arco
pdarco@unisa.it

Universitá di Salerno

Elementi di Crittografia

Contenuti

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?

Lo stesso attacco appena descritto si applica alle modalitá OFB e CTR

$$\Downarrow$$

Anche usando cifrature CPA-sicure non é sufficiente per prevenire alterazioni dei messaggi. E circa ECB e CBC?

- ECB: modifiche anche di un solo bit del cifrato influenzano ancora il messaggio in chiaro, anche se la posizione *non* é controllabile
- CBC: cambiando il j-esimo bit di IV si cambia il j-esimo bit del primo blocco del messaggio

$$m_1 := F_k^{-1}(c_1) \oplus IV$$

Tutti gli altri blocchi restano inalterati, ma il primo contiene informazioni importanti.

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

$\Downarrow$

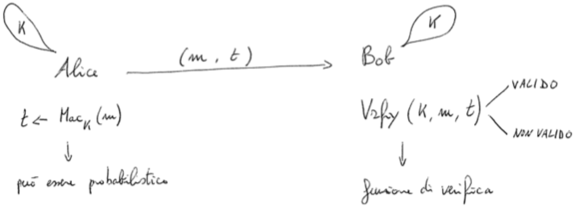
Un meccanismo aggiuntivo é necessario

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

Sicurezza dei codici di autenticazione di messaggi

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

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)

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

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

1. Il challenger genera $k \leftarrow Gen(1^n)$ e setta l'oracolo $Mac_k(\cdot)$
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.

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:

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

Codici non falsificabili

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.

Attacchi di tipo replay

Sono una minaccia reale ma *non* sono modellati dalla definizione.

Due soluzioni comuni:

- usare numeri di sequenza nei messaggi
- marche temporali (timestamp) che certificano il tempo

Nota: rispettando la definizione Adv puó produrre un *nuovo* Mac su un messaggio per cui un Mac é giá stato calcolato. Cioé Adv

$$\text{dato } (m, t) \qquad \text{produce } (m, t') \qquad \text{dove } t \neq t'.$$

In alcune applicazioni puó essere un problema

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:

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

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

Nota che:

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

Mac forti

Proposizione 4.4. Se $\Pi = (Gen, Mac, Vrfy)$ é un Mac sicuro che usa la verifica canonica, allora Π é anche fortemente sicuro.

Molti Mac usati nelle applicazioni reali usano la verifica canonica.

Osservazione: la definizione fornisce ad Adv accesso all'oracolo per ottenere Mac di messaggi. Perché non diamo ad Adv anche accesso ad un oracolo di verifica?

Per Mac con verifica canonica, non c'é differenza: qualsiasi Mac che soddisfa la Definizione 4.2 soddisferebbe anche una definizione con un oracolo di verifica

Per Mac con verifica generica potrebbe essere diverso.

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*.

La funzione *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.

Side-channel attack

Un attacco di tipo side-channel usa l'osservazione fisica dell'hardware o l'analisi dei messaggi di errore per carpire informazioni circa bit segreti. L'attacco puó

- sfruttare variazioni temporali, il consumo di energia, o la radiazione elettromagnetica che un dispositivo genera durante l'esecuzione di un algoritmo
- misurare le risposte agli errori di formattazione o ai guasti indotti
- sfruttare informazione rilasciata dalla computazione circa i pattern di accesso della CPU alla memoria cache
- sfruttare bug o peculiaritá del linguaggio di programmazione usato
- una combinazione dei differenti tipi di informazione rilasciata

Costruzioni sicure di codici

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.)

Teorema 4.6. Se F é una funzione pseudocasuale, allora la Costruzione 4.5 realizza un Mac sicuro di lunghezza fissata per messaggi di lunghezza n.

Dim. Sia A ppt e sia $\widetilde{\Pi} = (\widetilde{Gen}, \widetilde{Mac}, \widetilde{Vrfy})$ (ipotetico, $f \in Func_n$ invece di F_k).

Risulta

$$Pr[\text{Mac-forge}_{A,\widetilde{\Pi}}(n) = 1] = 1/2^n.$$

Infatti, per ogni $m \notin Q$, il valore $t = f(m)$ é uniformemente distribuito in $\{0,1\}^n$ dal punto di vista di A. Mostreremo che $\exists$ una funzione $negl()$ tale che:

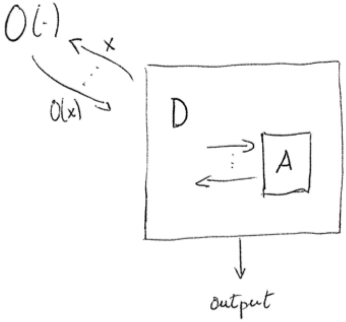
$$|Pr[\text{Mac-forge}_{A,\widetilde{\Pi}}(n) = 1] - Pr[\text{Mac-forge}_{A,\Pi}(n) = 1]| \leq negl(n)$$

che, combinato con il risultato precendente, implica

$$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \leq 1/2^n + negl(n), \qquad \text{provando il teorema.}$$

Per provare l'affermazione, costruiremo un distinguisher D con accesso oracolare ad una funzione, il cui obiettivo é stabilire se la funzione é casuale o pseudocasuale.

D emula l'esperimento Mac-forge$_{A,?}(n)$ e osserva se A vince o perde.



A pensa di star eseguendo l'esperimento reale. In realtà è D che lo sta "emulando".

Distinguisher D:
D is given input 1^n and access to an oracle $\mathcal{O} : \{0,1\}^n \to \{0,1\}^n$, and works as follows:

1. Run $\mathcal{A}(1^n)$. Whenever $\mathcal{A}$ queries its MAC oracle on a message m (i.e., whenever $\mathcal{A}$ requests a tag on a message m), answer this query in the following way:

   Query $\mathcal{O}$ with m and obtain response t; return t to $\mathcal{A}$.

2. When $\mathcal{A}$ outputs (m, t) at the end of its execution, do:
   (a) Query $\mathcal{O}$ with m and obtain response $\hat{t}$.
   (b) If (1) $\hat{t} = t$ and (2) $\mathcal{A}$ never queried its MAC oracle on m, then output 1; otherwise, output 0.

D é ppt se A é ppt. Inoltre:

- Se $O(\cdot)$ é pseudocasuale
  $\Rightarrow$ la vista di A che esegue come subroutine di D é identica a quella che avrebbe nell'esperimento Mac-forge$_{A,\Pi}(n)$
- Se $O(\cdot)$ é casuale
  $\Rightarrow$ la vista di A che esegue come subroutine di D é identica a quella che avrebbe nell'esperimento Mac-forge$_{A,\tilde{\Pi}}(n)$

Pertanto, si ha:

$$Pr[D^{F_k(\cdot)}(n) = 1] = Pr[\text{Mac-forge}_{A,\Pi}(n) = 1]$$

e

$$Pr[D^{f(\cdot)}(n) = 1] = Pr[\text{Mac-forge}_{A,\tilde{\Pi}}(n) = 1].$$

Ma poiché F é pseudocasuale, $\exists$ una funzione $negl(n)$ tale che

$$|Pr[D^{F_k(\cdot)}(n) = 1] - Pr[D^{f(\cdot)}(n) = 1]| \leq negl(n)$$

$$\Downarrow$$

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

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

Come usare Π' per costruire Π? Alcune idee

1. $m_1 m_2 \ldots m_d \rightarrow t_1 t_2 \ldots t_d$, dove $t_i = Mac'_k(m_i)$
   - previene l'invio di blocchi non autenticati
   - non previene attacchi di riordino $m_2 m_1 \rightarrow t_2 t_1$
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"

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

$$\downarrow$$

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

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

Lunghezza arbitraria

CONSTRUCTION 4.7

Let $\Pi' = (\mathsf{Mac}', \mathsf{Vrfy}')$ be a fixed-length MAC for messages of length n. Define a MAC as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ of (nonzero) length $\ell < 2^{n/4}$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Choose a uniform identifier $r \in \{0,1\}^{n/4}$.

  For $i = 1, \ldots, d$, compute $t_i \leftarrow \mathsf{Mac}'_k(r \| \ell \| i \| m_i)$, where i, ℓ are encoded as strings of length $n/4$.[†] Output the tag $t := \langle r, t_1, \ldots, t_d \rangle$.
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ of length $\ell < 2^{n/4}$, and a tag $t = \langle r, t_1, \ldots, t_{d'} \rangle$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Output 1 if and only if $d' = d$ and $\mathsf{Vrfy}'_k(r \| \ell \| i \| m_i,\ t_i) = 1$ for $1 \le i \le d$.

[†] Note that i and ℓ can be encoded using $n/4$ bits because $i, \ell < 2^{n/4}$.

Lunghezza arbitraria

Teorema 4.8 Se Π' é un Mac a lunghezza fissa per messaggi di lunghezza n, la Costruzione 4.7 produce un Mac sicuro per messaggi di lunghezza arbitraria.

Dim. Intuizione. Nell'ipotesi in cui Π' é sicuro:

- A non puó introdurre un nuovo blocco con un tag valido
- le informazioni aggiunte ad ogni blocco prevengono il riuso

Sia Π il Mac della Costruzione 4.7.

Mostreremo che:

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

Lunghezza arbitraria

Indichiamo con *Repeat* l'evento "lo stesso identificatore é presente in due dei tag restituiti dall'oracolo nell'esperimento".

Sia $(m, t =< r, t_1, \ldots >)$ l'output finale di A.

Indichiamo con *Newblock* l'evento "almeno uno dei blocchi $r||\ell||i||m_i$ non é stato autenticato in precedenza tramite query all'oracolo."

Risulta $Pr[\text{Mac-forge}_{A,\Pi}(n) = 1]$ uguale a:

$$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge Repeat] + Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{Repeat}]$$

Inoltre, $Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{Repeat}]$ risulta uguale a:

$$\begin{aligned} & Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{Repeat} \wedge Newblock] \\ + \quad & Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{Repeat} \wedge \overline{Newblock}]. \end{aligned}$$

$$\Downarrow$$

Lunghezza arbitraria

Dalle precedenti risulta:

$$\begin{aligned}
&Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \\
&\quad \leq Pr[\mathit{Repeat}] + Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \mathit{Newblock}] \\
&\quad\quad + Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{\mathit{Repeat}} \wedge \overline{\mathit{Newblock}}].
\end{aligned}$$

Consideriamo i tre termini separatamente.

Claim 4.9 $Pr[\mathit{Repeat}]$ é trascurabile.

Dim. Sia $q(n)$ il numero (polinomiale) di query all'oracolo effettuate da A. L'oracolo usa r_i tali che $|r_i| = n/4$, scelti uniformemente a caso in $\{0,1\}^{n/4}$.

$$\Rightarrow Pr[\mathit{Repeat}] \quad = \quad Pr[r_i = r_j \text{ per } i \neq j] \quad \leq \quad \frac{q(n)^2}{2^{n/4}}.$$

□

Lunghezza arbitraria

Soffermiamoci ora sul termine
$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{Repeat} \wedge \overline{Newblock}]$.

Faremo vedere che se $\text{Mac-forge}_{A,\Pi}(n) = 1$ ma *Repeat* non si verifica, allora **deve** verificarsi *Newblock*. Pertanto, risulta:

$$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \overline{Repeat} \wedge \overline{Newblock}] = 0.$$

Sia $q(n)$ il num. di query che A fa all'oracolo ed r_i l'identificatore della i-esima query.

Se *Repeat* non si verifica, allora $r_1, \ldots, r_{q(n)}$ sono *tutti distinti*. Ora se $r \notin \{r_1, \ldots, r_{q(n)}\}$, allora *Newblock* chiaramente si verifica!

Altrimenti, deve risultare $r = r_j$ per qualche j ed i blocchi

$$r||\ell||1||m_1, \quad r||\ell||2||m_2, \quad \ldots \text{ di } (m, t =< r, t_1, \ldots >)$$

possono essere stati autenticati soltanto durante la j-esima query all'oracolo. Anche in questo caso, mostriamo che *Newblock* si verifica!

Sia m_j il messaggio usato da A per la sua j-esima query all'oracolo ed ℓ_j la sua lunghezza.

Occorre considerare due casi:

1. $\ell \neq \ell_j \Rightarrow$ i blocchi autenticati durante la j-esima query hanno *tutti* $\ell_j \neq \ell$ in seconda posizione. Quindi, nessun blocco di m puó essere stato autenticato durante la j-esima query. Per esempio, $r||\ell||1||m_1$ **non** é stato autenticato durante la j-esima query $\Rightarrow$ *Newblock* si verifica

2. $\ell = \ell_j \Rightarrow$ poiché $m \neq m_j$ (per ipotesi $\text{Mac-forge}_{A,\Pi}(n) = 1$) $\exists$ un indice i per cui $m_i \neq m_j^{(i)}$, dove $m_j^{(i)}$ denota l'i-esimo blocco di m_j. Il blocco $r||\ell||i||m_i$ **non** é stato autenticato durante la j-esima query $\Rightarrow$ *Newblock* si verifica.

$\square$

Lunghezza arbitraria

Claim 4.10 $Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \textit{Newblock}]$ é trascurabile.

Il claim poggia sulla sicurezza di Mac'.

Costruiamo, infatti, un avversario A' ppt che attacca lo schema Π' (utilizzato da Π)

A' ha successo nel produrre un tag valido su un messaggio non autenticato in precedenza con

$$Pr[\text{Mac-forge}_{A',\Pi'}(n) = 1] \geq Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \textit{Newblock}]$$

Poiché Π' é per ipotesi sicuro, allora la prima probabilitá deve essere trascurabile

$$\Downarrow$$

$Pr[\text{Mac-forge}_{A,\Pi}(n) = 1 \wedge \textit{Newblock}]$ deve essere altrettanto.

Lunghezza arbitraria

A' esegue A come subroutine e risponde alle query di A per tag di specifici messaggi come segue

- sceglie $r \leftarrow \{0,1\}^{n/4}$
- divide il messaggio in modo opportuno
- effettua le query necessarie al proprio oracolo $Mac'(\cdot)$

A' tiene traccia di tutte le query che ha rivolto a $Mac'(\cdot)$

Quando A dá in output $(m, t =< r, t_1, \ldots >)$, A' controlla se l'evento *Newblock* si verifica.

Se ció accade, A' trova il primo blocco $r||\ell||i||m_i$ che non é stato usato in una query per $Mac'(\cdot)$ e dá in output $(r||\ell||i||m_i, t_i)$.

Altrimenti, non dá nulla in output.

Lunghezza arbitraria

La "vista" che A ha quando eseguito come subroutine da A' é distribuita esattamente come in $\text{Mac-forge}_{A,\Pi}(n)$

$$\Downarrow$$

Le probabilitá di $\text{Mac-forge}_{A,\Pi}(n)$ e *Newblock* **non** cambiano.

$$\Downarrow$$

Se $\text{Mac-forge}_{A,\Pi}(n) = 1$, il tag restituito da A é corretto su *ciascun blocco* del messaggio m, ed in particolare sul blocco che non é stato mai usato in una query da A' verso l'oracolo Mac' e che A' dá in output.

Quindi, ogni volta che $\text{Mac-forge}_{A,\Pi}(n) = 1$ e *Newblock* occorre, $\text{Mac-forge}_{A',\Pi'}(n) = 1 \Rightarrow$ il Claim 4.10 vale.

Pertanto, mettendo assieme i risultati provati, $Pr[\text{Mac-forge}_{A,\Pi}(n) = 1] \leq negl(n)$.

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

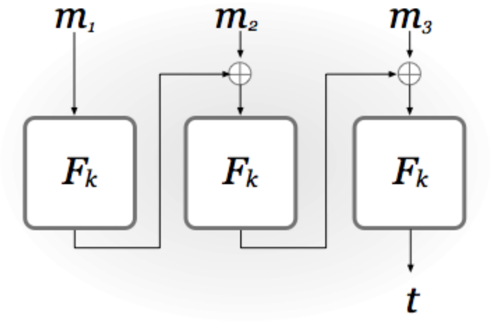
La costruzione **non** é efficiente.

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

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.

Lunghezza arbitraria

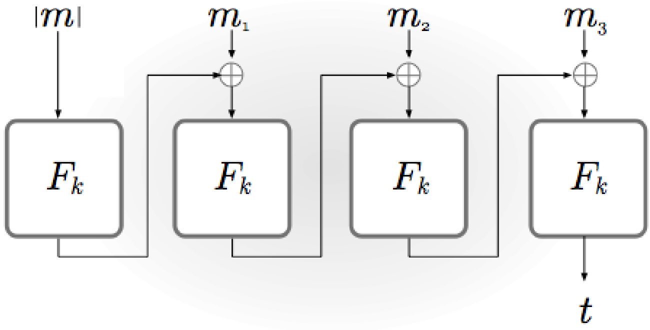
Conclusione: nuovamente, anche piccole ed apparentemente innocue modifiche ad una primitiva crittografica possono renderla insicura.

Messaggi di lunghezza arbitraria: esistono due modi per estendere la costruzione

1. Anteporre ad m la sua lunghezza $|m|$, codificata come una stringa di n bit e calcolare il CBC-MAC di base
2. L'algoritmo di generazione della chiave sceglie due chiavi e poi
   - calcola il CBC-MAC di base usando k_1, sia esso t
   - dá in output $t^* := F_{k_2}(t)$.

Le due chiavi possono essere anche generate al volo usando una PRF, i.e., $k_1 = F_k(1)$ e $k_2 = F_k(2)$, memorizzando una sola chiave k. Ma é ancora costoso.

Solitamente la prima strategia viene preferita.



GMAC e Poly1305

Nuove soluzioni efficienti, adottate da diversi standard.

- Schema generale
  1. funzione $\epsilon(n)$-difference universal $h_{k_h}(m)$ e PRF $F_{k_F}(\cdot)$
  2. MAC $t =< r, s >$ (randomizzato, r uniforme), dove $s = h_{k_h}(m) + F_{k_F}(r)$
- GMAC e Poly1305: block cipher + $h_{k_h}(m)$ basata su polinomi
- Usano campi finiti diversi

La PRF viene valutata una sola volta

Riduzioni di sicurezza *piú strette* rispetto a CBC-MAC

Al solito n indica il parametro di sicurezza. K_n, M_n, T_n insiemi di chiavi, messaggi e tag (T_n deve essere un gruppo per ogni n, ci torneremo ...)

Definizione 4.14. Una funzione con chiave $h : K_n \times M_n \rightarrow T_n$ é $\epsilon(n)$-difference universal se, per tutti gli n, per ogni $m, m' \in M_n$, ed ogni $\Delta \in T_n$, risulta

$$Pr[h_k(m) - h_k(m') = \Delta] \leq \epsilon(n),$$

per k scelta uniformemente in K_n.

Nota che deve essere $\epsilon(n) \geq 1/|T_n|$. Inoltre h deve essere efficientemente calcolabile.

Schema Generico

1. Gen: su input 1^n, sceglie uniformemente $k_h \in K_n$ e $k_F \in \{0,1\}^n$
2. Mac: su input (k_h, k_F) ed $m \in M_n$, sceglie r uniformemente in $\{0,1\}^n$ e calcola il tag
$$t =< r, h_{k_h}(m) + F_{k_F}(r) > .$$
3. Vrfy: su input (k_h, k_F), $m \in M_n$ e $t =< r, s >$, dá in output 1 se
$$s = h_{k_h}(m) + F_{k_F}(r).$$

Tecnicalitá che ci servono (ci torneremo ...).

Un campo finito F_q contenente q elementi esiste per ogni potenza di un numero primo.

Un polinomio non nullo di grado ℓ su un campo finito ha al piú ℓ zeri.

Sia F un campo finito. Sia $F^{<\ell}$ l'insieme dei vettori con meno di ℓ elementi. Sia $M = F^{<\ell}$. Sia $m = m_1, \ldots, m_{t-1} \in M$, con $t \leq \ell$, e sia m_t una codifica della lunghezza di m. Sia $m(X)$ il polinomio

$$m(X) = m_1 \cdot X^t + m_2 \cdot X^{t-1} + \ldots + m_t \cdot X.$$

La funzione $h : F \times F^{<\ell} \to F$ con chiave definita come

$$h_k(m) = m(k)$$

é una funzione $\ell/|F|$-difference universal.

Prova. Siano $m, m' \in F^{<\ell}$ e $\Delta \in F$. Consideriamo il polinomio

$$P(X) = m(X) - m'(X) - \Delta.$$

P é un polinomio non nullo di grado al piú ℓ. Risulta

$$Pr_{k \in F}[h_k(m) - h_k(m') = \Delta] = Pr_{k \in F}[P(k) = 0] \leq \ell/|F|,$$

poiché P ha al piú ℓ radici (zeri).

GMAC e Poly1305

La costruzione GMAC usa un cifrario a blocchi per instanziare la PRF $F(\cdot,\cdot)$ e la funzione $\ell/|F|$-difference universal basata sui polinomi definita sul campo $F_{2^{128}}$, contenente 2^{128} elementi.

La costruzione Poly1305 é realizzata in modo simile, ma usa il campo F_p definito per il primo $p = 2^{130} - 5$.

Il tag finale é calcolato come

$$t =< r, [h_{k_h}(m) + F_{k_F}(r) \bmod 2^{128}] > .$$