FUNZIONI HASH ONE-WAY ITERATE

SNEFRU SHA-1

di

Davide Gallo

per

Elementi di Crittografia 2004/2005

Prof.sa Rosaria Rota

Funzioni Hash : contesto

Oltre alla Segretezza i moderni sistemi di crittografia a chiave asimmetrica vogliono garantire :

1. **Integrità** del messaggio inviato (si vuole evitare il rischio che, per errori di trasmissione o per intrusione, un messaggio pervenga al destinatario corrotto rispetto all'originale)

2. **Autenticazione** dell'invio (il mittente deve essere riconoscibile come autore del messaggio)

3. **Impossibilità di ripudio** (il mittente non deve poter negare la produzione del messaggio che è avvenuta per sua volontà in un certo istante)

4. **Identificazione** del destinatario (il mittente deve essere sicuro dell'associazione chiave pubblica/destinatario)

Funzioni Hash : finalità

- I primi due punti (Integrità e Autenticazione) sono risolti con la nota tecnica della **Firma Digitale**, che nella sua forma più semplice, non è altro che il messaggio cifrato con la chiave privata del mittente.

- Poiché cifrare ai soli fini di firma un intero messaggio è un'operazione onerosa in termini di calcolo, è più conveniente cifrare con la chiave privata del mittente un "**Riassunto**" (residuo) del messaggio.

- Lo scopo delle **Funzioni Hash** è quello di creare un riassunto o meglio l'**Impronta del messaggio** (in inglese **Fingerprint** o anche **Digest**).

Funzioni Hash : caratteristiche

1. Una Funzione di Hash (o **Message Digest**) è una funzione **H** che, dato un **input M** di dimensione qualsiasi, produce un **output Y** (Hash) di dimensione fissa (in genere 128 bit) : **Y = H(M)**

2. L'idea alla base è che il valore hash H(M) sia una rappresentazione **non ambigua e non falsificabile** del messaggio M

3. Dato un input X deve essere **altamente improbabile** che un altro input Z generi lo stesso hash (**Collisioni**) cioè : **H(Z) = H(X)**

4. Inoltre la funzione deve essere **computazionalmente semplice da eseguire, cioè calcolare Y come H(M)**

5. **One-Way (irreversibili) : cioè deve essere computazionalmente impossibile, noto H(M), ricalcolare M**

Funzioni Hash : collisioni

Le Funzioni Hash si classificano in base alla resistenza alle collisioni:

- **Resistenza Debole alle Collisioni :**

  "Dato M, è computazionalmente impraticabile

  trovare un altro M´ tale che H(M) = H(M´)"

- **Resistenza Forte alle Collisioni :**

  "Computazionalmente impraticabile

  trovare una coppia (M, M'), con M'≠ M,

  tale che H(M) = H(M')

Dalla Resistenza alle Collisioni dipende la Sicurezza di una Funzione Hash !

Funzioni Hash : attacchi

Le due proprietà sull'assenza di collisioni (debole e forte), corrispondono due diversi tipi di attacchi :

- **Attacco a forza bruta (Brute Force) :**
  Trovare un messaggio che produca un dato hash (cioè un hash uguale a quello di un messaggio dato)

- **Attacco del compleanno (Birthday Attack) :**
  Trovare due messaggi che producano lo stesso hash, indipendentemente dal valore di questo hash.

La **complessità** dei due attacchi è molto diversa : se l'hash è lungo m bit, la complessità del primo è $\mathbf{2^m}$, quella del secondo è $\mathbf{2^{m/2}}$.

Attacco Brute Force

Caso di una **funzione hash debole** contro le collisioni con **output di 32 bit**:

1. Mefisto prepara 2 versioni di un contratto M ed M* tali che :
    - **M** è favorevole ad Annarella
    - **M*** è sfavorevole ad Annarella

2. Mefisto **modifica M* a caso** facendo piccoli cambiamenti come aggiunta di spazi, punteggiatura, sinonimi (32 possibilità sono **2^{32} messaggi** !) **finchè H(M) = H(M*)**

3. Annarella firma M con la sua chiave => **Firma(h(M))**

4. Mefisto ha quindi ottenuto anche la firma di M* => **Firma(h(M*)) !!**

Possibile Messaggio Falsificabile

Cara Annarella,

ti {scrivo / sto scrivendo} da {un bellissimo / uno spendido} posto {della costiera Amalfitana / vicino Amalfi}

……..

{Colui / La persona} che {ti porterà / è portatore di} questa {lettera / missiva} è di fiducia!

……..

Paradosso del Compleanno

Domanda : “Quante persone scegliere a caso affinché, con probabilità ≈ 50% ,ce ne siano almeno due con lo stesso compleanno?”

Risposta : “Ne bastano 23 !”

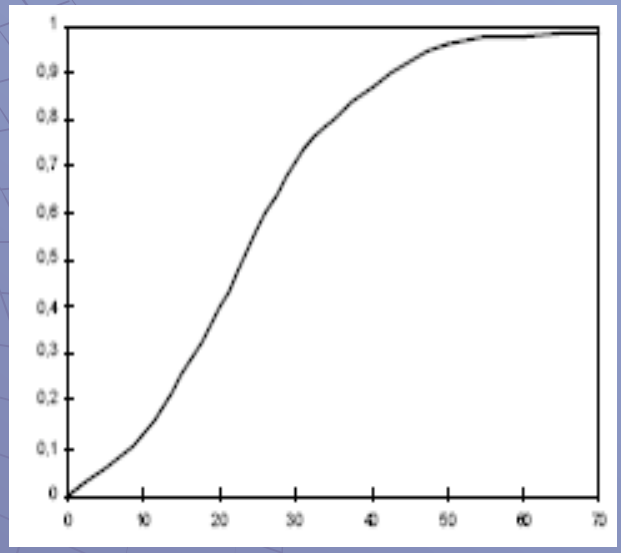
N = 365 numero di giorni possibili ; T = numero di persone prese a caso

$\_$ = Prob. di trovarne due uguali e $1-\_$ = Prob. che siano tutti diversi

$$\_ = \frac{365 \_ 364 \_ \ldots \_ (N - t + 1)}{365^{t}}$$

Quindi con $t = 23 \rightarrow \_ = 0{,}5073$

mentre con $t = 100 \rightarrow \_ = 0{,}99997$



Attacco del Compleanno

Applicando all'inverso questo ragionamento otteniamo il principio di base :

1. **Scelgo a caso diversi messaggi**
2. **Verifico se ottengo almeno due valori hash uguali (collisione)**

Problema da risolvere :

Dato un insieme di cardinalità **n** (output dell'hash) , quanti elementi **t** (messaggi) scegliere se si vuole che la probabilità che ci siano almeno due elementi uguali sia _ ?

$$t \approx \sqrt{n \cdot 2 \ln \frac{1}{1-\varepsilon}}$$

cioè $t \approx \sqrt{n}$ con $\varepsilon = 1 - e^{-t(t-1)/n}$

es. con $n = 2^{128}$ servono $t = 2^{64}$

Sicurezza con Hash 128 bit

- **Costo di un attacco per computare collisioni $2^{64} \approx 2*10^{19}$ valutazioni della funzione.**
- **Attacco < 1mese con costo di $10.000.000**

  P. van Orschot e M. Wiener [1994]
- **I due autori ipotizzano che il costo si dimezza ogni 18 mesi.**
- **Oggi quindi costerebbe solo $75.000 dollari, costo che qualcuno potrebbe ritenere economicamente accettabile per falsificare un contratto !**

Funzioni Hash Iterate

Caratteristiche :

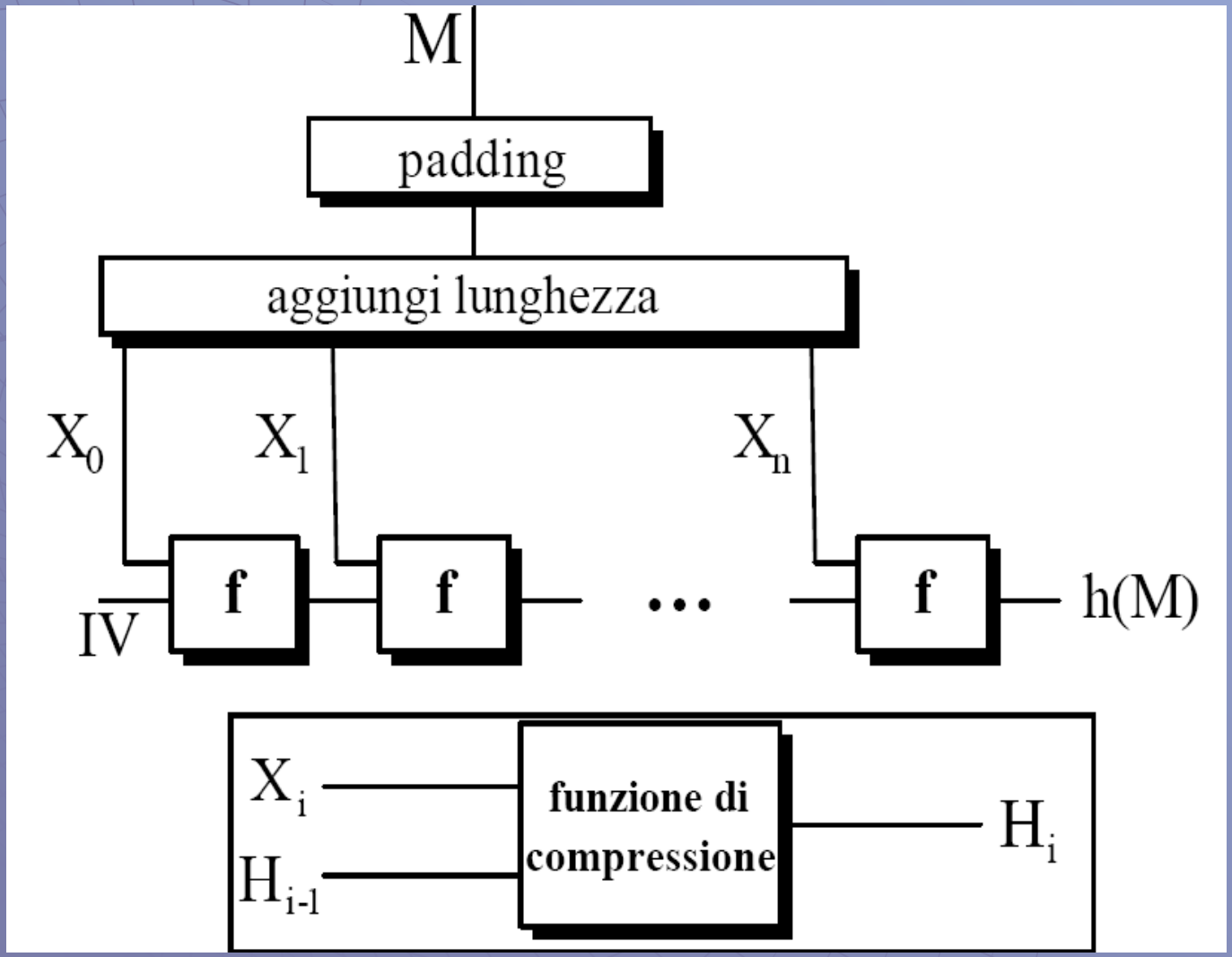
- **Veloci :** implementabili via software in modo efficiente
- **Sicure :** difficile trovare due messaggi diversi che vadano in collisione

Principi :

- **Input di lunghezza fissa :** il messaggio viene suddiviso in blocchi di dimensione prefissata pari all'input (in genere 512 bit),quindi la lunghezza totale deve essere un multiplo di quella dei blocchi, ciò si ottiene aggiungendo un **PADDING**

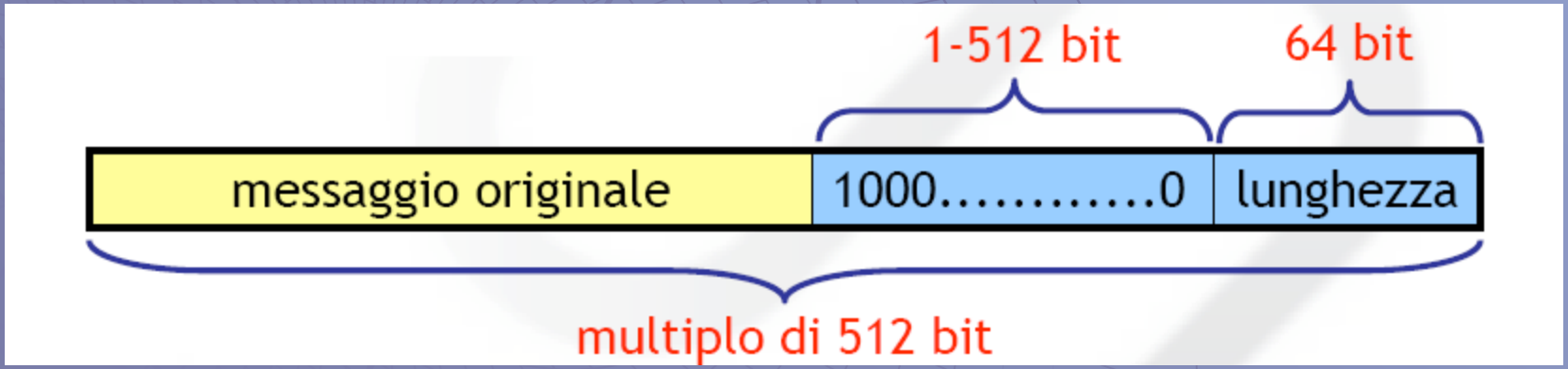
- **Parametrizzazione :** la funzione viene iterata in modo che al passo n prende in input un blocco del messaggio e l'hash calcolato al passo n-1, inoltre viene inizializzata con un valore Random detto **IV** (initializing value)

Modello di funzionamento



Padding

- Va aggiunto al messaggio un certo numero di bit in modo che la lunghezza totale risulti un multiplo di 512 bit.
- Si aggiunge un bit a 1 e poi tanti bit a 0 quanto basta perché la lunghezza risulti di 64 bit minore rispetto ad un multiplo di 512 bit (se la lunghezza originale è già corretta si aggiungono comunque 512 bit).
- Si aggiungono 64 bit contenenti la lunghezza originale del messaggio (modulo 2^{64}).



Funzioni di Compressione

- **Funzione HASH** applicata ad ogni iterazione su uno dei blocchi del messaggio insieme al risultato dell'iterazione precedente.
- E' costituita dalla composizione di operazioni bit-wise : **SHIFT, AND, OR, XOR, ROTATION**.
- E' suddivisa in passi di elaborazione detti **ROUND.**
- Per la proprietà della composizione : se **$H(M)=F(M_1)°….° F(M_n)$** ha collisioni tutte le F(Mi) hanno collisioni.

SNEFRU

- Funzione One-Way Hash inventata da Ralph Merkle, nel 1990
- Capostipite di una famiglia di algoritmi di crittografia che include i successori Khufu e Khafre (leggendari Faraoni Egizi)
- Khufu è veloce e adatta a grandi quantità di dati perché si avvale di cifrari precalcolati
- Khafre è lenta e adatta a piccole quantità di dati (non usa cifrari precalc.)
- Elabora blocchi di 512 bit
- Fornisce in outbut digest di 128 oppure 256 bit
- La funzione prende in input l'elemento calcolato all'iterazione precedente e lo cripta usando un **cifrario con una chiave fissa**, dopodichè applica al risultato uno XOR con il blocco corrente :

$$\mathbf{F(B_n)= Encript(key, F(B_n\text{-}1)) \ XOR \ B_n}$$

SHA-1

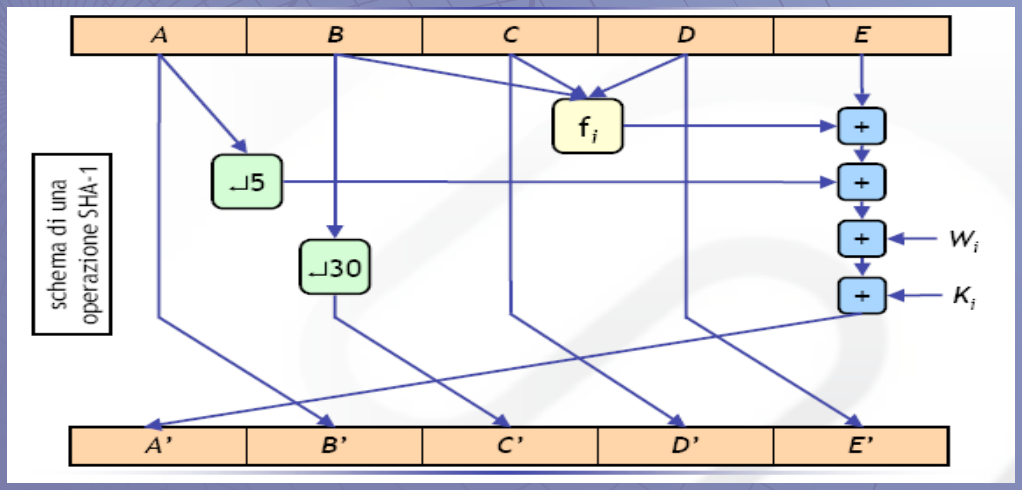
- SHA-1 (Secure Hash Algorithm 1) è stato progettato dal NIST
- Il messaggio (che deve essere di lunghezza inferiore a 2^{64} bit) viene elaborato a blocchi di 512 bit.
- Fornisce in outbut digest di 160 bit.
- Ad ogni iterazione si calcola una funzione che dipende dal blocco corrente del messaggio e dal valore dell'hash all'iterazione precedente: il risultato viene sommato (per singola parola) al valore dell'iterazione precedente.
- Le 5 parole di 32 bit del digest (A, B, C, D, E) vengono inizializzate con:

  A = 0x67452301, B = 0xEFCDAB89, C = 0x98BADCFE, D = 0x10325476,

  E = 0xC3D2E1F0.
- Ogni iterazione calcola dei valori di A, B, C, D, E che vengono sommati ai precedenti prima di passare all'iterazione successiva.
- Alla fine, **la concatenazione di A, B, C, D, E costituisce il digest** del messaggio.

SHA-1 : ITERAZIONI

- Ogni iterazione è composta da 80 operazioni.
- Il blocco corrente di messaggio (512 bit) viene utilizzato per costruire una stringa di 5 x 512 bit (80 parole di 32 bit).
- Si scorre la stringa di 80 parole una alla volta (80 operazioni), calcolando i nuovi valori A', B', C', D', E' (con operazioni bit-wise specifiche).
- Alla fine, i nuovi valori verranno aggiunti agli A, B, C, D, E precedenti.
- Per calcolare la stringa vengono usati lo XOR e la Rotazione di 1 bit



Altre funzioni : RIPEMD, MD5

- **RIPEMD-160 :**
  - RACE Integrity Primitives Evaluation, progetto della Comunità Europea (1988-1992)
  - Tabella “left line” e “right line”
  - Input blocchi di 512 bit (16 parole da 32 bit)
  - Output digest di 160, 250, 320 bit
  - E’ risultata debole e facilmente attaccabile
- **MD5 :**
  - Message Digest 5 è stato progettato da Ron Rivest, ed è definito in RFC 1321 (1992)
  - Input blocchi di 512 bit
  - Output digest di 128 bit
  - Ogni iterazione prevede 4 passi per elaborare altrettante parole, e ogni passo 16 operazioni
  - Non è considerato molto sicuro

Implementazioni e prestazioni

	Size (byte)	Cicli	Mbit/sec	Mbyte/sec
MD2		2709	4.25	0.53
MD4	1190	241	191.2	23.90
MD5	1713	337	136.7	17.09
RIPEMD	2291	480	96.0	12.00
RIPEMD-128	2929	592	77.8	9.73
SHA-1	4323	837	55.1	6.88
RIPEMD-160	4808	1013	45.5	5.69
Snefru-128		5730	6.03	0.75
Snefru-256		5738	4.02	0.50
Tiger		1320	34.9	4.36

- **Architetture 80x86**
- **1996 (Pentium 90Mhz)**