



Funzioni Hash

lunghezza arbitraria/finita

Funzione Hash

b bit

- Idea alla base: il valore hash $h(M)$ è una rappresentazione non ambigua e non falsificabile del messaggio M
- Proprietà: comprime ed è facile da computare
- Applicazioni: firme digitali ed integrità dei dati



Firme digitali e Funzioni hash

Problema: firma digitale di messaggi grandi

Soluzione naive: Divisione in blocchi e firma per ogni blocco
problema per la sicurezza: una permutazione/composizione delle firme è una nuova firma

Soluzione di uso corrente:

firmare il valore hash del messaggio

[firma di M] = Firma_k($h(M)$)

Vantaggi: integrità dei dati ed efficienza degli algoritmi



Integrità dei dati e Funzioni hash

Tipico uso delle funzioni hash


- Computo al tempo T il valore hash del file M
- Conservo $H = h(M)$ in un luogo sicuro
- Per controllare se il file è stato successivamente modificato, calcolo $h(M')$ e verifico se $y = h(M')$

$h(M)$ è l'impronta digitale del file

Assicura se un file è stato modificato!





Funzioni Hash: Proprietà

- Facili da calcolare
-  ... poi?



Un possibile attacco

-  prepara 2 versioni di un contratto M ed M'
 - M è favorevole ad Annarella
 - M' è sfavorevole ad Annarella
- Modifica M' a caso (piccoli cambiamenti come aggiunta spazi: 32 possibilità sono 2^{32} messaggi!) finché $h(M) = h(M')$
- Annarella firma $M \Leftrightarrow \text{Firma}_k(h(M))$
-  ha quindi la firma di $M' \Leftrightarrow \text{Firma}_k(h(M'))$



Esempio di lettera fraudolenta

Cara Annarella,

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

.....

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

.....



Funzioni hash: sicurezza

- ❑ **Sicurezza forte:** computazionalmente difficile trovare 2 diversi messaggi con lo stesso valore hash
- ❑ **Sicurezza debole:** dato M è computazionalmente difficile trovare un altro M' tale che $h(M) = h(M')$
- ❑ **One-way:** dato y è computazionalmente difficile trovare M tale che $y = h(M)$



Sicurezza forte \supset One-way

- ❑ $h: X \rightarrow Z$ funzione hash, $|X| \geq 2 \cdot |Z|$
- ❑ Supponiamo **ALG** un algoritmo di inversione per h
- ❑ ... allora esiste un algoritmo *Las Vegas* che trova collisioni con probabilità $\geq 1/2$



Sicurezza forte \supset One-way

- ❑ $h: X \rightarrow Z$ funzione hash, $|X| \geq 2 \cdot |Z|$
- ❑ Supponiamo **ALG** un algoritmo di inversione per h
- ❑ ... allora esiste un algoritmo *Las Vegas* che trova collisioni con probabilità $\geq 1/2$

Scegli a caso x in X
 $z \leftarrow h(x)$
 $x' \leftarrow \text{ALG}(z)$
If $x' \neq x$ **then** x' ed x è una collisione
else fallito



Sicurezza forte \supset One-way

$$\begin{aligned} \text{Prob}(\text{successo}) &= \sum_{x \in X} \text{Prob}(\text{successo}|x) \cdot \text{Prob}(\text{scelgo } x) \\ &= \frac{1}{|X|} \sum_{x \in X} \frac{|X| - 1}{|X|} \quad [X] = \{x' \in X: x \neq x'\} \\ &= \frac{1}{|X|} \sum_{c \in C} \sum_{x \in c} \frac{|c| - 1}{|c|} \quad [C] = \{\{x\}: x \in X\} \\ &= \frac{1}{|X|} \sum_{c \in C} (|c| - 1) = \frac{1}{|X|} (\sum_{c \in C} |c| - \sum_{c \in C} 1) \\ &\geq \frac{|X| - |Z|}{|X|} \geq \frac{|X| - |X|/2}{|X|} = \frac{1}{2} \end{aligned}$$



Lunghezza valore hash

- ❑ Quanto grande l'hash per la sicurezza forte?
- ❑ Se $|\text{hash}(\bullet)| = 2^3$ bit ?
 - Quanti valori scegliere per essere certi di trovare almeno una collisione?
- ❑ $|\text{hash}(\bullet)| + 1$ diversi valori di M
 - \Rightarrow certezza di trovare almeno una collisione

Esempi $\left\{ \begin{array}{l} 2^{40} \text{ 😞} \\ 2^{80} \text{ 😊} \\ 2^{160} \text{ 😊} \end{array} \right.$



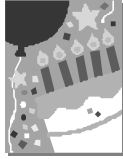
Lunghezza valore hash

- ❑ Nuovo attacco per trovare collisioni
 - Scelgo a caso diversi messaggi
 - Verifico se ottengo almeno due valori hash uguali
- ❑ Quanti messaggi per avere una buona probabilità di successo?
 - n numero dei diversi valori hash
 - t numero messaggi
 - ϵ probabilità di successo



Paradosso del compleanno

Quante persone scegliere a caso affinché, con probabilità ≥ 0.5 , ci siano almeno due con lo stesso compleanno?



Risposta: bastano 23 persone!



Paradosso del compleanno

Probabilità che tra t valori scelti a caso ed indipendentemente non ci siano valori uguali

$$\frac{365 \cdot 364 \cdot \dots \cdot (365 - t + 1)}{365^t} \begin{array}{l} \text{— casi favorevoli} \\ \text{— totale casi} \end{array}$$

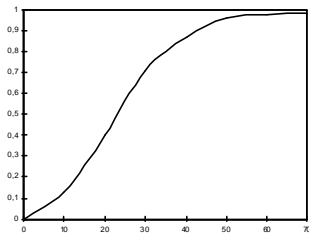
Probabilità che tra t valori scelti a caso ed indipendentemente almeno 2 sono uguali

$$1 - \frac{365 \cdot 364 \cdot \dots \cdot (365 - t + 1)}{365^t}$$



Paradosso del compleanno

Grafico di $1 - \frac{365 \cdot 364 \cdot \dots \cdot (365 - t + 1)}{365^t}$



$t = 23 \rightarrow \epsilon = 0.5073$
 $t = 100 \rightarrow \epsilon = 0.9999997$



Paradosso del compleanno

Scegliamo a caso elementi in un insieme di cardinalità n .

Quanti elementi scegliere se si vuole che la probabilità che ci siano almeno due elementi uguali sia ϵ ?

$$t \approx \sqrt{n \cdot 2 \ln \left(\frac{1}{1 - \epsilon} \right)}$$



Paradosso del compleanno

Scegliamo a caso 2 elementi z_1, z_2 in un insieme di cardinalità n .
Probabilità che sono diversi

$$\begin{aligned} \text{Prob}(z_2 \neq z_1) &= 1 - \text{Prob}(z_2 = z_1) \\ &= 1 - 1/n \end{aligned}$$



Paradosso del compleanno

Scegliamo a caso 2 elementi z_1, z_2 in un insieme di cardinalità n .
Probabilità che sono diversi

$$\text{Prob}(z_2 \neq z_1) = 1 - 1/n$$

Scegliamo a caso 3 elementi z_1, z_2, z_3 in un insieme di cardinalità n .
Probabilità che sono diversi

$$\text{Prob}(z_3 \neq z_1 \wedge z_3 \neq z_2 \mid z_2 \neq z_1) \cdot \text{Prob}(z_2 \neq z_1) = (1 - 2/n) (1 - 1/n)$$



Paradosso del compleanno

Scegliamo a caso z_1, z_2, \dots, z_t in un insieme di cardinalità n .
Probabilità che sono diversi

$$\text{Prob}(z_1, z_2, \dots, z_t \text{ diversi}) = (1 - (t-1)/n) \cdot \dots \cdot (1-2/n) \cdot (1 - 1/n)$$

Per piccoli x abbiamo $1-x \approx e^{-x}$

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots$$



Paradosso del compleanno

Scegliamo a caso z_1, z_2, \dots, z_t in un insieme di cardinalità n .
Probabilità che sono diversi

$$\text{Prob}(z_1, z_2, \dots, z_t \text{ diversi}) = \prod_{i=1}^{t-1} \left(1 - \frac{i}{n}\right) \approx \prod_{i=1}^{t-1} \left(e^{-i/n}\right)$$

Per piccoli x abbiamo $1-x \approx e^{-x}$

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots$$



Paradosso del compleanno

Scegliamo a caso z_1, z_2, \dots, z_t in un insieme di cardinalità n .
Probabilità che sono diversi

$$\begin{aligned} \text{Prob}(z_1, z_2, \dots, z_t \text{ diversi}) &= \prod_{i=1}^{t-1} \left(1 - \frac{i}{n}\right) \\ &\approx \prod_{i=1}^{t-1} \left(e^{-i/n}\right) \\ &\approx e^{-\frac{t(t-1)}{2n}} \end{aligned}$$

$$\begin{aligned} \varepsilon \triangleq \text{Prob}(\text{almeno una collisione tra } z_1, z_2, \dots, z_t) \\ \approx 1 - e^{-\frac{t(t-1)}{2n}} \end{aligned}$$



Paradosso del compleanno

$$\varepsilon \triangleq \text{Prob}(\text{almeno una collisione tra } z_1, z_2, \dots, z_t)$$

$$1 - \varepsilon \approx e^{-\frac{t(t-1)}{2n}}$$

$$\frac{-\ln(1-\varepsilon)}{2n} \approx \ln(1-\varepsilon)$$

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

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



Paradosso del compleanno

Scegliamo a caso elementi in un insieme di cardinalità n .
Quanti elementi scegliere se si vuole che la probabilità che ci siano almeno due elementi uguali sia ε ?

$$t \approx \sqrt{n \cdot 2 \ln \left(\frac{1}{1-\varepsilon}\right)}$$

Se $\varepsilon = 0.5$ allora $t \approx 1.17\sqrt{n}$

Applicazione: $n = 365$ e $\varepsilon = 0.5$ allora $t = 22.3$

Che relazione c'è con le funzioni hash?



Attacco del compleanno

- Scegliere t elementi a caso e calcolarne i valori hash.
- Quanti elementi scegliere per avere almeno una collisione?
(Assumiamo preimmagini uguali, caso migliore per chi sceglie h)
- Per una fissata probabilità ε , t è circa \sqrt{n}

Se $n = 2^{80}$ allora $t \approx 2^{40}$ 😞

Se $n = 2^{160}$ allora $t \approx 2^{80}$ 😊

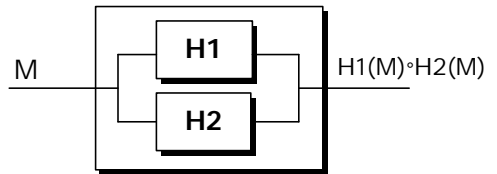


Sicurezza Hash 128 bit

- Costo di un attacco per computare collisioni
 $2^{64} \approx 2 \cdot 10^{19}$ valutazioni della funzione
- Attacco <1 mese e \$10.000.000
P. van Orschot e M. Wiener [1994]
- Si ipotizza che il costo dimezza ogni 18 mesi



Composizione funzioni hash

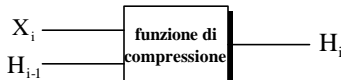


Trovare una collisione per $H(M)=H1(M) \cdot H2(M)$ significa trovare una collisione sia per H1 che per H2



Modello generale per funzioni hash iterate

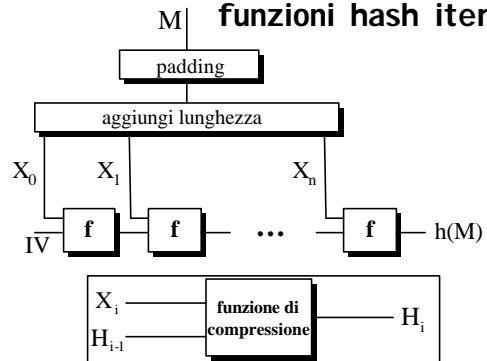
Funzione hash input taglia fissa \Rightarrow taglia arbitraria
Input M. Padding ed aggiunta della lunghezza di M.
Si ottiene un messaggio con blocchi di taglia uguale $X_1 X_2 \dots X_n$



H_0 è una costante iniziale
Computazione di $\dots H_i = f(X_i, H_{i-1}) \dots$
Valore hash $H_n = f(X_n, H_{n-1})$ } **computazione del valore hash**



Modello generale funzioni hash iterate



Sicurezza forte per f \Leftrightarrow Sicurezza forte per h

Idea della prova: una collisione per h implica una collisione per f
Supponiamo di aver calcolato $M \neq M'$ tali che $h(M) = h(M')$

Dopo il padding e l'aggiunta della lunghezza otteniamo
 $X_1 X_2 \dots X_n$ $X'_1 X'_2 \dots X'_m$

Assumiamo $n = m$

Nota che deve risultare $M \neq M' \Rightarrow X_1 X_2 \dots X_n \neq X'_1 X'_2 \dots X'_n$

Sia i tale che $f(X_i, H_{i-1}) = f(X'_i, H'_{i-1})$ ma $(X_i, H_{i-1}) \neq (X'_i, H'_{i-1})$

Trovata collisione per f !

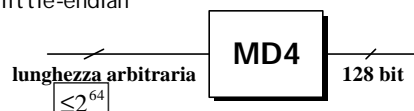
Se $n \neq m$, trovata collisione data la codifica finale della lunghezza

$$f(X_n, H_{n-1}) = f(X'_n, H'_{n-1}) \quad (X_n, H_{n-1}) \neq (X'_n, H'_{n-1})$$



Funzione hash MD4

- Progettata nel 1990 da Ron Rivest
- MD da **M**essage **D**igest
- Operazioni efficienti su architetture 32 bit little-endian





Obiettivi di progettazione per MD4

- ❑ **Sicurezza forte:** computazionalmente difficile trovare 2 messaggi con lo stesso valore hash
- ❑ **Sicurezza diretta:** sicurezza non basata su problemi teorici difficili computazionalmente
- ❑ **Velocità:** algoritmo adatto per implementazioni software molto veloci
- ❑ **Semplicità e Compattezza:** semplice da descrivere e da implementare, nessun uso di tabelle e di complesse strutture dati



MD4: padding del messaggio

- ❑ MD4 processa il messaggio in blocchi di 512 bit
Ogni blocco consta di 16 parole di 32 bit
- ❑ M messaggio originario di b bit \Rightarrow padding

$$M' = \boxed{M \underbrace{100\dots 0}_b}$$

(447-b) mod 512 bit 64 bit

M' consta di un numero di bit multiplo di 512, ovvero di un numero di parole N multiplo di 16



MD4: operazioni

- ❑ Funzioni definite su parole di 32 bit:
 - round 1: $f(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$ (if X then Y else Z)
 - round 2: $g(X,Y,Z) = (X \wedge Z) \vee (Y \wedge Z) \vee (X \wedge Y)$ (2 su 3)
 - round 3: $h(X,Y,Z) = X \oplus Y \oplus Z$ (bit di parità)

X	Y	Z	f	g	h
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	1

- ❑ Ogni round consiste di 16 operazioni
- ❑ $X+Y$ somma modulo 2^{32} , $X \ll s$ shift ciclico a sinistra di s bit

MD4

A \leftarrow 67452301; B \leftarrow efcdab89; C \leftarrow 98badcfe; D \leftarrow 10325476;
 for j=0 to N/16-1 do
 AA \leftarrow A; BB \leftarrow B; CC \leftarrow C; DD \leftarrow D;

round 1	round 2	round 3
A \leftarrow (A+f(B,C,D)+X[0]) \ll 3	A \leftarrow (A+g(B,C,D)+X[0]+p) \ll 3	A \leftarrow (A+h(B,C,D)+X[0]+q) \ll 3
D \leftarrow (D+f(A,B,C)+X[1]) \ll 7	D \leftarrow (D+g(A,B,C)+X[1]+p) \ll 5	D \leftarrow (D+h(A,B,C)+X[1]+q) \ll 9
C \leftarrow (C+f(D,A,B)+X[2]) \ll 11	C \leftarrow (C+g(D,A,B)+X[2]+p) \ll 9	C \leftarrow (C+h(D,A,B)+X[2]+q) \ll 11
B \leftarrow (B+f(C,D,A)+X[3]) \ll 19	B \leftarrow (B+g(C,D,A)+X[3]+p) \ll 13	B \leftarrow (B+h(C,D,A)+X[3]+q) \ll 15
A \leftarrow (A+f(B,C,D)+X[4]) \ll 3	A \leftarrow (A+g(B,C,D)+X[4]+p) \ll 3	A \leftarrow (A+h(B,C,D)+X[4]+q) \ll 3
D \leftarrow (D+f(A,B,C)+X[5]) \ll 7	D \leftarrow (D+g(A,B,C)+X[5]+p) \ll 5	D \leftarrow (D+h(A,B,C)+X[5]+q) \ll 9
C \leftarrow (C+f(D,A,B)+X[6]) \ll 11	C \leftarrow (C+g(D,A,B)+X[6]+p) \ll 9	C \leftarrow (C+h(D,A,B)+X[6]+q) \ll 11
B \leftarrow (B+f(C,D,A)+X[7]) \ll 19	B \leftarrow (B+g(C,D,A)+X[7]+p) \ll 13	B \leftarrow (B+h(C,D,A)+X[7]+q) \ll 15
A \leftarrow (A+f(B,C,D)+X[8]) \ll 3	A \leftarrow (A+g(B,C,D)+X[8]+p) \ll 3	A \leftarrow (A+h(B,C,D)+X[8]+q) \ll 3
D \leftarrow (D+f(A,B,C)+X[9]) \ll 7	D \leftarrow (D+g(A,B,C)+X[9]+p) \ll 5	D \leftarrow (D+h(A,B,C)+X[9]+q) \ll 9
C \leftarrow (C+f(D,A,B)+X[10]) \ll 11	C \leftarrow (C+g(D,A,B)+X[10]+p) \ll 9	C \leftarrow (C+h(D,A,B)+X[10]+q) \ll 11
B \leftarrow (B+f(C,D,A)+X[11]) \ll 19	B \leftarrow (B+g(C,D,A)+X[11]+p) \ll 13	B \leftarrow (B+h(C,D,A)+X[11]+q) \ll 15
A \leftarrow (A+f(B,C,D)+X[12]) \ll 3	A \leftarrow (A+g(B,C,D)+X[12]+p) \ll 3	A \leftarrow (A+h(B,C,D)+X[12]+q) \ll 3
D \leftarrow (D+f(A,B,C)+X[13]) \ll 7	D \leftarrow (D+g(A,B,C)+X[13]+p) \ll 5	D \leftarrow (D+h(A,B,C)+X[13]+q) \ll 9
C \leftarrow (C+f(D,A,B)+X[14]) \ll 11	C \leftarrow (C+g(D,A,B)+X[14]+p) \ll 9	C \leftarrow (C+h(D,A,B)+X[14]+q) \ll 11
B \leftarrow (B+f(C,D,A)+X[15]) \ll 19	B \leftarrow (B+g(C,D,A)+X[15]+p) \ll 13	B \leftarrow (B+h(C,D,A)+X[15]+q) \ll 15

A \leftarrow A+AA; B \leftarrow B+BB; C \leftarrow C+CC; D \leftarrow D+DD;
output (A, B, C, D)

Costanti
 p=5a827999
 q=6ed9eba1



Sicurezza di MD4

- MD4 è stato oggetto di molti attacchi
- crittoanalisi dei primi 2 round: Merkle ha provato che è facile trovare collisioni con round 3 omissso
- crittoanalisi degli ultimi 2 round: den Boer e Bosselaers [Crypto '91] hanno trovato collisioni con round 1 omissso
- Settembre 1995: Dobbertin [FSE '96] ha trovato collisioni per MD4 con un PC in pochi secondi



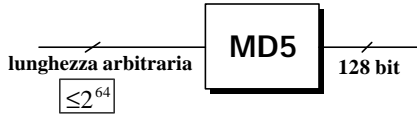
Test vectors MD4

Messaggio	MD4
"" (stringa vuota)	31d6cfe0d16ae931b73c59d7e0c089c0
"a"	bde52cb31de33e46245e05fbd6dfb24
"abc"	a448017aaf21d8525fc10ae87aa6729d
"message digest"	d9130a8164549fe818874806e1c7014b
"abcdefghijklmnopqrstuvwxyz"	d79e1c308aa5bbcddeea8ed63df412da9
"A...Za...z0...9"	043f8582f241db351ce627e153e7f0e4
8 volte "1234567890"	e33b4ddc9c38f2199c3e7b164fcc0536



MD5

- ❑ Progettato nel 1991 da Ron Rivest
- ❑ MD da Message Digest
- ❑ Operazioni efficienti su architetture 32 bit little-endian



Funzioni Hash

36



MD5	MD4
4 round con 4 · 16 operazioni	3 round con 3 · 16 operazioni
4 funzioni logiche	3 funzioni logiche
64 costanti additive	2 costanti additive
ogni passo aggiunge il risultato del passo precedente	non accade



MD5: padding del messaggio

- ❑ MD5 processa il messaggio in blocchi di 512 bit
- Ogni blocco consta di 16 parole di 32 bit
- ❑ M messaggio originario di b bit \Rightarrow padding

$$M' = \underbrace{M \ 100 \dots 0 \ b}_{(447-b) \bmod 512 \text{ bit} \ 64 \text{ bit}}$$

- ❑ M' consta di un numero di bit multiplo di 512, ovvero di un numero di parole N multiplo di 16

Funzioni Hash

38



MD5: operazioni

- ❑ Funzioni definite su parole di 32 bit:
 - round 1: $F(X,Y,Z) = (X \wedge Y) \vee (\neg X) \wedge Z$ (if X then Y else Z)
 - round 2: $G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$ (if Z then X else Y)
 - round 3: $H(X,Y,Z) = X \oplus Y \oplus Z$ (bit di parità)
 - round 4: $I(X,Y,Z) = Y \oplus (X \vee (\neg Z))$ (nuova funzione)

X	Y	Z	F	G	H	I
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0

- ❑ Ogni round consiste di 16 operazioni [ABCD.k.s.1]
 - $A \leftarrow B + ((A + W(B,C,D) + X[k] + T[i])) \ll s$
 - W è la funzione del round

Funzioni Hash

39



Costanti T[1...64]

T[i] = primi 32 bit di $|\sin(i)| = e^{2^{32}} \cdot |\sin(i)| \hat{u}$ (i in radianti)

1	d76aa478	17	f61e2562	33	fffa3942	49	f4292244
2	e8c7b756	18	c040b340	34	8771f681	50	432aff97
3	242070db	19	265e5a51	35	6d9d6122	51	ab9423a7
4	c1bdceee	20	e9b6c7aa	36	fde5380c	52	fc93a039
5	f57c0faf	21	d62f105d	37	a4beeaa4	53	65b59c3
6	4787c62a	22	02441453	38	4bdecfa9	54	8f0ccc92
7	a8304613	23	d8a1e681	39	f6bb4b60	55	ffeef47d
8	fd469501	24	e7d3fbc8	40	bebfb70	56	85845dd1
9	698098d8	25	21e1cde6	41	289b7ec6	57	6fa87e4f
10	8b44f7af	26	c33707d6	42	eaal27fa	58	fe2ce6e0
11	ffff5bb1	27	f4d50d87	43	d4ef3085	59	a3014314
12	895cd7be	28	455a14ed	44	04881d05	60	4e0811a1
13	6b901122	29	a9e3e905	45	d94d039	61	f7537e82
14	fd987193	30	fcfa3f8	46	e6db99e5	62	bd3af235
15	a679438e	31	676f02d9	47	1fa27c8	63	2ad7d2bb
16	49b40821	32	8d2a4c8a	48	c4ac5665	64	eb86d391

Funzioni Hash

40



MD5

A ← 0123456; B ← 89abcdef; C ← fdecba98; D ← 76543210;

for i = 0 to N/16-1 do
 for j = 0 to 15 do X[j] ← M'[16i+j]
 AA ← A; BB ← B; CC ← C; DD ← D;

round 1: [ABCD. 0.7. 1] [DABC. 1.12. 2] [CDAB. 2.17. 3] [BCDA. 3.22. 4]
 [ABCD. 4.7. 5] [DABC. 5.12. 6] [CDAB. 6.17. 7] [BCDA. 7.22. 8]
 [ABCD. 8.7. 9] [DABC. 9.12.10] [CDAB.10.17.11] [BCDA.11.22.12]

round 2: [ABCD.12.7.13] [DABC.13.12.14] [CDAB.14.17.15] [BCDA.15.22.16]
 [ABCD. 1.5.17] [DABC. 6. 9.18] [CDAB.11.14.17] [BCDA. 0.20.20]
 [ABCD. 5.5.22] [DABC.10. 9.22] [CDAB.15.14.23] [BCDA. 4.20.24]
 [ABCD. 9.5.25] [DABC.14. 9.26] [CDAB. 3.14.27] [BCDA. 8.20.28]
 [ABCD.13.5.29] [DABC. 2. 9.30] [CDAB. 7.14.21] [BCDA.12.20.32]

round 3: [ABCD. 5.4.33] [DABC. 8.11.34] [CDAB.11.16.35] [BCDA.14.23.36]
 [ABCD. 14.37] [DABC. 4.11.38] [CDAB. 7.16.39] [BCDA.10.23.40]
 [ABCD.13.4.41] [DABC. 0.11.42] [CDAB. 3.16.43] [BCDA. 6.23.44]
 [ABCD. 9.4.45] [DABC.12.11.46] [CDAB.15.16.45] [BCDA. 2.23.48]
 [ABCD. 0.6.49] [DABC. 7.10.50] [CDAB. 5.15.51] [BCDA. 5.21. 5]
 [ABCD.12.6.53] [DABC. 3.10.54] [CDAB. 1.15.55] [BCDA. 1.21.56]
 [ABCD. 8.6.57] [DABC.15.10.58] [CDAB.13.15.59] [BCDA.13.21.60]
 [ABCD. 4.6.61] [DABC.11.10.62] [CDAB. 9.15.63] [BCDA. 9.21.64]

A ← A·A; B ← B·BB; C ← C·CC; D ← D·DD;

output: (A, B, C, D)



Avalanche effect nell'MD5

In MD5 ogni passo somma il valore precedente

Round 1 in MD5

$$A \leftarrow B + ((A + F(B,C,D) + X[0] + T[1])) \ll 7$$

$$D \leftarrow A + ((D + F(A,B,C) + X[1] + T[2])) \ll 12$$

$$C \leftarrow D + ((C + F(D,A,B) + X[2] + T[3])) \ll 17$$

$$B \leftarrow C + ((B + F(C,D,A) + X[3] + T[4])) \ll 22$$

...

Round 1 in MD4:

$$A \leftarrow (A + f(B,C,D) + X[0]) \ll 3$$

$$D \leftarrow (D + f(A,B,C) + X[1]) \ll 7$$

$$C \leftarrow (C + f(D,A,B) + X[2]) \ll 11$$

$$B \leftarrow (B + f(C,D,A) + X[3]) \ll 19$$

...



Little-endian e Big-endian

Come si trasformano sequenze di byte in parole di 32 bit?
Conversione ambigua!

Sequenza byte B1, B2, B3, B4 nella parola W

Architetture **Little-endian** (come processori Intel 80xxx)
byte con indirizzo più basso è quello meno significativo
valore parola $W = 2^{24}B_4 + 2^{16}B_3 + 2^8B_2 + 2^0B_1$

Architetture **Big-endian** (come SUN SPARCstation)
byte con indirizzo più basso è quello più significativo
valore parola $W = 2^{24}B_1 + 2^{16}B_2 + 2^8B_3 + 2^0B_4$



Test vectors MD5

Messaggio	MD5
"" (stringa vuota)	d41d8cd98f00b204e9800998ecf8427e
"a"	0cc175b9c0f1b6a831c399e269772661
"abc"	900150983cd24fb0d6963f7d28e17f72
"message digest"	f96b697d7cb7938d525a2f31aaf161d0
"abcdefghijklmnopqrstuvwxyz"	c3fcd3d76192e4007dfb496cca67e13b
"A...Za...z0...9"	d174ab98d277d9f5a5611c2c9f419d9f
8 volte "1234567890"	57edf4a22be3c955ac49da2e2107b67a



SHS

- SHS per **Secure Hash Standard**
- SHA per **Secure Hash Algorithm**
- Standard del Governo americano dal 1993
- Modificato nel luglio 1994, denotato SHA-1
(unica differenza: aggiunta di uno shift nell'espansione dei blocchi)
- Operazioni efficienti su architetture 32 bit big-endian
- Stessi principi di MD4 ed MD5, ma più sicuro



SHA: padding del messaggio

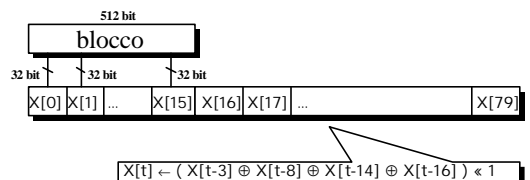
- SHA processa il messaggio in blocchi di 512 bit
Ogni blocco consta di 16 parole di 32 bit
- M messaggio originario di b bit \rightarrow padding

$$M' = \underbrace{M}_{(447-b) \text{ mod } 512 \text{ bit}} \underbrace{100\dots0}_{64 \text{ bit}}$$

- M' consta di un numero di bit multiplo di 512, ovvero di un numero di parole N multiplo di 16



Espansione blocco ed iterazioni



- 80 iterazioni
- Una parola ed una costante per ogni iterazione



Funzioni logiche di SHA

Funzione $F(t,X,Y,Z)$

round $t = 0, \dots, 19$: $F(t,X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$ (if X then Y else Z)
 round $t = 20, \dots, 39$: $F(t,X,Y,Z) = X \oplus Y \oplus Z$ (bit di parità)
 round $t = 40, \dots, 59$: $F(t,X,Y,Z) = (X \wedge Z) \vee (Y \wedge Z) \vee (X \wedge Y)$ (2 su 3)
 round $t = 60, \dots, 79$: $F(t,X,Y,Z) = Y \oplus X \oplus Z$ (bit di parità)

X	Y	Z	F(0...)	F(20...)	F(40...)	F(60...)
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

Funzioni Hash

48



Costanti additive di SHA

Costante additiva $K[t]$:

round $t = 0, \dots, 19$: 5a827999
 round $t = 20, \dots, 39$: 6ed9eba1
 round $t = 40, \dots, 59$: 8f1bbcdc
 round $t = 60, \dots, 79$: ca62c1d1

Funzioni Hash

49



```

A=67452310; B=efcdab89; C=98badcfe; D=10325476; E=c3d2e1f0;
for i = 0 to N/16-1 do
  for j = 0 to 15 do
    X[j] ← M [16i+j]
  for t = 16 to 79 do
    X[t] ← ( X[t-3] ⊕ X[t-8] ⊕ X[t-14] ⊕ X[t-16] ) ◀ 1
    AA ← A; BB ← B; CC ← C; DD ← D; EE ← E;
  for t=0 to 79 do
    TEMP ← (A◀5) + F(t,B,C,D) + E + X[t] + K[t]
    E ← D
    D ← C
    C ← (B◀30)
    B ← A
    A ← TEMP
  A ← A + AA; B ← B + BB; C ← C + CC; D ← D + DD; E ← E + EE;
output: (A, B, C, D, E)

```

SHA-1

espansione
 da 16 ad 80 parole,
 "◀1" non c'era in SHA

Funzioni Hash

50



SHA-256, SHA-512, SHA-384

- ❑ Hash di SHA-1 è 160 bit
 - Sicurezza contro attacco del compleanno 80 bit
- ❑ Lunghezza chiavi AES: 128, 192, 256
- ❑ Proposti nuovi SHA (12 ottobre 2000)
 - Lunghezza valore hash: 256, 512, 384 bit
 - Sicurezza attacco del compleanno 128, 256, 192 bit
- ❑ Draft di Federal Information Processing Standard (FIPS), gennaio 2001

Funzioni Hash

51



SHA-256, SHA-512, SHA-384

- ❑ Stessi principi di MD4, MD5, SHA-1
- ❑ SHA-256
 - Messaggio diviso in blocchi di 512 bit
 - Parole da 32 bit
- ❑ SHA-512
 - Messaggio diviso in blocchi di 1024 bit
 - Parole da 64 bit
- ❑ SHA-384
 - Valore hash = primi 384 bit di SHA-512, con costanti iniziali cambiate

Funzioni Hash

52



RIPEDM

- ❑ RACE Integrity Primitives Evaluation (RIPE)
 - progetto della Comunità Europea [1988-1992]
- ❑ Prima versione RIPEMD (hash di 128 bit)
 - Debolezze scoperte da H. Dobbertin [1995]
- ❑ RIPEMD-128 RIPEMD-160 [1996]
- ❑ Operazioni efficienti su architetture 32 bit little-endian
- ❑ Stessi principi di MD4, MD5, SHA-1
- ❑ **left line** e **right line**

Funzioni Hash

53



RIPEMD: padding del messaggio

- RIPEMD processa il messaggio in blocchi di 512 bit
- Ogni blocco consta di 16 parole di 32 bit
- M messaggio originario di b bit \Rightarrow padding

$$M' = \boxed{M \ 100\dots 0 \ b}$$

(447-b) mod 512 bit 64 bit

- M' consta di un numero di bit multiplo di 512, ovvero di un numero di parole N multiplo di 16



RIPEMD 160

- left line e right line
- 80 iterazioni
- Singola iterazione $j=0,1,\dots,79$

$$T \leftarrow ((A + F(j, B, C, D) + X[i+r(j)] + K_j) \ll S_j) + E$$

Funzione logica

Selezione parola valore in [0,15]

costante

shift



Funzioni logiche di RIPEMD 160

Funzione F(t,X,Y,Z)

- round $t = 0, \dots, 15$: $F(t, X, Y, Z) = X \oplus Y \oplus Z$ (bit di parità)
- round $t = 16, \dots, 31$: $F(t, X, Y, Z) = (X \wedge Y) \vee (\neg X) \wedge Z$ (if X then Y else Z)
- round $t = 32, \dots, 47$: $F(t, X, Y, Z) = (X \vee \neg Y) \oplus Z$
- round $t = 48, \dots, 63$: $F(t, X, Y, Z) = (X \wedge Z) \vee (\neg Z) \wedge Y$ (if Z then X else Y)
- round $t = 64, \dots, 79$: $F(t, X, Y, Z) = Y \oplus (Y \vee \neg Z)$

X	Y	Z	F(0...)	F(16...)	F(32...)	F(48...)	F(64...)
0	0	0	0	0	1	0	1
0	0	1	1	1	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	1
1	0	0	1	0	1	0	0
1	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0
1	1	1	1	1	0	1	0



Costanti

left line

right line

j	K _j		K' _j	
0...15	00000000	0	50a28be6	$2^{30} \sqrt[3]{2}$
16...31	5a827999	$2^{30} \sqrt{2}$	5c4dd124	$2^{30} \sqrt[3]{3}$
32...47	6ed9eba1	$2^{30} \sqrt{3}$	6d703ef3	$2^{30} \sqrt[3]{5}$
48...63	8f1bbcdc	$2^{30} \sqrt{5}$	7a6d76e9	$2^{30} \sqrt[3]{7}$
64...79	a953fd4e	$2^{30} \sqrt{7}$	00000000	0



H₀ ← 67452301 H₁ ← EFCDBA89 H₂ ← 98BADCFE H₃ ← 10325476 H₄ ← C3D2E1F0

for i = 0 to N-1 step 16 do

(A,B,C,D,E) ← (H₀,H₁,H₂,H₃,H₄)

(A',B',C',D',E') ← (H₀,H₁,H₂,H₃,H₄)

RIPEMD 160

for j = 0 to 79 do

T ← ((A + F(j,B,C,D) + X[i+r(j)] + K_j) « S_j) + E

A ← E; E ← D; D ← C « 10; C ← B; B ← T;

T ← ((A'+F(79-j,B',C',D') + X[i+r'(j)] + K'_j) « S'_j) + E'

A' ← E'; E' ← D'; D' ← C' « 10; C' ← B'; B' ← T';

T ← H₁ + C + D; H₁ ← H₂ + D + E; H₂ ← H₂ + E + A';

H₃ ← H₂ + A + B; H₄ ← H₀ + B + C; H₀ ← T;


output: (H₀,H₁,H₂,H₃,H₄)



Shift a sinistra S_j S'_j

j	X ₀	X ₁	X ₂	X ₃	X ₄	X ₅	X ₆	X ₇	X ₈	X ₉	X ₁₀	X ₁₁	X ₁₂	X ₁₃	X ₁₄	X ₁₅
0...15	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
16...31	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12
32...47	11	13	6	7	14	9	13	15	14	8	13	6	5	12	7	5
48...63	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12
64...79	9	15	5	11	6	8	13	12	5	12	13	14	11	8	5	6

j	X ₀	X ₁	X ₂	X ₃	X ₄	X ₅	X ₆	X ₇	X ₈	X ₉	X ₁₀	X ₁₁	X ₁₂	X ₁₃	X ₁₄	X ₁₅
0...15	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
16...31	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
32...47	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
48...63	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8
64...79	8	5	12	9	12	5	14	6	8	13	6	5	15	13	11	11




Selezione parola r r'

j	X ₀	X ₁	X ₂	X ₃	X ₄	X ₅	X ₆	X ₇	X ₈	X ₉	X ₁₀	X ₁₁	X ₁₂	X ₁₃	X ₁₄	X ₁₅
0...15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16...31	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
32...47	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12
48...63	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2
64...79	4	0	5	9	7	12	2	10	14	1	3	8	11	6	15	13

j	X ₀	X ₁	X ₂	X ₃	X ₄	X ₅	X ₆	X ₇	X ₈	X ₉	X ₁₀	X ₁₁	X ₁₂	X ₁₃	X ₁₄	X ₁₅
0...15	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
16...31	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
32...47	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
48...63	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14
64...79	12	15	10	4	1	5	8	7	6	2	13	14	0	3	9	11

Funzioni Hash 60




Selezione parola

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
r	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
p	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12

$p(j) = 9j+5 \pmod{16}$

	left line	right line
0...15	identità	π
16...31	ρ	$\rho \pi$
32...47	ρ^2	$\rho^2 \pi$
48...63	ρ^3	$\rho^3 \pi$
64...79	ρ^4	$\rho^4 \pi$


61



Test vectors RIPEMD-160 RIPEMD-128

Messaggio	RIPEMD-160	RIPEMD-128
** (stringa vuota)	9c1185a5c5e9fc54612808977e e8f548b2258d31	cdf26213a150dc3ecb610f18f6 b38b46
a	0bdc9d2d256b3ee9daae347be 6f4dc835a467ffe	86be7afa339d0fc7fc785e72f 578d33
abc	8eb208f7e05d987a9b044a8e9 8c6b087f15a0bfc	c14a12199c66e4ba8463b0f69 144c77
message digest	5d0c689ef49d2fae572b881b12 3a85ffa21595f3e	9e327b3d6e523062af1132d7 df9d1b8
abcdefghijklnopqrstuvwxyz	f71c27109c692c1b56bbdceb5b 9a2865b3708dbc	fd2aa607f71dc8f510714922b3 71834e
abcdcbcedefdefefghfghijhij jihijkiijklmklmnmnopnopq	12a053384a9c0c88e405a06c2 7dcf49ada62eb2b	af1aa0689d0fafa2ddc22e88b49 133a06
A..Za..z0..9	b0e20b6e3116640286ed3a87a 5713079b21f5189	d1e959eb179c911faea4624c60 e5c702
8 volte *1234567890*	9b752e45573d4b39f4dbd332 3cab82bf63326bf	3f45ef194732c2dbb2c4a2c76 9795fa3
1 milione di volte *a*	52783243c1697bdbel6d37f97 f68f08325dc1528	4a7f5723f954eba121c9d8f63 20431f

Funzioni Hash 63

- 
- ### Motivazioni progettuali
- Progettazione conservativa (come MD4)
 - Due esecuzioni parallele: left line e right line con la stessa logica ma con differenze:
 - Diverse costanti additive K_j e K'_j
 - Ordine delle funzioni $F(t, X, Y, Z)$
 - Ordine selezioni parole nel blocco
 - Probabilmente sarà possibile attaccare una delle 2 linee nel futuro, ma non entrambe
- Funzioni Hash 63




Motivazioni progettuali: permutazioni

Permutazione p: 2 parole vicine nella left line saranno lontane almeno 7 posizioni nella right line

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
r	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
p	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12

	left line	right line
0...15	identità	π
16...31	ρ	$\rho \pi$
32...47	ρ^2	$\rho^2 \pi$
48...63	ρ^3	$\rho^3 \pi$
64...79	ρ^4	$\rho^4 \pi$

64

- 
- ### Motivazioni progettuali: shift
- Shift <5 e >16 considerati deboli
 - Ogni messaggio è ruotato di valori diversi per ogni round, non sempre con la stessa parità
 - Gli shift applicati ad una parola non devono avere un pattern speciale (e.g., il totale non è divisibile per 32)
 - Non molti shift sono divisibili per 4
- Funzioni Hash 65



RIPEMD-256 e RIPEMD-320

- Valore hash più lungo, ma non più sicure
- 2 esecuzioni parallele di RIPEMD-128 e RIPEMD-160
 - Diversi valori iniziali
 - Combinazione delle 2 esecuzioni solo alla fine della funzione di compressione: scambio di *chaining variable* ad ogni iterazione
- Sicurezza non migliore
 - RIPEMD-128 \equiv RIPEMD-256
 - RIPEMD-128 \equiv RIPEMD-320

Funzioni Hash

66



Test vectors RIPEMD-256

Messaggio	RIPEMD - 256
** (stringa vuota)	02ba4c4e5f8ecd1877fc52d64d30e37a2d9774fb1e5d026380ae0168e3c5522d
"a"	f9333e45d857f5d90a91bab70a1eba0cfb1be4b0783c9acfdcb83a9134692925
"abc"	afb6d6228b9d8cbbcef5ca2d03e6dba10ac0bc7dcb4680e1e42d2e975459b65
"message digest"	87e971759a1ce47a514d5c914c392c9018c7c46bc144655544fcdf54a5070c0e
"abcdefghijklmnopqrstuvwxyz"	649d3034751ea216776bf9a18acc81bc7896118a51979e68782dd1fd97d8d5133
"abcdcbcedcdefdefgefghfghighijhijkijklkmlmnlmnomnopopq"	3843045583aac6c8c8d9128573e7a9809afb2a0f34ccc36ea9e72f16f6368e3f
"A...Za...z0...9"	5740a408ac16b720b84424ae931cbb1fe363d1d0b4017f1a89f7e6de77a0b8
8 volte "1234567890"	06fdcc7a409548aaf91368c06a6275b553e3f099bf0ea4edfd6778df89a890dd
1 milione di volte "a"	ac953744e10e31514c1504d4d8d7b677342e33399788296e43ae4850ce4f97978



Test vectors RIPEMD-320

Messaggio	RIPEMD - 320
** (stringa vuota)	22d65d5661536cdc75c1fd5cde7b41b9f27325ebc61e8557177d705a0ec880151c3a32a00899b8
"a"	ce78850638f92658a5a585097579926dda667a5716562cfc6f6b77f63542f99b04705d6970dff5d
"abc"	de4c01b3054f8930a79d09ae738e92301e5a17085bef9dc1b8d116713e74f82fa942d64cdbc4682d
"message digest"	3a8e28502ed45d422f68844f9dd316e7b98533fa3f2a91d29f84d425c88d6b4ef727df6ea7c0197
"abcdefghijklmnopqrstuvwxyz"	cabdb1810b92470a2093aa6bce05952c28348c4f3ff60841975166bb40ed234004b8824463e6b009
"abcdcbcedcdefdefgefghfghighijhijkijklkmlmnlmnomnopopq"	d034a7950cf722021ba4b84df769a5de206e259df4c9bb4a4268c0e935bbc7470a969c9d072a1ac
"A...Za...z0...9"	ed544940c86d67f250d232c30b7b3e5770e0c60c8cb9a4cafe3b11388af9920e1b99230b843c86a4
8 volte "1234567890"	557888a5f5ed8ed62ab66945c6d2a0a47ecd5341e915eb8feald0524955f825dc717e4a008ab2d42
1 milione di volte "a"	bdee37f4371e2064e68b0d862da16292ae36f40965e8c8509e63d1dbddec503e2b63eb9245bb66

Funzioni Hash

70



Implementazioni

Pentium 90 Mhz [1996]
Mbit/secondo

	Assembly	C
MD4	165.7	81.4
MD5	113.5	59.7
SHA-1	46.5	21.2
RIPEMD	82.5	44.0
RIPEMD-128	63.8	35.6
RIPEMD-160	39.8	19.3

Funzioni Hash

69



Implementazioni su Pentium

- Implementazioni 80x86 ottimizzate
- Pentium 90 Mhz
- Codice e dati nella cache (8k)
- A. Bosselaers
- 1996, 1997

	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

Funzioni Hash

70



Implementazioni

- Crypto++
- Celeron 850MHz
- Windows 2000

MD5	100,738
SHA-1	48,462
SHA-256	24,746
SHA-512	8,246
RIPEMD-160	30,725
DES	12,871
DES triplo	4,748
AES	30,325

Mbyte/sec
2²⁰ byte

RSA 1024 firma	10,29
RSA 1024 verifica	0,30
DSA 1024 firma	5,50
DSA 1024 verifica	2,27
DSA 1024 firma	6,38

Millisecondi
per operazione

71



Altre funzioni Hash

- ❑ **Snefru**, Ralph Merkle [1990], 128 oppure 256 bit
- ❑ **N-hash**, Nippon Telephone and Telegraph [1990], 128 bit
- ❑ **HAVAL**, Zheng-Pieprzyk-Seberry [1992] 128-160-192-224-256 bit
- ❑ **FFT-hash I**, C. Schnorr [1991], rotto dopo pochi mesi
- ❑ **FFT-hash II**, C. Schnorr [1992], rotto dopo poche settimane
- ❑ ...



Funzioni Hash basate su cifrari a blocchi

Se è disponibile una implementazione di un cifrario a blocchi ...

- ❑ Cifrario a blocchi $E_K(\cdot)$ per input ad n bit
- ❑ Funzione g che da n bit produce una chiave
- ❑ $M'_1 \dots M'_t$ è il messaggio M con eventuale padding
- ❑ H_0 è una costante predefinita, H_i è il valore hash
- ❑ $H_i = E_{g(H_{i-1})}(M'_i) \oplus M'_i$ [Matyas-Meyer-Oseas]
- ❑ $H_i = E_{g(H_{i-1})}(M'_i) \oplus M'_i \oplus H_{i-1}$ [Miyaguchi-Preneel]
- ❑ $H_i = E_{M'_i}(H_{i-1}) \oplus H_{i-1}$ [Davies-Meyer]



MDC-2

- ❑ Se si usa DES l'output è solo 64 bit!
- ❑ *Manipulation Detection Code* con 2 cifrature per blocco di input, denotato **MDC-2** (output con DES = 128 bit)
- ❑ $M'_1 \dots M'_t$ è il messaggio M con padding in blocchi di n bit
- ❑ H_0, H'_0 sono costanti predefinite
- ❑ H_i, H'_i è il valore hash

