Requisiti di una funzione di hash

•È facile calcolare l'hash di un messaggio

•È difficile trovare il messaggio che ha generato un dato hash (non invertibilità delle funzioni di hash)

•È difficile modificare un messaggio senza modificare il relativo hash(resistenza debole alle collisioni)

•È difficile trovare due messaggi che abbiano lo stesso hash (resistenza forte alle collisioni)

Gli hash sono una sorta di "impronta digitale" a lunghezza fissa di un messaggio. Sono quindi utili in diversi settori applicativi come l'accesso diretto negli archivi, per identificare eventuali errori di trasmissione dei dati, per garantire che un messaggio non sia stato modificato da un eventuale attacco.
Nella maggior parte dei casi, quando un sistema informatico deve memorizzare una password, questa non viene salvata in chiaro per motivi di sicurezza, viene memorizzato l'hash della password per evitare che sia possibile risalire alla password originale.
MD5 (message-digestalgorithm5) è un algoritmo di hash sviluppato da Ronald Rivest(la "R" di RSA). Genera hash da 128 bit (32 caratteri esadecimali). È stato ampiamente usato finora, ma negli ultimi anni sono state dimostrate alcune sue debolezze crittografiche, per cui non è più considerato sicuro per applicazioni critiche.
Tecnicamente, MD5 è una funzione di compressione, che dato un input composto da una stringa di lunghezza arbitraria, restituisce una firma digitale che consiste in una stringa da 128 bit. Si presuppone che l'hash (ovvero l'output) restituito dalla funzione sia univoco o, più precisamente, che sia molto improbabile ottenere due hash identici da due input diversi.
Al giorno d'oggi, tutti sanno quanto sono importanti le firme digitali, la loro sicurezza dipende dalla "forza crittografica" delle funzioni che le generano. Le funzioni di hashing, hanno anche altre funzioni oltre alla sicurezza, infatti permettono di verificare l'integrità dei downloads, tramite un checksum da confrontare: Assieme al file del download, viene fornita la stringa che corrisponde all'hash dei file immessi come input. Se l'hash ottenuto non corrisponde a quello fornito dal download, potrebbe essersi verificata una perdita di dati.

MD5 è l'acronimo di Message Digest 5, un algoritmo di crittografia a senso unico (cioè che non prevede di essere decrittato), creato nel 1991 da Ronald Rivest a causa dell'inefficienza del suo predecessore: MD4.

Tecnicamente, MD5 è una funzione di compressione, che dato un input composto da una stringa di lunghezza arbitraria, restituisce una firma digitale che consiste in una stringa da 128 bit. Si presuppone che l'hash (ovvero l'output) restituito dalla funzione sia univoco o, più precisamente, che sia molto improbabile ottenere due hash identici da due input diversi.

Al giorno d'oggi, tutti sanno quanto sono importanti le firme digitali, la loro sicurezza dipende dalla "forza crittografica" delle funzioni che le generano. Le funzioni di hashing, hanno anche altre funzioni oltre alla sicurezza, infatti permettono di verificare l'integrità dei downloads, tramite un checksum da confrontare: Assieme al file del download, viene fornita la stringa che corrisponde all'hash dei file immessi come input. Se l'hash ottenuto non corrisponde a quello fornito dal download, potrebbe essersi verificata una perdita di dati.
In fase di registrazione la password utente subisce il processo di hashing e l'impronta viene conservato nella base dati. In fase di autenticazione la password inserita subisce la medesima funzione di hashing e l'impronta generata è confrontato con quanto memorizzato in base dati. La sicurezza dell'algoritmo è

garantita dal fatto che il sistema non conosce la password utente ma solamente la sua impronta che, per sua natura, non è invertibile.

Procedimento alla formazione dell'hash:

Padding: Vengono aggiunti bit alla fine del messaggio da codificare finchè la lunghezza del messaggio diventa pari a 448 mod 512. In particolare, il primo bit aggiunto è un "1", mentre i successivi sono tutti "0".
Aggiunta delle informazioni sulla lunghezza del messaggio (calcolata prima del padding) in codifica a 64 bit. Se la lunghezza del messaggio era minore di 264, vengono utilizzati solamente i 64 bit inferiori del messaggio, ottenendo così 2 word (da 32 bit ciascuna) che vengono accodate al messaggio dando precedenza alla word inferiore.
A questo punto il messaggio ha ottenuto una lunghezza multipla di 512.
Inizializzazione del buffer MD: questo buffer è composto da 4 word a 32 bit, inizializzate come segue:

A: 01 23 45 67

B: 89 ab cd ef

C: fe dc ba 98

D: 76 65 32 10

Elaborazione del messaggio:Vengono definite quattro funzioni che ricevono in ingresso tre word e ne restituiscono una:

F(x,y,z) = (x AND y) OR(NOTx OR z)

G(x,y,z) = (x AND z) OR(y OR NOTz)

H(x,y,z) = (x XOR y XOR z)

I(x,y,z) = y XOR(x OR NOTz)

In particolare ogni funzione adotta la seguente logica: se è vero x, allora passa ad y altrimenti a z. Ecco lo pseudocode dell'algoritmo:

//Nota: Tutte le variabili sono "unsigned" da 32 bit

var int[64] r, k//r specifica lo spostamento per iterazione

r[0..15] := {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22}

r[16..31] := {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20}

r[32..47] := {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23}

r[48..63] := {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}

//Usa la parte binaria intera del seno di interi come costante:

for i from 0 to 63

k[i] := floor(abs(sin(i + 1)) × (2 pow 32))

```
//Inizializzazione
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476
//Pre-processazione
append "1" bit to message
append "0" bits until message length in bits ≡ 448 (mod 512)
append bit (bit, not byte) length of unpadded message as 64-bit little-endian integer to message
//Processa il messaggio in pezzi da 512
for each 512-bit chunk of message
break chunk into sixteen 32-bit little-endian words w[i], 0 ≤ i ≤ 15
//inizializza il valore dell'hash di questo pezzo
var int a := h0
var int b := h1
var int c := h2
var int d := h3
//Main loop:
for i from 0 to 63
if 0 ≤ i ≤ 15 then
f := (b and c) or ((not b) and d)
g := i
else if 16 ≤ i ≤ 31
f := (d and b) or ((not d) and c)
g := (5×i + 1) mod 16
else if 32 ≤ i ≤ 47
f := b xor c xor d
g := (3×i + 5) mod 16
```

```
else if 48 ≤ i ≤ 63

f := c xor (b or (not d))

g := (7×i) mod 16

temp := d

d := c

c := b

b := b + leftrotate((a + f + k[i] + w[g]) , r[i])

a := temp

//Aggiunge questa parte di hash al risultato

h0 := h0 + a

h1 := h1 + b

h2 := h2 + c

h3 := h3 + d

var int digest := h0 append h1 append h2 append h3
//(expressed as little-endian)
//Definizione della funzione "leftrotate"
leftrotate (x, c)
return (x <> (32-c));
```

Algoritmo hash quickhash64

```
#include <iostream>
using namespace std;
unsigned long long quickhash64(const char *str, unsigned long long mix = 0);
int main()
{
   cout << quickhash64("CiaoMondo") << endl;
   return 0;
}
 unsigned long long quickhash64(const char *str, unsigned long long mix )
{
   int i=0;    const unsigned long long mulp = 2654435789;     mix ^= 104395301;
   /* cicla LunghezzaStringa volte*/

   while(*str)   mix += (*str++ * mulp)^(mix >> 23);} // >> operatore bit a bit shift a destra
   return mix^(mix << 37);
}
```