

Funzioni Hash

- 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

Funzioni Hash 0

Firme digitali e Funzioni hash

Problema: firma digitale di messaggi grandi

Soluzione naïve: 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

Funzioni Hash 1

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 2

Funzioni Hash: Proprietà

- Facili da calcolare
- ? ... poi?

Funzioni Hash 3

Un possibile attacco

- prepara 2 versioni di un contratto M ed M'
 - M è favorevole ad Alice
 - M' è sfavorevole ad Alice
- M' viene modificato a caso (piccoli cambiamenti come aggiunta spazi: 32 possibilità sono 2^{32} messaggi!) finchè
 $h(M) = h(M')$
- Alice firma $M \implies Firma_k(h(M))$
- ha quindi la firma di $M' \implies Firma_k(h(M'))$

Funzioni Hash 4

Esempio di lettera fraudolenta

Caro Massimo,

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

.....

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

.....

Funzioni Hash 5

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 $y = h(M)$

Funzioni Hash 6

Sicurezza forte \Rightarrow 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 $\geq 1/2$

Funzioni Hash 7

Sicurezza forte \Rightarrow 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

Funzioni Hash 8

Sicurezza forte \Rightarrow One-way

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

Funzioni Hash 9

One-way e Sicurezza forte

$g: \{0,1\}^* \rightarrow \{0,1\}^b$ funzione hash, prop. sicurezza forte

$$h(x) = \begin{cases} 1^{\circ x} & \text{se } |x|=b \text{ bit} \\ 0^{\circ g(x)} & \text{altrimenti} \end{cases}$$

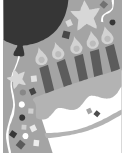
$h: \{0,1\}^* \rightarrow \{0,1\}^{b+1}$ è una funzione hash

- sicurezza forte
- non è "sempre" one-way

Funzioni Hash 10

Paradosso del compleanno

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




Funzioni Hash 11

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!



Funzioni Hash 12

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 ?

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

Funzioni Hash 13

Paradosso del compleanno

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

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

Funzioni Hash 14

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

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

Funzioni Hash 15

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 $1-x \approx e^{-x}$

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

Funzioni Hash 16

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 $1-x \approx e^{-x}$

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

Funzioni Hash 17

Paradosso del compleanno

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

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

$$\approx \prod_{i=1}^{t-1} \left(e^{-i/n}\right)$$

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

$\epsilon \triangleq \text{Prob}(\text{almeno una collisione tra } z_1, z_2, \dots, z_t)$
 $\approx 1 - e^{-t(t-1)/2n}$

Funzioni Hash 18

Paradosso del compleanno

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

$$1 - \epsilon \approx e^{-t(t-1)/(2n)}$$

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

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

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

Funzioni Hash 19

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 ?

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

Se $\epsilon = 0.5$ allora $t \approx 1.17\sqrt{n}$
 Applicazione: $n = 365$ e $\epsilon = 0.5$ allora $t = 22.3$
 Che relazione c'è con le funzioni hash?

Funzioni Hash 20

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 $n = 2^{40}$ allora $t \approx 2^{20}$ 😞

○ Se $n = 2^{128}$ allora $t \approx 2^{64}$ 😊

Funzioni Hash 21

Modello generale per funzioni hash iterate

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

```

    graph LR
      X1 --> F[funzione di compressione]
      H0[H0] --> F
      F --> H1[H1]
      H1 --> F
      F --> Hn[Hn]
    
```

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

Funzioni Hash 22

Sicurezza forte per $f \implies$ 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

$$X_1 X_2 \dots X_n \quad X'_1 X'_2 \dots X'_m$$

Assumiamo $n = m$

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

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

Funzioni Hash 23

Funzione hash MD4

- Progettata nel 1990 da Ron Rivest
- MD da Message Digest
- Operazioni efficienti su architetture 32 bit little-endian

Funzioni Hash 24

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

Funzioni Hash 25

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

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

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

Funzioni Hash 26

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 « s shift ciclico a sinistra di s bit

Funzioni Hash 27

MD4

```

A=67452301; B=efcdab89; C=98badcfe; D=10325476;
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
    A=(A+f(B,C,D)+X[0])<<3
    D=(D+f(A,B,C)+X[1])<<7
    C=(C+f(D,A,B)+X[2])<<11
    B=(B+f(C,D,A)+X[3])<<19
    A=(A+f(B,C,D)+X[4])<<3
    D=(D+f(A,B,C)+X[5])<<7
    C=(C+f(D,A,B)+X[6])<<11
    B=(B+f(C,D,A)+X[7])<<19
    A=(A+f(B,C,D)+X[8])<<3
    D=(D+f(A,B,C)+X[9])<<7
    C=(C+f(D,A,B)+X[10])<<11
    B=(B+f(C,D,A)+X[11])<<19
    A=(A+f(B,C,D)+X[12])<<3
    D=(D+f(A,B,C)+X[13])<<7
    C=(C+f(D,A,B)+X[14])<<11
    B=(B+f(C,D,A)+X[15])<<19
    round 2
    A=(A+g(B,C,D)+X[0]+p)<<3
    D=(D+g(A,B,C)+X[4]+p)<<5
    C=(C+g(D,A,B)+X[8]+p)<<9
    B=(B+g(C,D,A)+X[12]+p)<<13
    A=(A+g(B,C,D)+X[1]+p)<<3
    D=(D+g(A,B,C)+X[5]+p)<<5
    C=(C+g(D,A,B)+X[9]+p)<<9
    B=(B+g(C,D,A)+X[13]+p)<<13
    A=(A+g(B,C,D)+X[2]+p)<<3
    D=(D+g(A,B,C)+X[6]+p)<<5
    C=(C+g(D,A,B)+X[10]+p)<<9
    B=(B+g(C,D,A)+X[14]+p)<<13
    A=(A+g(B,C,D)+X[3]+p)<<3
    D=(D+g(A,B,C)+X[7]+p)<<5
    C=(C+g(D,A,B)+X[11]+p)<<9
    B=(B+g(C,D,A)+X[15]+p)<<13
    round 3
    A=(A+h(B,C,D)+X[0]+q)<<3
    D=(D+h(A,B,C)+X[8]+q)<<9
    C=(C+h(D,A,B)+X[4]+q)<<11
    B=(B+h(C,D,A)+X[12]+q)<<15
    A=(A+h(B,C,D)+X[1]+q)<<3
    D=(D+h(A,B,C)+X[5]+q)<<9
    C=(C+h(D,A,B)+X[9]+q)<<11
    B=(B+h(C,D,A)+X[13]+q)<<15
    A=(A+h(B,C,D)+X[11]+q)<<3
    D=(D+h(A,B,C)+X[10]+q)<<9
    C=(C+h(D,A,B)+X[6]+q)<<11
    B=(B+h(C,D,A)+X[14]+q)<<15
    A=(A+h(B,C,D)+X[1]+q)<<3
    D=(D+h(A,B,C)+X[9]+q)<<9
    C=(C+h(D,A,B)+X[5]+q)<<11
    B=(B+h(C,D,A)+X[13]+q)<<15
    A=A+AA; B=B+BB; C=C+CC; D=D+DD;
  output: (A, B, C, D)
    
```

Funzioni Hash 28

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

Funzioni Hash 29

MD5

- Progettato nel 1991 da Ron Rivest
- MD da **M**essage **D**igest
- Operazioni efficienti su architetture 32 bit little-endian

Funzioni Hash 30

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

Funzioni Hash 31

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 \implies padding

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

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

Funzioni Hash 32

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	1
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	0
1	1	1	1	1	1	0

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

Funzioni Hash 33

Costanti T[1...64]

$T[i] = \text{primi } 32 \text{ bit di } |\sin(i)| = \lfloor 2^{32} \cdot |\sin(i)| \rfloor$ (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	a4beea44	53	655b59c3
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	d9d4d039	61	f7537e82
14	fd987193	30	fcfa3f8	46	e6db99e5	62	bd3af235
15	a679438e	31	676f02d9	47	1fa27cf8	63	2ad7d2bb
16	49b40821	32	8d2a4c8a	48	c4ac5665	64	eb86d391

Funzioni Hash 34

MD5

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

```

for i = 0 to N/16-1 do
  for j = 0 to 15 do
    X[i] = M[i...16+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 11.14.19] [BCDA 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 1.4.37] [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]
    round 4 { [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+AA, B=B+BB, C=C+CC, D=D+DD;
  output: (A, B, C, D);
    
```

Funzioni Hash 35

Avalanche effect nell'MD5

In MD5 ogni passo somma il valore precedente

Round 1 in MD5

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

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

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

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

...

Round 1 in MD4:

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

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

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

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

...

Funzioni Hash 36

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 $W=2^{24}B4 + 2^{16}B3 + 2^8B2 + 2^0B1$

Architetture **Big-endian** (come SUN SPARCstation)
byte con indirizzo più basso è quello più significativo
valore parola $W=2^{24}B1 + 2^{16}B2 + 2^8B3 + 2^0B4$

Funzioni Hash 37

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

Funzioni Hash 38

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 \implies 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 39

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 40

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 41

SHA-1

```

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[i...16+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)
    
```

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

Funzioni Hash 42

Velocità di MD4, MD5 e SHA-1

PC 33MHz 486SX, megabytes/secondo

MD4	236
MD5	174
SHA-1	75

Funzioni Hash 43

Altre funzioni Hash

- **Snefru**, Ralph Merkle [1990], 128 oppure 256 bit
- **N-hash**, Nippon Telephone and Telegraph [1990], 128 bit
- **RIPE-MD**, progetto RIPE Comunità Europea [1992], 128 bit
 - trovate debolezze da Dobbertin [1995]
 - **RIPE-MD 160** [1996] e **RIPE-MD 128**
- **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 44

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
- $M'_1 \dots M'_i$ 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]
- $H_i = E_{g(H_{i-1})}(M'_i) \oplus M'_j \oplus H_{i-1}$ [Miyaguchi-Preneel]
- $H_i = E_{M'_i}(H_{i-1}) \oplus H_{i-1}$ [Davies-Meyer]

Funzioni Hash 45

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)
- $M'_1 \dots M'_i$ è il messaggio M con padding in blocchi di n bit
- H_0, \tilde{H}_0 sono costanti predefinite
- H_i, \tilde{H}_i è il valore hash

Funzioni Hash 46

Message Authentication Code (MAC)

messaggio M → calcolo del MAC → MAC(M,K)

chiave segreta K

Applicazioni: Verifica autenticità del messaggio M
Integrità dei dati

Proprietà: facile da calcolare ☺
difficile derivare K da M e MAC(M,K)
difficile produrre MAC(M,K) senza sapere K per un M anche conoscendo vari M_i e $MAC(M_i, K)$

Funzioni Hash 47

CBC-MAC

- CBC sta per **C**ipher **B**lock **C**haining (con $IV=0$)
- Uno dei più usati (FIPS PUB 113 e ANSI X9.17)
- Testo $X = X_1 X_2 \dots X_n$ diviso in blocchi di 64 bit

Valore troncato: da 16 a 64 bit più a sinistra

Funzioni Hash 48

MAC basati su Funzioni Hash

- MAC di M con chiave K usando la funzione hash H
- $H(K, M)$
 - potrebbe aggiungere blocchi ad M ottenendo $H(K, M')$
 - Possibile soluzione: $H(K, L, M)$ con L lunghezza di M
- $H(M, K)$ Se H segue il modello generale allora la dipendenza da K è limitata solo alla computazione finale
- $H(K, M, K)$ o meglio $H(K_1, M, K_2)$
- $H(K, H(K, M))$
- $E_K(H(M))$ dove $E_K(\cdot)$ è un cifrario a blocchi

Funzioni Hash 49