COMPUTER FORENSICS

Lezione 10: Protocolli Crittografici

Funzioni di hash

(2ª parte)

A.A. 2021/22

Dott. Lorenzo LAURATO

Nella puntata precedente…

La crittologia:

- **scienza che si occupa della comunicazione in forma sicura e di solito segreta.**

Crittografia
studio e applicazione dei principi e delle tecniche per rendere l'informazione inintelligibile a tutti tranne che al destinatario

VS

Crittoanalisi
scienza e arte di risolvere i crittosistemi per recuperare l'informazione nascosta

Nella puntata precedente…
La crittografia: i tre stadi

- **Primo stadio**
  - Dalle prime civiltà fino al secolo scorso
  - Algoritmi sviluppati e implementati “a mano”
- **Secondo stadio**
  - Seconda guerra mondiale
  - Apparizione delle prime macchine cifranti
- **Terzo stadio**
  - Ultimi 50 anni
  - Utilizzo di computer
  - Fondamenti matematici

Nella puntata precedente…
La crittografia: primitive

- I protocolli di basano su una serie di protocolli più semplici detti primitive crittografiche:
  - risolvono problemi “facili”
  - possono essere usate per risolvere problemi più complessi

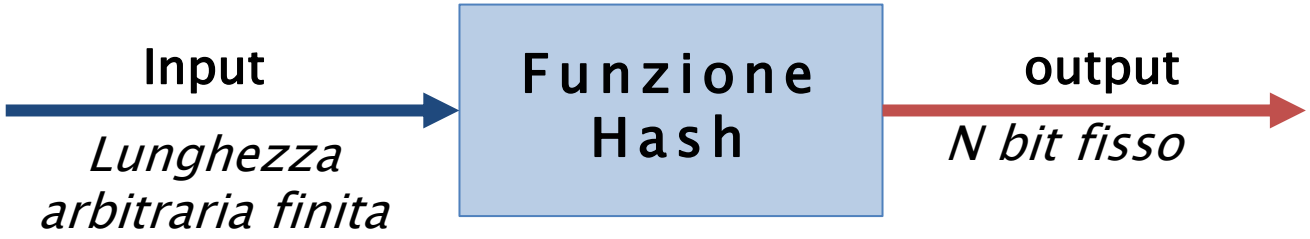
- risolvono i seguenti problemi:
  - **Cifratura**: *cifrari simmetrici o asimmetrici o a chiave pubblica;*
  - **Autenticazione ed integrità**: *Funzioni Hash e MAC*
  - **Firme digitali**
  - **Altro**: *generazione di numeri pseudo-casuale,prove zero-knowledge, etc.*

Hash

- MD4/5 e SHA1

Funzione Hash

- il valore hash **h(M)** è una rappresentazione non ambigua e non falsificabile del «*messaggio M*»



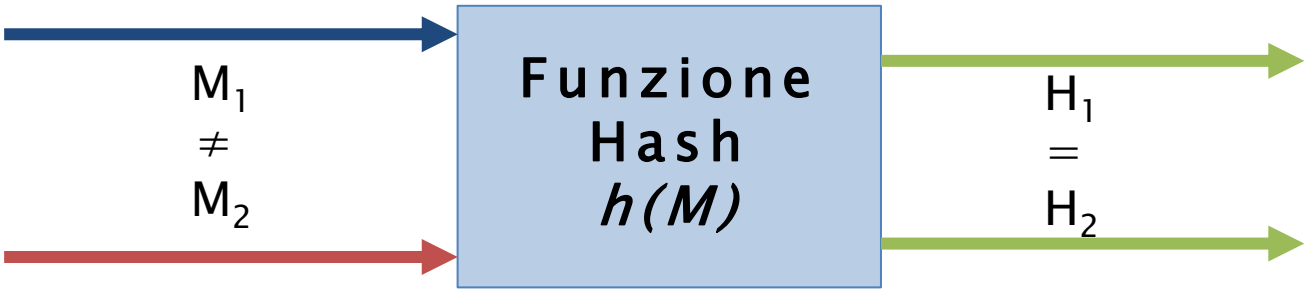
Integrità dei dati

Funzione Hash

collisione

$$h: \Sigma^* \rightarrow \Sigma^n$$

$$h(m_1) = h(m_2)$$



Esistono infinite collisioni

Funzione Hash
proprietà

- **One-way** (***pre-image resistant***):
  - dato un *hash* **y**, è computazionalmente difficile trovare **M** | y=h(M)
- **Sicurezza debole** (***2nd pre-image resistance***):
  - dato **M**, è computazionalmente difficile trovare una variazione di M, **M´**|h(M)=h(M´)
- **Sicurezza forte** (***collision resistance***):
  - computazionalmente difficile trovare 2 diversi messaggi con lo stesso valore hash

Funzione Hash
definizioni

- **Una One-Way Hash Function (OWHF):**
  - verifica le proprietà pre-image e 2nd pre-image resistant;
  - viene detta *weak one-way hash function*

- **Una Collision Resistant Hash Function (CRHF):**
  - verifica la proprietà di collision resistance
  - viene detta *strong one-way hash function*

Funzione Hash
costruzione

Messaggi di lunghezza arbitraria sono trattati utilizzando hash con input fisso:

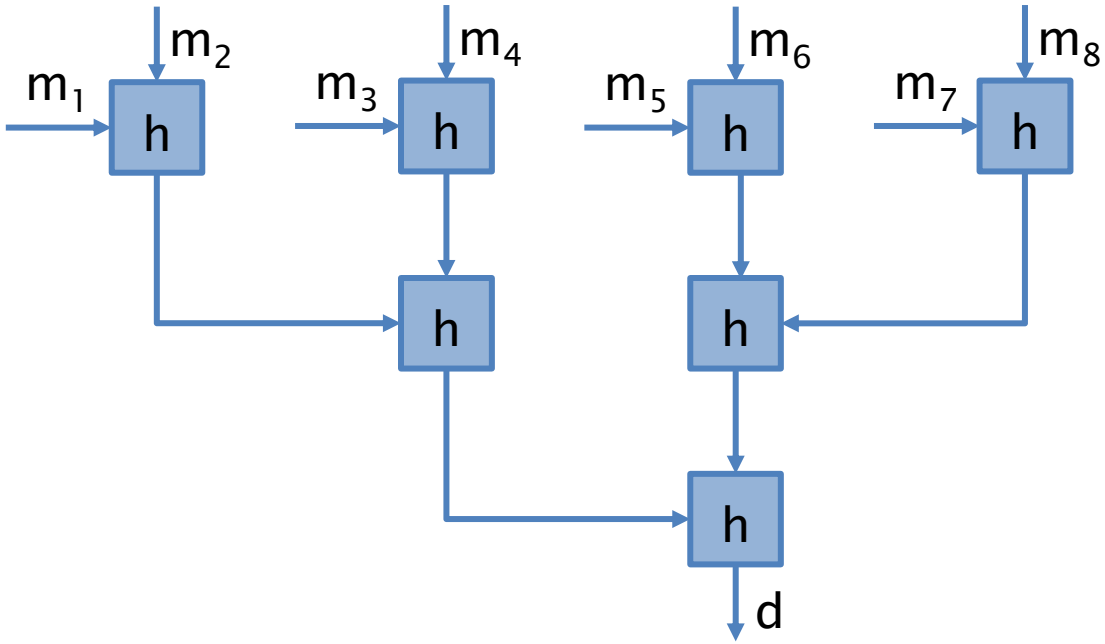
- il messaggio input **M** viene diviso in **k blocchi** di lunghezza fissa: $m_1, m_2, \ldots, m_k$

- i blocchi vengono trattati in modo:
  - seriale/iterato
  - parallelo

Funzione Hash

costruzione

Modello hash parallelo

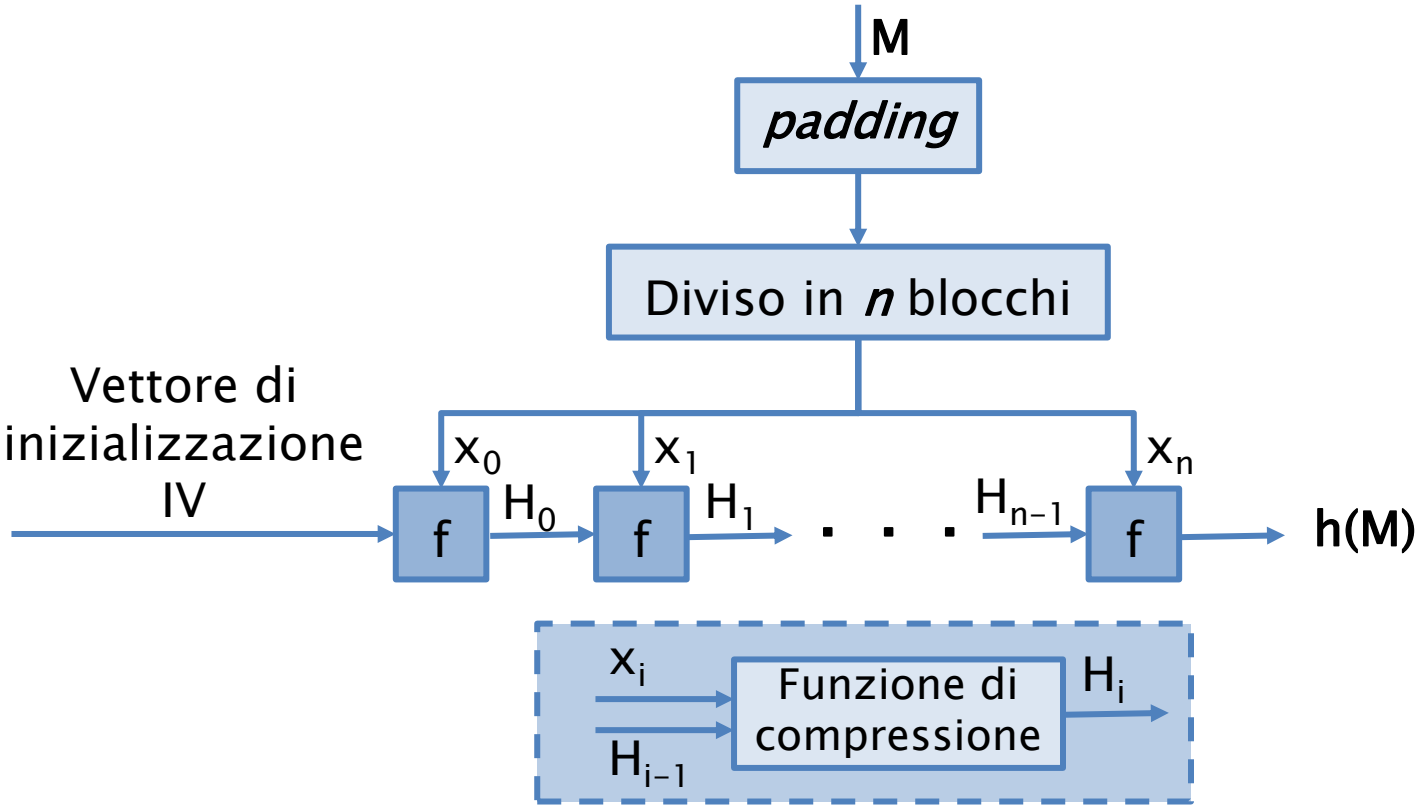


È resistente alle collisioni se lo è la funzione ***h***

Funzione Hash

costruzione

Modello hash iterato

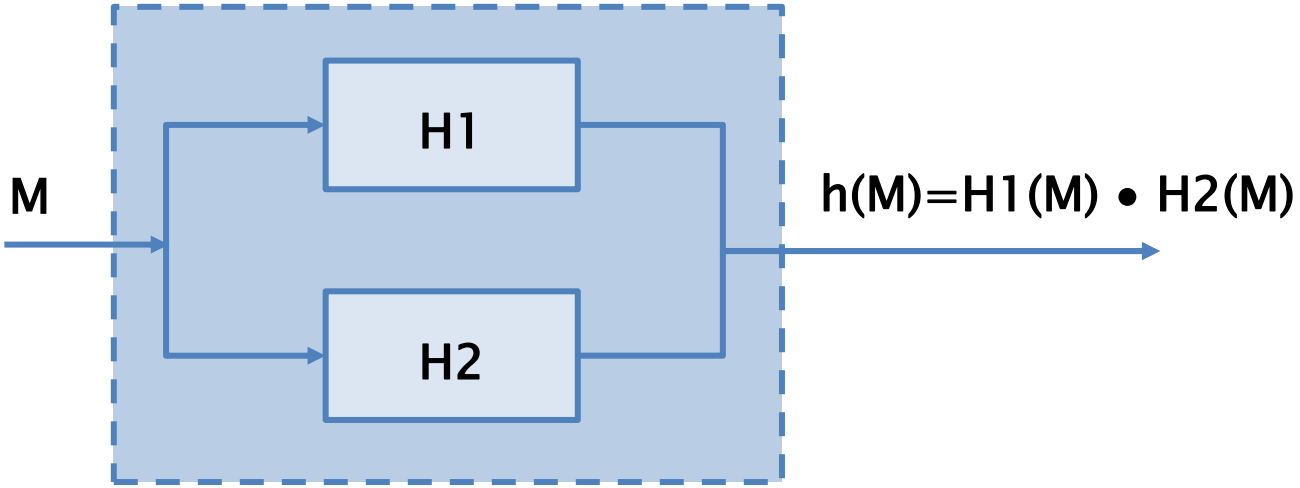


*una collisione per **h**(M) implica una collisione di **f***

Funzione Hash

costruzione

Modello hash cascata



una collisione per h(M) vuol dire trovare
una collisione sia per H1 che per H2

Funzione Hash
costruzione

Cifrari a blocchi

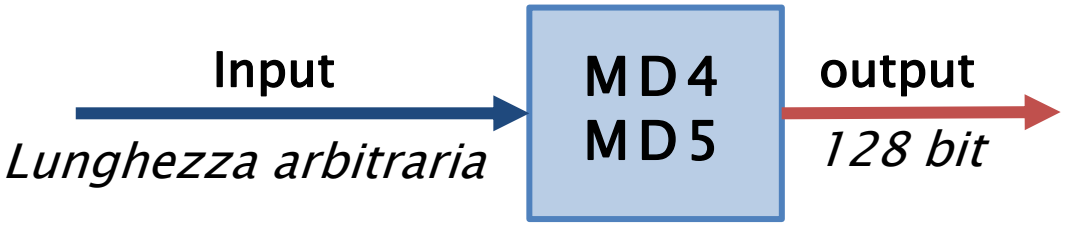
- Cifrario a blocchi E_k (•) per input ad ***n*** bit
- Funzione ***g*** che da ***n*** *bit* produce una chiave
  - **M'$_1$ ... M'$_t$** : è il messaggio **M** con eventuale *padding*
  - **H_0** : è una costante predefinita
  - **H_t** : è il valore hash

  - $H_i = E_{g(H_{i-1})}(M'_i) \oplus M_i'$ [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]

Funzione Hash
MD4/MD5

MD = **M**essage **D**igest

- **MD4**: Progettata nel 1990 da Ron Rivest
  - *RFC1320 -> RFC6150*
- **MD5**: Progettata nel 1991 da Ron Rivest
  - *RFC1321 -> RFC6151*
- Operazioni efficienti su architetture 32 bit little-endian



Little-endian e Big-endian

Rappresentazione di parole da 32 bit

PAROLA W *32 bit*			
B4	B3	B2	B1

- **Little-endian**
  - il byte con indirizzo più basso è quello meno significativo
  - valore parola: $W = 2^{24}B4 + 2^{16}B3 + 2^{8}B2 + 2^{0}B1$
- **Big-endian**
  - il byte con indirizzo più basso è quello più significativo
  - valore parola: $W = 2^{0}B4 + 2^{8}B3 + 2^{16}B2 + 2^{24}B1$

Funzione Hash
MD4/MD5: Obiettivi

- **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

Funzione Hash

MD4/MD5: padding del messaggio

- **MD4/MD5** processa il messaggio in <u>blocchi di 512 bit</u>
  - Ogni blocco consta di 16 parole di 32 bit
- **M'** sarà costituito da:
  - messaggio originario **M**
  - ***p** bit* di padding
  - ***b** bit* di rappresentazione della lunghezza di M (*max* 2^{64})

$$M' = M\ \underbrace{100\ldots0}_{p}\ \underbrace{b}_{64bit}$$

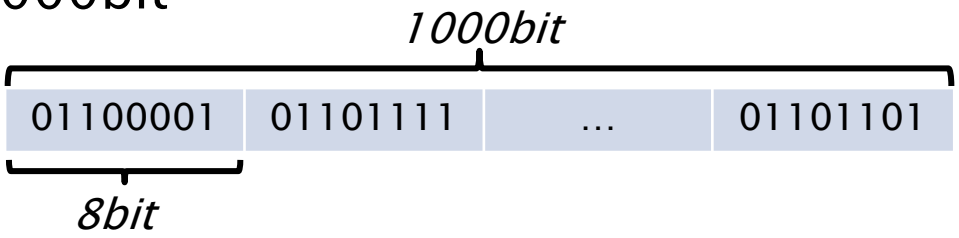
$$p \mid (p+M) \bmod_{512} = 448 \Longleftrightarrow 512-[(M+b) \bmod_{512}]$$

- **M'** consta di un numero di bit multiplo di 512, ovvero di un numero *L blocchi* di 512 bit
  - Ovvero di N parole con N multiplo di 16:
    - L=N/16 blocchi di 512 bit

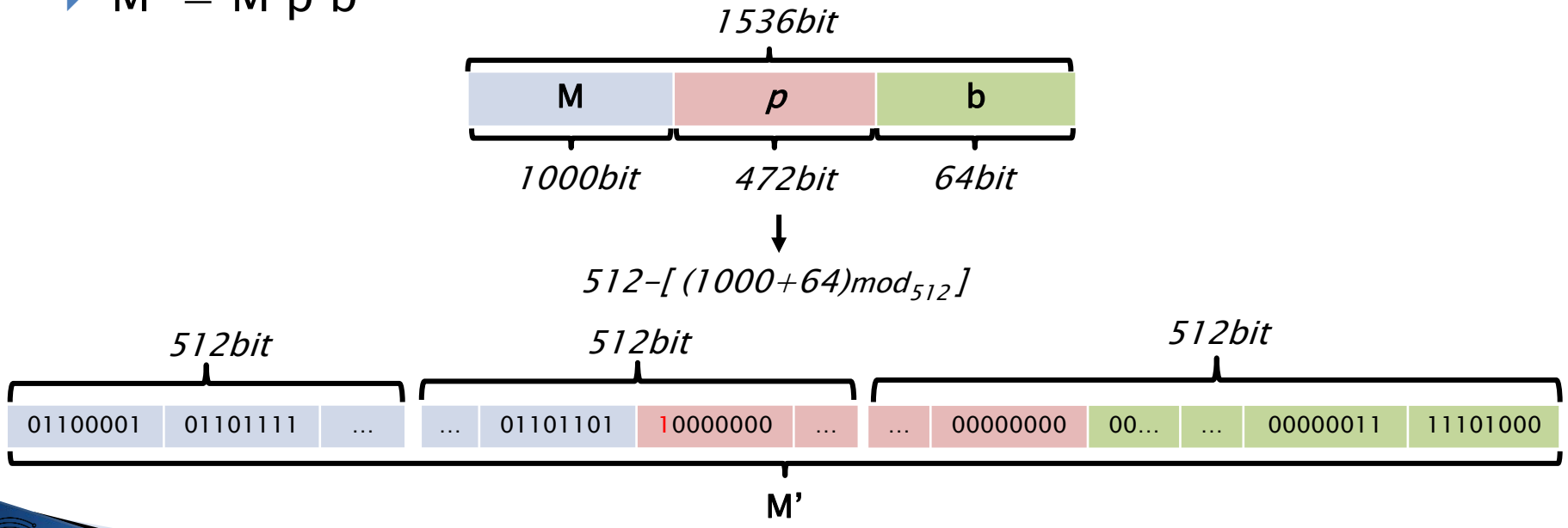
Funzione Hash

MD4/MD5: padding del messaggio

- |M|=1000bit



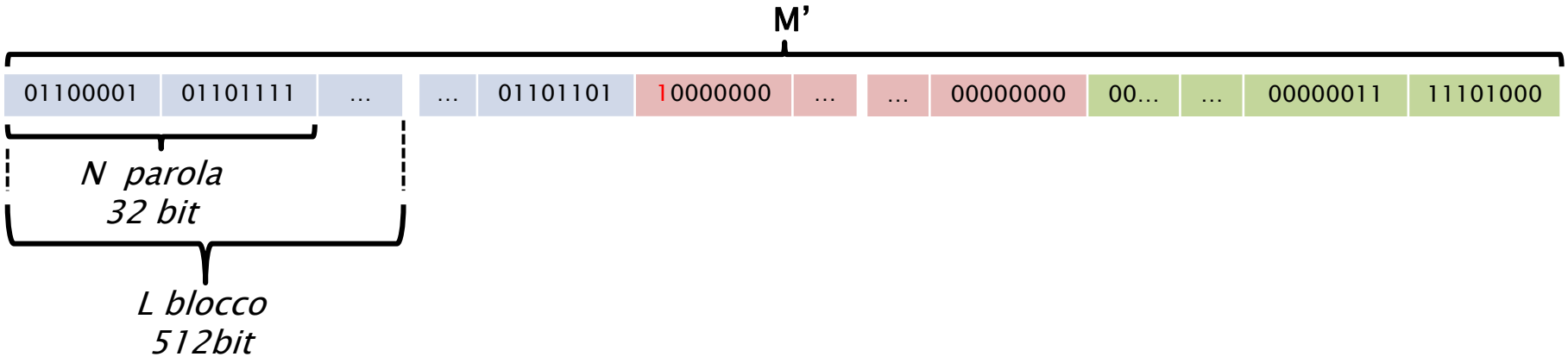
- M' = M p b



Funzione Hash

MD4/MD5: padding del messaggio

- M'=**3 L** *blocchi di 512bit*
- M'=**48 N** *parole da 32bit*



Funzione Hash
MD4/MD5: operazioni

- **MD4/5 impiegano diverse operazioni sulle word in input X e Y restituendo una nuova word:**
  - $(X \wedge Y)$: **and** bit a bit di X e Y
  - $(X \vee Y)$: **or** bit a bit di X e Y
  - $(X \oplus Y)$: **xor** bit a bit di X e Y
  - $(\neg X)$: **complemento** bit a bit di X
  - $(X + Y)$: somma intera modulo 2^{32}
  - $(X << s)$:**shift** circolare a sinistra di s bit

Tutte le operazioni sono molto veloci

Funzione Hash

MD4/MD5: funzioni

- **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: $G(X,Y,Z) = (X \wedge Z) \vee ((Y \wedge (\neg Z))$ **[MD5]** [if Z then X else Y]
  - round 2: $G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge Z) \vee (X \wedge Y)$ **[MD4]** [2 su 3]
  - round 3: $H(X,Y,Z) = X \oplus Y \oplus Z$ [bit di parità]
  - round 4: $I(X,Y,Z) = Y \oplus (X \vee (\neg Z))$ **[MD5]**

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

Funzione Hash

MD4/MD5: funzione di compressione

TRE round [MD4] – QUATTRO round [MD5]

- Ogni round prende in input:
  - blocco corrente di 512 bit = 16 word
  - valore corrente del buffer, 4 word ABCD per 128 bit
- Ogni round consiste di 16 operazioni:
  - [ABCD.k.s] [**MD4**]
  - [ABCD.k.s.i] [**MD5**]
- L'output dell'ultima fase viene sommato all'input della prima fase:
  - la somma avviene word a word
- L'output della L-esima fase è il digest a 128 bit

Funzione Hash

MD4: funzione di compressione

Ogni round consiste di 16 operazioni [ABCD.k.s]

Ogni operazione agisce sul buffer di 4 word A.B.C.D.

$$t = (A + W(B, C, D) + X[j] + y[j])) << s[j]$$

$$(A, B, C, D) = (D, T, B, C)$$

- **X[j]** è predefinito ed è reperibile all'interno dell'algoritmo
- **s[j]** indica lo shift ciclico
- **y[j]** è la costante additiva relativa al round corrente:
  - ***Round 2:*** 5A827999 [$\sqrt{2}$]
  - ***Round 3:*** 6ED9EBA1 [$\sqrt{3}$]
- **W** è la funzione del round (F,G,H)

Funzione Hash

MD5: funzione di compressione

Ogni round consiste di 16 operazioni [ABCD.k.s.i]

Ogni operazione agisce sul buffer di 4 word A.B.C.D.

$$A \leftarrow B + ((A + W(B,C,D) + X[k] + T[i])) \ll s$$

- **K** è l'indice della parola
- **s** indica lo shift ciclico
- **i** è l'indice dell'iterazione
- **W** è la funzione del round (F,G,H,I)
- **X[k] ← M´[16i+k]** è la k-*esima* word di 32 bit nell'i-*esimo* blocco
- **T[i]** è l'i-*esimo* elemento della tabella di 64 valori

Funzione Hash

MD5: funzione di compressione

Costanti T[1…64]

T[i] ← primi 32 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	ffeff47d
8	fd469501	24	e7d3fbc8	40	bebfbc70	56	85845dd1
9	698098d8	25	21e1cde6	41	289b7ec6	57	6fa87e4f
10	8b44f7af	26	c33707d6	42	eaa127fa	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	fcefa3f8	46	e6db99e5	62	bd3af235
15	a679438e	31	676f02d9	47	1fa27cf8	63	2ad7d2bb
16	49b40821	32	8d2a4c8a	48	c4ac5665	64	eb86d391

Funzione Hash

MD4: Esecuzione dell'algoritmo

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;

N = numero di «word»(32bit) del messaggio «M»

Round 1

//[ABCD K S] INDICA A = (A + F(B,C,D) + X[K]) << S

[ABCD. 0. 3] [DABC. 1. 7] [CDAB. 2.11] [BCDA . 3.19]
[ABCD. 4 . 3] [DABC. 5. 7] [CDAB. 6.11] [BCDA . 7.19]
[ABCD. 8 . 3] [DABC. 9. 7] [CDAB.10.11] [BCDA.11.19]
[ABCD.12. 3] [DABC.13. 7] [CDAB.14.11] [BCDA.15.19]

Round 2

// [ABCD K S] INDICA A = (A + G(B,C,D) + X[K]) + 5A827999) << S

[ABCD. 0. 3] [DABC. 4. 5] [CDAB. 8. 9] [BCDA.12.13]
[ABCD. 1. 3] [DABC. 5. 5] [CDAB. 9. 9] [BCDA.13.13]
[ABCD. 2. 3] [DABC. 6. 5] [CDAB.10. 9] [BCDA.14.13]
[ABCD. 3. 3] [DABC. 7. 5] [CDAB.11. 9] [BCDA.15.13]

Funzione Hash

MD4: Esecuzione dell'algoritmo

Round 3

//[ABCD K S] INDICA A = (A + H(B,C,D) + X[K] + 6ED9EBA1) << S

[ABCD. 0. 3] [DABC. 8. 9] [CDAB. 4.11] [BCDA.12.15]
[ABCD. 2. 3] [DABC.10. 9] [CDAB. 6.11] [BCDA.14.15]
[ABCD. 1. 3] [DABC. 9. 9] [CDAB. 5.11] [BCDA.13.15]
[ABCD. 3. 3] [DABC.11. 9] [CDAB. 7.11] [BCDA.15.15]

A←A+AA; B ← B+BB; C ← C+CC; D ← D+DD;

output: (A, B, C, D)

Funzione Hash

MD5: Esecuzione dell'algoritmo

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;

N = numero di «word»(32bit) del messaggio «M»

Round 1

// [ABCD k s i] INDICA A = B + ((A + F(B, C, D) + X[k] + T[i]) << s)

[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]
[ABCD.12. 7.13] [DABC.13.12.14] [CDAB.14.17.15] [BCDA.15.22.16]

Round 2

// [ABCD k s i] INDICA A = B + ((A + G(B, C, D) + X[k] + T[i]) << s)

[ABCD. 1. 5.17] [DABC. 6. 9.18] [CDAB.11.14.19] [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]

Funzione Hash

MD5: Esecuzione dell'algoritmo

Round 3

// [ABCD k s i] INDICA A = B + ((A + H(B, C, D) + X[k] + T[i]) << s)

[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 k s i] INDICA A = B + ((A + I(B, C, D) + X[k] + T[i]) << s)

[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)

Funzione Hash

MD5/MD4: differenze

MD5

- 4 round = 4 · 16 operazioni
- 4 funzioni logiche
- 64 costanti additive
- ogni passo aggiunge il risultato del passo precedente

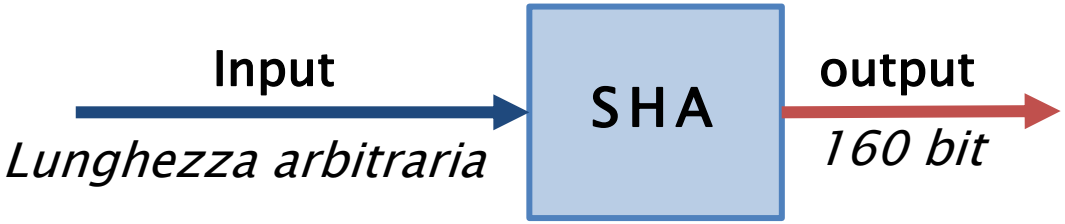
MD4

- 3 round = 3 · 16 operazioni
- 3 funzioni logiche
- 2 costanti additive

Funzione Hash
SHS/SHA

SHS = **S**ecure **H**ash **S**tandard
SHA = **S**ecure **H**ash **A**lgorithm

- Standard del Governo americano dal 1993
- Modificato nel luglio 1994, denotato SHA-1 *(RFC3174/RFC6194)*
  - *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



Funzione Hash
SHA: padding del messaggio

- **SHA** processa il messaggio in <u>blocchi di 512 bit</u>
  - Ogni blocco consta di 16 parole di 32 bit
- **M'** sarà costituito da:
  - messaggio originario **M**
  - ***p** bit* di padding
  - ***b** bit* di rappresentazione della lunghezza di M (*max* 2^{64})

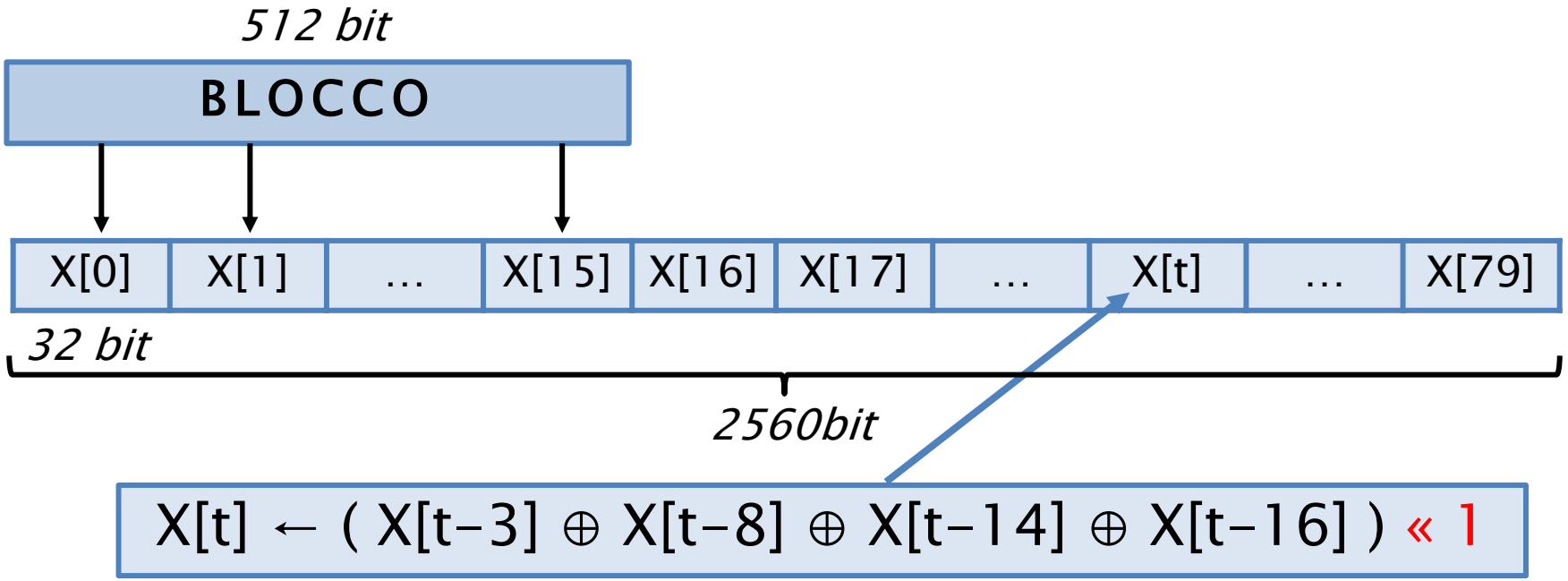
$$M' = M\ \underbrace{100\ldots0}_{p}\ \underbrace{b}_{64bit}$$

$$p \mid (p+M) \bmod_{512} = 448 \Longleftrightarrow 512-[(M+b) \bmod_{512}]$$

- **M'** consta di un numero di bit multiplo di 512, ovvero di un numero *L blocchi* di 512 bit
  - Ovvero di N parole con N multiplo di 16:
    - L=N/16 blocchi di 512 bit

Funzione Hash

SHA: Espansione blocco ed iterazioni



- 4 round di 20 operazioni ciascuna (80 iterazioni)
- Per ogni iterazione una parola X[i] viene calcolata dal blocco input

Funzione Hash

SHA: funzioni

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

- round 1 t= 0,...,19: $F(t,X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$ [if X then Y else Z]
- round 2 t=20,...,39: $F(t,X,Y,Z) = X \oplus Y \oplus Z$ [bit di parità]
- round 3 t=40,...,59: $F(t,X,Y,Z) = (X \wedge Z) \vee (Y \wedge Z) \vee (X \wedge Y)$ [2 su 3]
- round 4 t=60,...,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	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

Funzione Hash

SHA: costanti additive

Costante additiva K[t]:

- round 1 t = 0,...,19: 5a827999
- round 2 t = 20,...,39: 6ed9eba1
- round 3 t = 40,...,59: 8f1bbcdc
- round 4 t = 60,...,79: ca62c1d1

Rappresenta la parte intera di $2^{30} \bullet \sqrt{P_i}$

- $P_1=2$; $P_2=3$; $P_3=5$; $P_4=10$;

Funzione Hash

Algoritmo 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´[16i+j]
  for t = 16 **to** 79 **do**
    X[t] ← (X[t-3] ⊕ X[t-8] ⊕ X[t-14] ⊕ X[t-16]) « 1 ← *Aggiunto con SHA-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)

Funzione Hash
SHA-1 vs MD4/MD5

- **Sicurezza forte**
  - maggiore in SHA-1, output 32 bit più lungo di MD4/5 (160 contro 128)
  - Attacco a compleanno 280 contro 264
- **Sicurezza contro l'analisi**
  - MD5 è soggetta ad alcuni attacchi
- **Velocità**
  - entrambi algoritmi molto veloci; SHA-1 ha più passi (80 contro 64) e il buffer ha 160 bit rispetto ai 128 bit di MD5
- **Semplicità e Compattezza**
  - semplice da descrivere e da implementare, nessun uso di tabelle e di complesse strutture dati

Funzione Hash
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 *message digest* : 256, 512, 384 bit
  - Sicurezza attacco del compleanno 128, 256, 192 bit
- **Draft di Federal Information Processing Standard (FIPS), gennaio 2001**

Funzione Hash
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

Funzione Hash
altre funzioni di 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
- **RIPEMD-160**
  - Bosselaers, Dobbertin, Praeneel [1996]

SSRI Lorenzo Laurato s.r.l.

Via Coroglio nr. 57/D (BIC- Città della Scienza)
80124 Napoli

Tel. 081.19804755
Fax 081.19576037

lorenzo.laurato@unina.it
lorenzo.laurato@ssrilab.com

www.docenti.unina.it/lorenzo.laurato
www.computerforensicsunina.forumcommunity.net