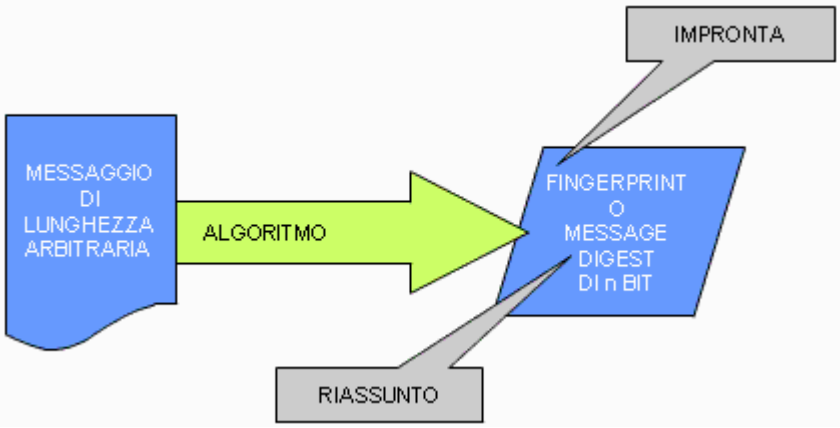
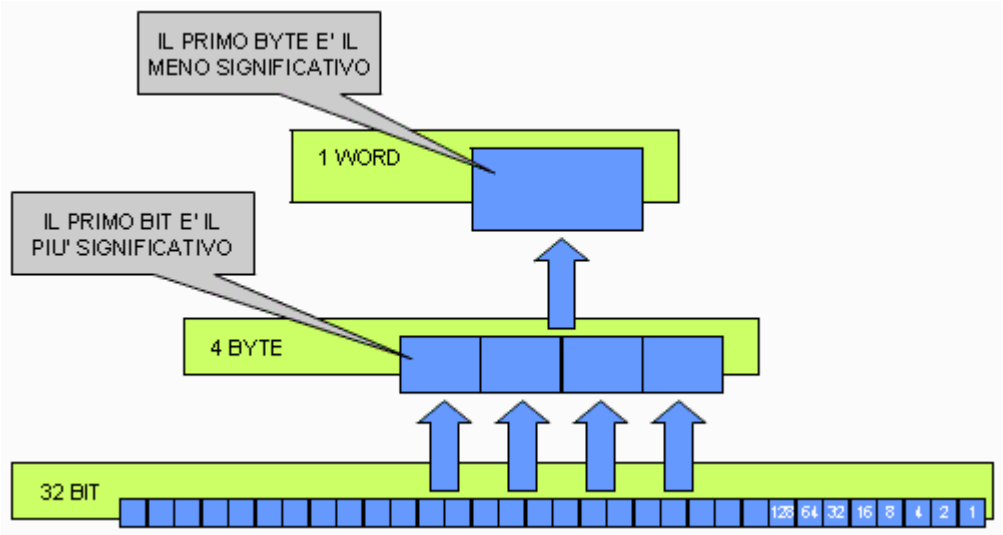
L'algoritmo MD5 genera un'impronta, chiamata anche fingerprint o message digest, della lunghezza di 128 bits, di un messaggio di lunghezza arbitraria.



Terminologia e convenzioni

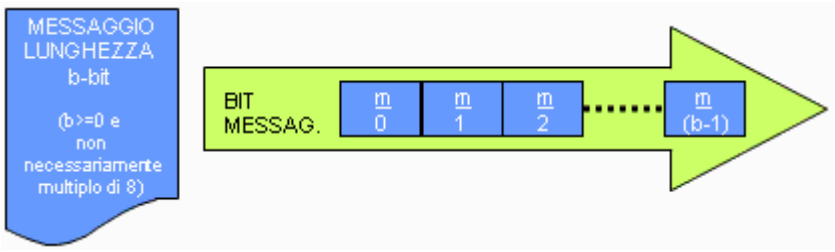
In questo documento sono state utilizzate queste convenzioni:



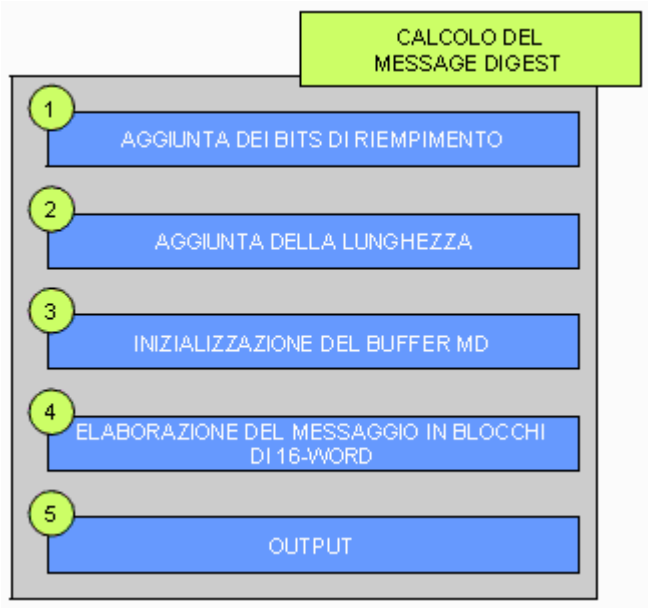
X <<< s	il valore a 32-bit ottenuto ruotando a sinistra X di s bits
not(X)	complemento binario di X
X v Y	OR binario tra X e Y
X xor Y	XOR binario tra X e Y
XY	AND binario tra X ed Y

Analisi dell'algoritmo

L'algoritmo, appartenente alla RSA Data Security, Inc. è stato sviluppato da R. Rivest nel 1991.



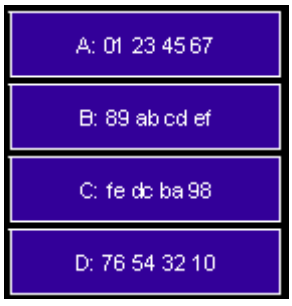
L'algoritmo è suddiviso in cinque fasi principali:



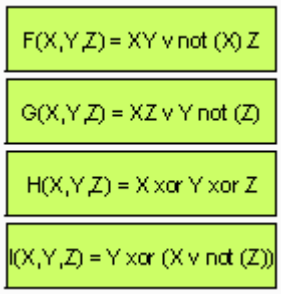
Aggiunta bits di riempimento: il messaggio viene sempre esteso (padding) così che la sua lunghezza in bits sia congruente a 448 mod 512. Il primo bit di estensione è sempre un '1' seguito da una serie di '0' mentre il numero di bits di estensione va da un minimo di 1 ad un massimo di 512.

Aggiunta della lunghezza: viene aggiunta una rappresentazione a 64-bit della lunghezza del messaggio (b) prima del riempimento. Se la lunghezza era maggiore a 2^64 vengono utilizzati solo i 64 bits inferiori di b e le due word a 32-bit risultanti vengono accodate, seguendo la rappresentazione vista in precedenza, con la word più bassa per prima. Il messaggio ottenuto ha una lunghezza multipla di 512 bits (in pratica 16 words da 32-bit).

Inizializzazione del buffer MD (initial variable/chaining variable): si tratta di un buffer di quattro word (A, B, C, D) a 32-bit aventi questi valori esadecimali di inizializzazione (la prima word per prima):



Elaborazione del messaggio (compression function): vengono definite quattro funzioni ausiliare che ricevono in ingresso tre words da 32-bit e producono in uscita una sola word a 32-bit:



Per ogni bit le funzioni applicano la condizione che, se è vero X passano ad Y, altrimenti a Z, senza introdurre una propria logica ma, seguendo quella dei singoli bits.

La funzione H produce, mediante un operazione di XOR binario, la 'parità' dei bit ingresso. In questo passaggio viene anche utilizza una tabella T di 64 elementi T[1… 64] così da avere per ogni T[i] un valore equivalente alla parte intera di 4294967296 volte abs(sin(i)), con i espresso in radianti.

Ogni blocco da 16-word viene elaborato seguendo questo algoritmo:

```
/* Elabora ogni blocco da 16 word. */
For i = 0 to N/16-1 do

/* Copia il blocco i dentro X. */
For j = 0 to 15 do
     Set X[j] to M[i*16+j]
end /* del ciclo j */

/* Salva A come AA, B come BB, C come CC, e D come DD. */
AA = A
BB = B
CC = C
DD = D

/* Passaggio 1. */
/* [abcd k s i] indica l'operazione:
a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Effettua le seguenti 16 operazioni. */

[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]

/* Passaggio 2. */
/* [abcd k s i] indica l'operazione:
a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Effettua le seguenti 16 operazioni. */

[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0  20 20]
[ABCD  5  5 21]  [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 31]  [BCDA 12  20 32]

/* Passaggio 3. */
/* [abcd k s t] indica l'operazione:
a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* Effettua le seguenti 16 operazioni. */

[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 47]  [BCDA  2 23 48]

/* Passaggio 4. */
/* [abcd k s t] indica l'operazione:
a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* Effettua le seguenti 16 operazioni. */

[ABCD  0  6  49]  [DABC  7 10  50]  [CDAB 14 15 51]  [BCDA  5  21 52]
[ABCD 12  6  53]  [DABC  3 10  54]  [CDAB 10 15 55]  [BCDA  1  21 56]
[ABCD  8  6  57]  [DABC 15 10  58]  [CDAB  6 15 59]  [BCDA 13  21 60]
[ABCD  4  6  61]  [DABC 11 10  62]  [CDAB  2 15 63]  [BCDA  9  21 64]

/* Quindi esegue le seguenti addizioni. (Ad ognuno dei quattro registri viene aggiunto il
valore che aveva prima dell'inizio del processo) */

A = A + AA
B = B + BB
C = C + CC
D = D + DD

end /* fine del ciclo  i */
```

Output: Il message digest è ottenuto partendo dal byte meno significativo di A seguito da quelli di B, C, e terminato con il byte più significativo di D.