Crittografia avanzata
Lezione del 21 Marzo 2011

Attacchi alle funzioni hash

- Collisioni: trovare due messaggi M1 e M2 tali che H(M1) = H(M2) in meno di 2^(L/2) tentativi
  - L è la lunghezza dell'hash
  - Mezzi per via del birthday attack
- First preimage: dato X, trovare M tale che H(M)=X in meno di 2^L tentativi
- Second preimage: dato M1, trovare M2 tale che H(M1) = H(M2) in meno di 2^L tentativi

Message Authentication Codes

- Scopo: garantire integrità e autenticità di un messaggio
- Tipicamente brevi
- Sono diversi da meccanismi simili destinati alla rilevazione di errori (es. CRC) perché:
  - Usano una chiave
  - Sono crittograficamente robusti
- Caratteristiche simili alle funzioni hash, ma con una chiave
- Es. CBC MAC trasforma un algoritmo a blocchi in un MAC (con alcune cautele)

MAC con chiave

- Garanzia dell'integrità di un file
  - Firma: chiave asimmetrica
  - Hash con chiave: chiave simmetrica
- Come costruire un hash con chiave?
  - Si cercano soluzioni che non siano (troppo) dipendenti dalla funzione hash utilizzata

HMAC: Keyed-hash MAC

- Definito in RFC 2104 e FIPS PUB 198

  $HMAC(K,m) = H((K \oplus opad) \| H((K \oplus ipad) \| m))$

- Dove:
  - outer padding opad = 0x5c5c5c…5c5c
  - inner padding ipad = 0x363636…3636
  - $\|$ è la concatenazione
- Rende bene l'idea di come la crittografia non si improvvisi

Obiettivi

- Usare funzioni Hash esistenti senza modificarle
- Mantenere le prestazioni
- Usare le chiavi in modo semplice
- Avere una funzione di uso generale la cui robustezza sia stata analizzata
  - Estremamente importante per l'IETF, la crittografia e la sicurezza compaiono dappertutto, anche dove non ci sono competenze specifiche...
- Permettere di sostituire la funzione di hash con facilità

Altre possibilità?

- Perché non semplicemente:

$$HMAC(K,m) = H(K \parallel m)$$

$$HMAC(K,m) = H(m \parallel K)$$

...

- Per ognuna delle combinazioni ovvie esistono attacchi applicabili in determinati casi e contesti
  - O si conoscono esattamente i casi e si possono escludere, o è meglio affidarsi a funzioni robuste
  - Es. uso di MD5 e SHA-1 in SSL

Soluzioni semplici

- MAC = H(key ‖ message)
  - Con alcune è possibile appendere testo al messaggio senza conoscere la chiave, ottenendo un MAC valido
    - la propagazione della chiave è limitata
- MAC = H(message ‖ key)
  - Se c'è una collisione nell'Hash con messaggio, per molte funzioni c'è una collisione nel MAC
    - Senza conoscere la chiave
- HMAC rientra nella famiglia:
  - MAC = H(key1 ‖ H(key2 ‖ message))

Altro?

- Si vogliono funzioni efficienti
  - Che si possano calcolare su uno stream di messaggio
  - Che si possano verificare su uno stream di messaggio

Uso di MAC e firma

- La firma digitale e i MAC sono due strumenti per garantire l'integrità di messaggi e documenti
- Si tratta di strumenti di base, ma l'uso dipende dal contesto
- Problema: i documenti, messaggi e file “vivi” devono essere modificati
  - Di fatto, i MAC e le firme corrispondono a “foto” della situazione

Usi di HMAC

- Garanzia “temporanea” di autenticità e integrità in uno scambio di messaggi
- Simile alla firma, ma:
  - Non c'è non repudiation
  - Non è pensato per una conservazione a lungo termine

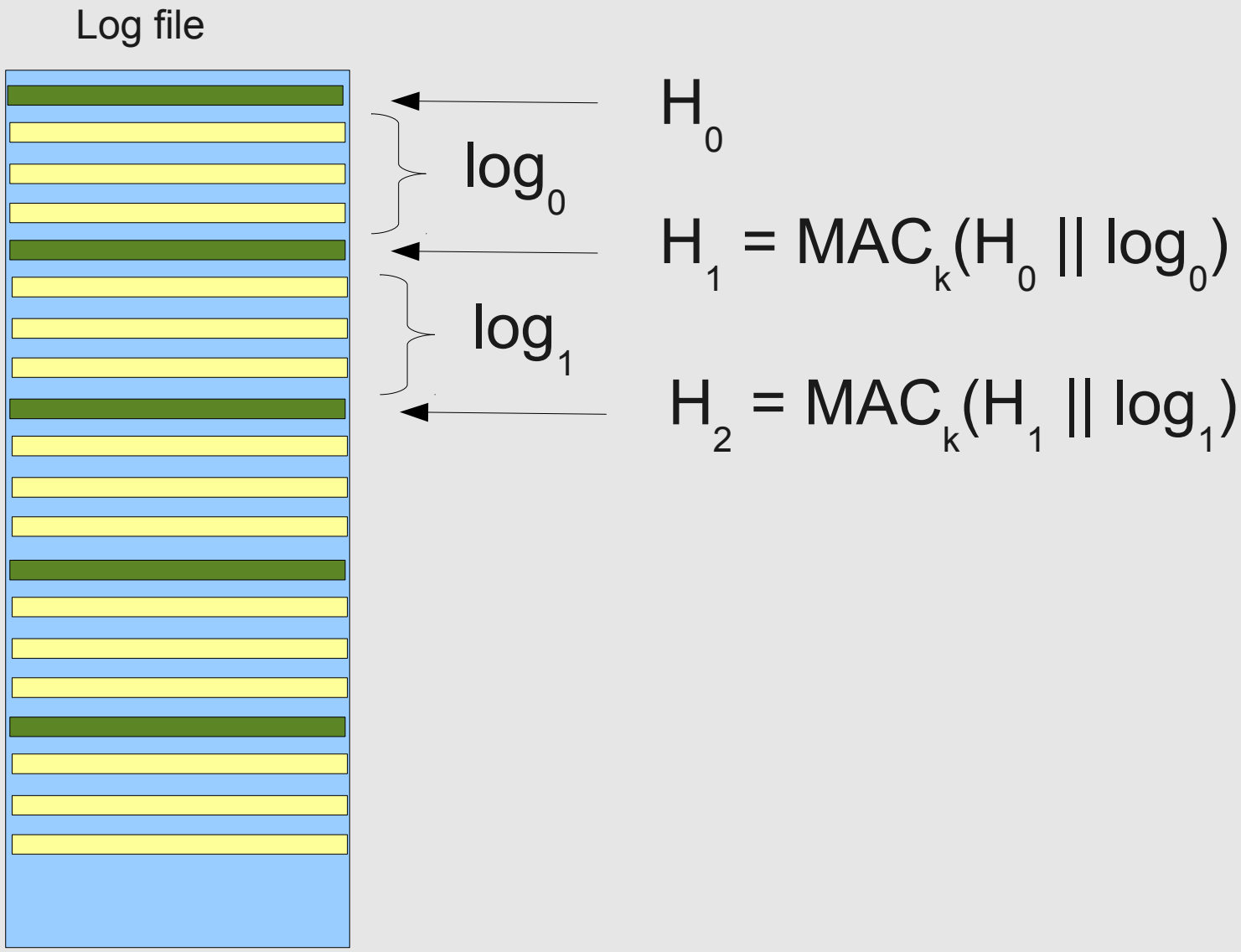
Uso di HMAC

- Protocolli di comunicazione
  - Una volta contrattata una chiave condivisa
- In generale, quando meccanismi di firma non sarebbero efficienti
  - O quando è comunque utilizzata una chiave condivisa
- Importante: queste protezioni hanno protetto protocolli Internet anche in presenza di collisioni nelle funzioni hash (vedi RFC

Caso particolare: i log

- L'integrità dei log è richiesta da molte normative ed è utile da verificare in generale
- I log sono file che crescono in append (salvo rotazione)
  - Un caso particolare più semplice da gestire
  - Garantire l'integrità rispetto a cosa e a quando?
    - Accesso non autorizzato al logserver?
    - Manipolazione successiva?

Chained hashing



Hash chaining

- Si parte da un pad casuale $H_{0,}$ inserito in testa al file di log e una chiave K
- Periodicamente, di calcola un MAC e si inserisce nel file

$$H_i = MAC(H_{i-1} \,||\, log)$$

  - Log è il frammento del file di log successivo a H_{i-1}
  - Può essere fatto per ogni evento...
- Se il file viene modificato nelle parti coperte dai MAC, la modifica risulta evidente

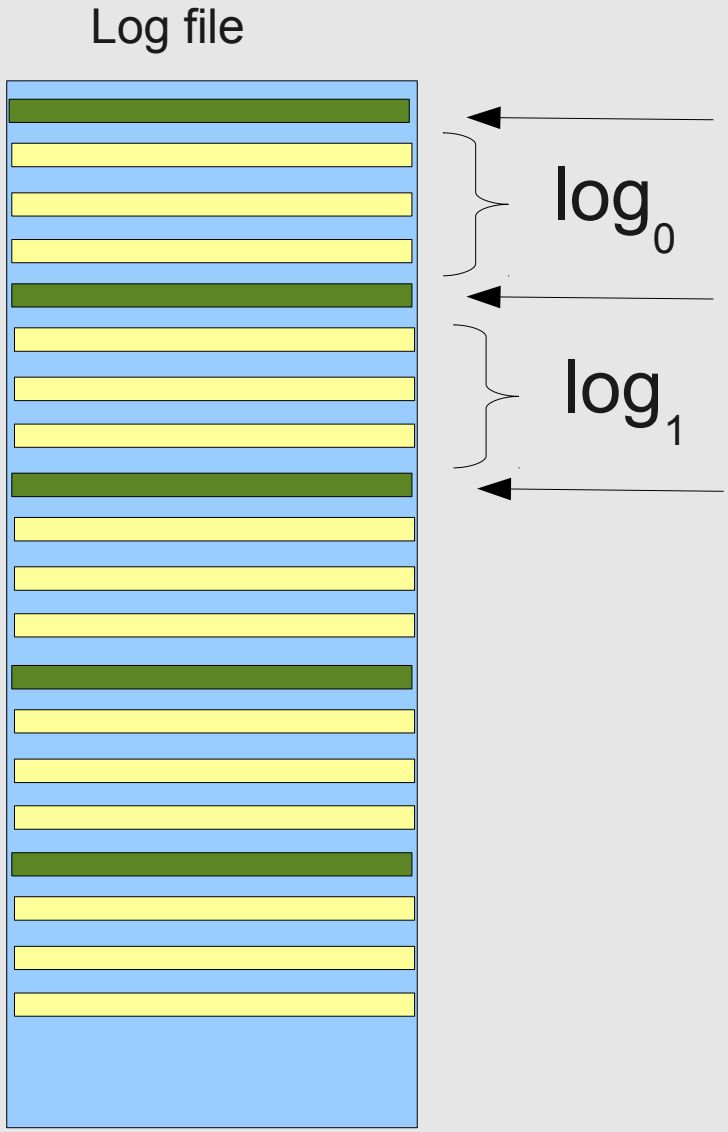
Limiti

- Frequenza:
  - Gli eventi possono essere modificati finché non vengono inclusi in un MAC
- Dove è stato alterato il file?
  - Chi vuole manomettere il file può modificarlo in diverse parti per “nascondere” le vere modifiche
    - Ad esempio, modificare gli hash

Protezione della chiave

- Se la chiave è in memoria, alla compromissione del sistema può corrispondere una compromissione della chiave
  - Perfect forward secrecy: la nuova chiave è un hash della precedente
  - La compromissione della chiave non permette di modificare i MAC già apposti

Chiave modificata



H_0 $k_1=H(k_0)$

$H_1 = MAC_{k1}(H_0 \,||\, log_0)$ $k_2=H(k_1)$

$H_2 = MAC_{k2}(H_1 \,||\, log_1)$ $k_3=H(k_2)$

Rainbow tables

- Tipicamente usate per hash di password, ma non solo
- È possibile precomputare (parte di) una funzione crittografica
  - Parte della funzione
  - Su un sottoinsieme delle chiavi
- Trade-off fra spazio e tempo

Hash chains

- Esempio: precomputare una funzione hash su password banali
  - Ma lo spazio necessario può ancora essere eccessivo
- Soluzione: si trova una funzione “di riduzione” R con un comportamento pseudocasuale da hash a password e si costruiscono catene

  p1->h1->p2->h2->....->pn
- Si memorizzano solo p1 e pn
- Questo non ci garantisce che ogni password sia in una qualche catena

Utilizzo

- Dato un hash hx, si inizia a computare una catena che parta da hx

  hx->px->...

- Fino a quando non si trova una password che sia alla fine di una catena (pn)
- A quel punto si parte dalla p1 di quella catena e si calcola la catena
  - C'è una “buona probabilità” che la catena contenga hx e quindi che il valore precedente sia la password
  - Ma non è detto, ci sono collisioni: se non c'è, si prosegue con la catena

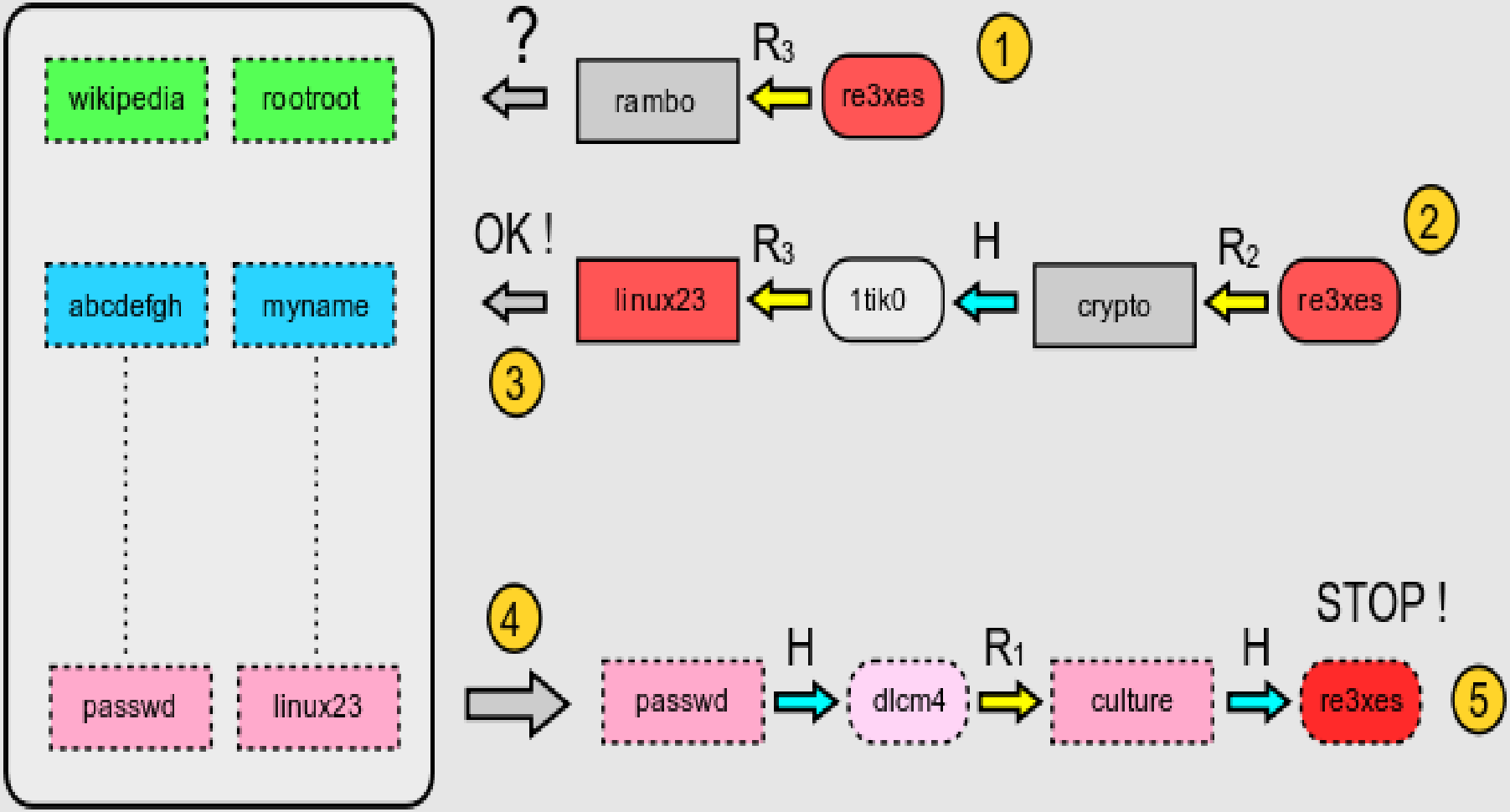
Collisioni

- Più password possono generare lo stesso hash
- Ancora più facile con la funzione R
- Più catene “convergono” sulla stessa pn
- Es. R=”primi 6 caratteri”
  - In questo caso tutti gli hash che iniziano con gli stessi 6 caratteri “convergono” sulla stessa password
  - Quindi abbiamo nelle nostre catene molte meno password (e hash) di quanti ce ne aspetteremmo

Rainbow tables

- Invece di una sola funzione R se ne definisce una sequenza R1... Rn
- In questo modo, perché ci sia una collisione fra due catene, devono collidere allo stesso step (altrimenti, essendo diverse le funzioni, poi divergono nuovamente)

Uso delle raibow tables



Fair cryptosystems

- Problema: le comunicazioni cifrate impediscono le intercettazioni legittime
- Soluzione: dare alle autorità accesso alla chiave
  - Ma in modo “controllato”
- Possiamo:
  - Prendere la chiave (privata) e darla in “secret sharing” a diverse autorità
  - Prendere una chiave robusta e darne una parte “in secret sharing” a più autorità

Come verificare la correttezza?

- Un malintenzionato potrebbe dare parti di una chiave “fasulla”
- La cosa si scoprirebbe solo al momento della ricostruzione
  - Vale la pena di correre il rischio
- Come fornire la possibilità di controllare la correttezza dell'escrow senza scoprire la chiave?
  - Una soluzione è imporre l'hardware di cifratura

Esercizio: riuso del keytext

- Il testo è codificato come segue:

Plaintext	Binary	Plaintext	Binary	Plaintext	Binary
A	00000	J	01001	S	10010
B	00001	K	01010	T	10011
C	00010	L	01011	U	10100
D	00011	M	01100	V	10101
E	00100	N	01101	W	10110
F	00101	O	01110	X	10111
G	00110	P	01111	Y	11000
H	00111	Q	10000	Z	11001
I	01000	R	10001	space	11010

Depth

- Riutilizzo del keytext (errore comune).
- c=k⊕m
- Due testi cifrati che riutilizzino il keytext si dicono “in depth”
- Nell'esempio che segue, almeno un testo contiene “the”

CIPHERTEXT 1:

11001 10101 01110 01111 01011 00110 10001 00000
10001 00011 10010 11011 00000 01100 01101

CIPHERTEXT 2:

01001 11001 10100 01110 10101 01111 00101 00011
01111 01111 11001 11000 01101 10011 01001

Soluzione

- Sappiamo che se il keytext è riutilizzato, abbiamo da c=k⊕m che
- c1⊕ c2= (k1⊕ m1)⊕(k2⊕m2)=m1⊕ m2
- Soluzione:

```
Close the doors
Shut the windows
```