

# Funzioni hash con PHP



**Daniele Frongia e Raffaello Martinelli**

Ottobre 2008

Nell'immagine di copertina le saline di Trapani (il *salt* è un importante fattore per le funzioni hash). Foto per gentile concessione di *Agenzia Publy Web Promotion*.

Copyright (c) 2008 Daniele Frongia and Raffaello Martinelli. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

## **Sommario**

Un funzione hash è una funzione non invertibile che trasforma un testo di lunghezza arbitraria in una stringa di lunghezza fissa. Algoritmi come MD5, RIPEMD-160, SHA-512, GOST vengono utilizzati per realizzare firme digitali, rendere sicure transazioni bancarie, proteggere database, eseguire investigazioni giudiziarie, controllare la qualità della trasmissione di video e per numerosi altri scopi. Il primo obiettivo di questo lavoro è stato effettuare una panoramica delle funzioni hash, dalle proprietà matematiche agli scenari applicativi, con particolare attenzione alla resistenza alle collisioni. Il secondo obiettivo è stato mettere in pratica le funzioni con un linguaggio di programmazione, PHP, confrontare gli algoritmi in termini di performance e suggerire alcune indicazioni per aumentare il livello di sicurezza dei sistemi web based.

**Parole chiave:** hash, sicurezza informatica, steganografia, firma digitale, funzioni one-way, password cracking, tabelle rainbow, PHP

## Indice

Autori .....	5
Ringraziamenti .....	5
Introduzione .....	6
CAPITOLO UNO Le funzioni hash .....	8
1.1 Funzioni di hashing: definizione.....	9
1.2 Caratteristiche di una funzione hash .....	10
1.3 Sul funzionamento degli algoritmi di hash.....	12
1.4 Hashing crittografico.....	12
1.5 Hashing non crittografico.....	19
1.6 Possibili attacchi ai sistemi protetti da funzioni hash.....	24
1.7 Miglioramento della sicurezza (contromisure per le collisioni).....	26
1.8 Panoramica sugli algoritmi di hash.....	27
CAPITOLO DUE Hashing con PHP.....	31
2.1 PHP e funzioni hash .....	33
2.2 Algoritmi di hash con PHP5 .....	33
2.3 L'hash di stringhe.....	34
2.4 L'hash di un file .....	35
2.5 L'hash di dati eterogenei .....	37
2.6 Confronto tra algoritmi utilizzando PHP.....	38
Approfondimenti online.....	42
Testo della GNU Free Documentation License.....	44

## **Autori**

Daniele Frongia e Raffaello Martinelli, entrambi tecnologi dell'Istituto Nazionale di Statistica ([Istat](#)). Sebbene il lavoro sia frutto di un impegno comune, Daniele Frongia è autore dei paragrafi 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.8, mentre Raffaello Martinelli è autore dei paragrafi 1.7, 2.1, 2.2, 2.3, 2.4, 2.5. Il paragrafo 2.6 sul confronto tra algoritmi è stato curato da entrambi gli autori. Daniele Frongia ha inoltre modificato ed integrato alcuni articoli di Wikipedia riportandone alcune parti nel presente lavoro.

## **Ringraziamenti**

Alla prima versione di questo testo hanno contribuito Andrea Stanco (un matematico a portata di mano serve sempre...), Roberto Gismondi per l'analisi dei risultati del confronto tra algoritmi, Paolo Franciosa per una *decifrazione linguistica*, Umberto Cerruti per l'esempio di steganografia, Eugenio Rustico per il suo divertente *hashemall*, Alessio Damato per gli spunti sul *digital video fingerprinting* e l'instancabile Andrea Libratore per il supporto IT. A loro il nostro ringraziamento e l'invito a contribuire al testo una volta pubblicato su WikiBooks.

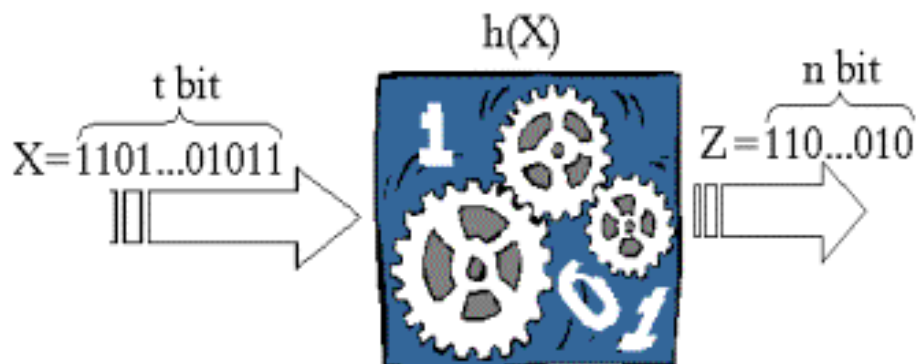
## Introduzione

Il verbo inglese *to hash* significa sminuzzare, ma anche pasticciare. Infatti per [hash](#) si intende una polpetta con avanzi di carne e verdure, insomma un'ottima pappa per cani.



Hash di manzo, patate e carote ([Wikipedia](#) )

Tuttavia in questo documento non ci occuperemo di cibo per animali. Un hash, da un punto di vista matematico, è una funzione non invertibile che trasforma un testo di lunghezza arbitraria in una stringa di lunghezza fissa. Vedremo quindi come le funzioni hash stanno assumendo una notevole importanza sia per questioni di sicurezza informatica che per altri scopi. Possiamo applicare una funzione hash ad un qualsiasi oggetto digitale: ad un testo ma anche ad un qualsiasi file, come ad esempio uno script di un sistema web oppure una foto.



Funzione hash come macchina che comprime sequenze di bit (immagine di [Alfredo De Santis](#))

Nel primo capitolo di occuperemo delle funzioni hash nei loro aspetti teorici e applicativi. Nella seconda invece prenderemo in esame [PHP](#) come linguaggio

di programmazione per implementare tali funzioni. La scelta è ricaduta su PHP per diversi motivi: è uno dei cardini tecnologici del Web 2.0 (Wikipedia, YouTube, Facebook, etc), è open source e freeware e le sue ultime release consentono l'implementazione di numerosi algoritmi di hash.

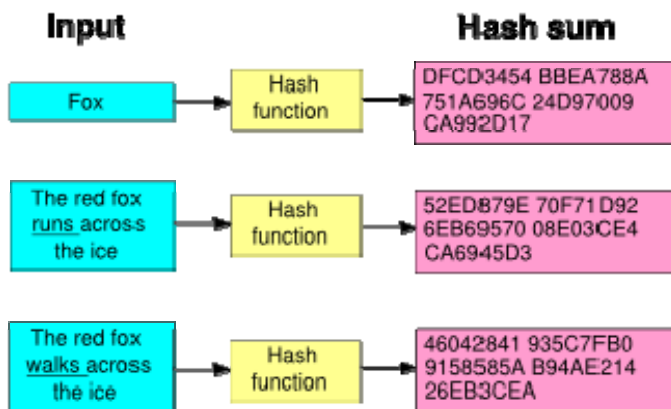
Questo documento va considerato come un contributo *open* che necessita di ulteriori approfondimenti e del coinvolgimento di altri esperti: per questo motivo verrà pubblicato (dopo un opportuno adeguamento dei formati e dei contenuti) su [WikiBooks](#) e sarà quindi possibile modificarlo e migliorarlo.

CAPITOLO UNO

# Le funzioni hash

## 1.1 Funzioni hash: definizione

In questi primi tre paragrafi vedremo la definizione e le proprietà delle funzioni hash. Per hash si intende una funzione non invertibile che trasforma una stringa o un file di lunghezza arbitraria in una stringa di lunghezza fissa. Tale stringa è chiamata in diversi modi: impronta digitale, valore hash o hash, checksum crittografico o message digest. Nella figura è riportato il meccanismo di una funzione hash che prende in input una stringa e restituisce un output di lunghezza fissa (l'output è limitato ai primi 4 byte e la notazione è esadecimale).



Funzionamento di una funzione hash ([Wikipedia](#))

In questo testo indicheremo con  $fh$  la funzione hash che associa a un messaggio  $M$  il corrispondente valore di hash  $h$ . Per semplicità utilizzeremo il simbolo  $M$  (o  $x$ ) per indicare un elemento del dominio di  $fh$ .

### Algoritmi o funzioni?

Spesso in letteratura si parla di algoritmi di hash. Il passaggio dal concetto di funzione a quello di algoritmo è, nel nostro caso, piuttosto immediato. Esiste infatti una formalizzazione del problema di algoritmo per cui questo viene definito come una sequenza finita di operazioni elementari che preso un valore in input ne genera uno in uscita. Dato quindi un algoritmo  $H$  si denota con  $fh$  la funzione che associa a ogni ingresso  $x$  di  $H$  il corrispondente output  $fh(x)$ .

### Breve digressione sulle proprietà matematiche delle funzioni hash

Qualche considerazione sulle funzioni hash considerate dal punto di vista matematico. A tal proposito vale la pena riprendere la [definizione di funzione](#) e indichiamo rispettivamente con  $X$  e  $Y$  il dominio e il codominio della funzione:

$$f : X \rightarrow Y$$

Il dominio delle funzioni hash  $X$  è l'insieme di tutte le possibili sequenze binarie di lunghezza finita. Il codominio  $Y$ , dato che i valori di hash sono di dimensione limitata e costante, può essere considerato come un [sottoinsieme proprio](#) del dominio.

Le funzioni hash sono [non invertibili](#), vale a dire che non esiste la funzione inversa tale che a partire da un elemento del codominio si possa risalire al corrispondente elemento del dominio; in altre parole, non è possibile risalire dal valore di hash al testo di partenza. Alcune volte nelle definizioni di funzione hash si specifica la proprietà di univocità, proprietà che però è implicita nella definizione stessa di funzione.

Perché le funzioni hash sono non invertibili? Una funzione è invertibile se e solo se è biiettiva, ovvero, equivalentemente, se è iniettiva e suriettiva. Ma una funzione hash non è iniettiva, dal momento che elementi distinti del dominio possono non avere un'immagine distinta. Non vale quindi la condizione:

$$\forall x_1, x_2 \in X, x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$$

Si può giungere a questo risultato anche con un approccio differente: le funzioni hash non sono iniettive perché non esiste una [corrispondenza biunivoca](#) tra dominio e codominio. E in effetti dato che la cardinalità dell'insieme dei possibili input è maggiore di quella dell'insieme dei possibili valori hash, per il [principio dei cassetti](#) ad almeno un hash corrisponderanno più testi possibili. Questo fenomeno, il tallone d'Achille delle funzioni hash, prende il nome di "collisione".

Ovviamente non tutte le funzioni non invertibili possono essere utilizzate per l'hashing: ad esempio  $f(x) = x^2$  è non invertibile (non è iniettiva: e.g.  $f(2) = f(-2) = 4$ ) ma è ugualmente facile risalire alla preimmagine (2 e -2) dell'immagine data (4) tramite la radice quadrata. Quando si parla di hash quindi per "non invertibile" si fa riferimento ad una condizione più restrittiva. Questa condizione richiama il concetto di [funzione one-way](#), funzione che deve essere semplice da calcolare e "difficile" da invertire. Per "difficile" intendiamo che, data una funzione one-way  $f$ , e definito un  $x$  casuale, nessun algoritmo probabilistico è in grado di calcolare in tempo polinomiale una preimmagine di  $f(x)$  con probabilità significativamente superiore a zero. L'[esistenza delle funzioni one-way](#) rientra tra i problemi aperti dell'informatica.

## 1.2 Caratteristiche di una funzione hash

Non esiste un insieme univoco di caratteristiche che contraddistingue le

funzioni hash. Molto dipende dal contesto in cui si utilizzano. Quando l'ambito è crittografico ovviamente le caratteristiche desiderabili sono più restrittive, e diventano prerequisiti. Di seguito le principali caratteristiche:

### **Caratteristiche generali**

- Il testo  $M$  può essere di lunghezza arbitraria, mentre  $fh(M)=h$  di lunghezza costante.
- $h$  deve essere un testo non comprensibile e sempre non simile all'originale.
- $fh(M)$  deve essere facilmente calcolabile.
- $fh$  deve essere coerente (deve cioè rispettare la definizione di funzione matematica): a messaggio uguale deve corrispondere uguale hash.

### **Caratteristiche necessarie in ambito crittografico**

- $fh$  deve essere non invertibile (sia nel senso di non biiettività che di "invertibilità" delle funzioni one-way): non deve essere possibile risalire, in un tempo congruo con la dimensione dell'hash, ad un testo/file che possa generarlo. A tal proposito si vedano anche le [proprietà delle funzioni one way](#).
- Pur non essendo iniettiva,  $fh$  deve essere tale che la probabilità che si verifichino delle collisioni non sia significativamente superiore a zero.
- Ad una piccola modifica del testo originale deve corrispondere un grande cambiamento nel valore di hash (il c.d. "effetto valanga").
- Ognuna delle possibili sequenze binarie che costituiscono l'hash deve avere la stessa probabilità di essere generata delle altre: detta in altri termini, una funzione hash si dovrebbe comportare come una funzione che genera output "casuali" pur rimanendo strettamente deterministica.
- $fh$  deve essere resistente alle collisioni.

### **Proprietà relative alle collisioni**

Abbiamo visto che quando due testi producono lo stesso hash, si parla di collisione, e la qualità di una funzione hash è misurata direttamente in base alla difficoltà nell'individuare due testi che generino una collisione. Per scongiurare l'utilizzo di algoritmi di hashing in passato considerati sicuri è stato infatti sufficiente riuscire a generare una collisione. Questo è quello che è avvenuto ad esempio per gli algoritmi MD2, MD4 e MD5. In alcuni casi, quando l'insieme dei possibili input è conosciuto a priori, è possibile creare una [funzione hash perfetta](#), che riduce a zero la probabilità di collisioni.

Vediamo ora come formalizzare le proprietà di resistenza alle collisioni.

Siano  $M1$  e  $M2$  due generici valori di input e  $fh$  la funzione hash, si definiscono:

### *Resistenza debole alle collisioni*

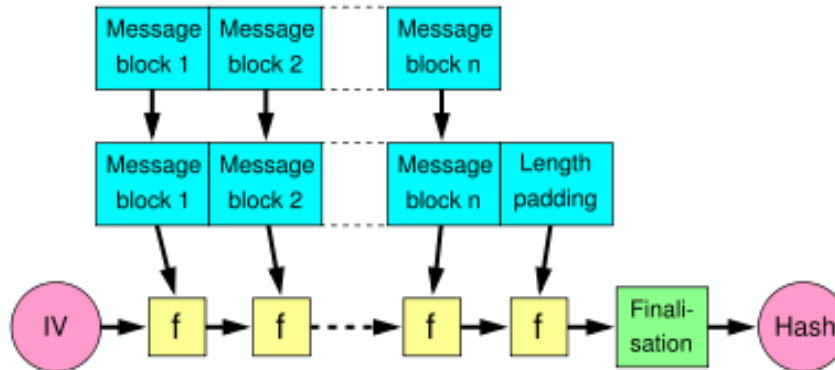
Dato un input  $M1$  è difficile trovare un altro  $M2$  (diverso da  $M1$ ) tale che  $fh(M1) = fh(M2)$ . Per difficile si intende che la probabilità di individuare tale  $M2$  non è significativamente superiore a zero. Utilizzando un'altra notazione possiamo dire che per difficile si intende computazionalmente impossibile.

### *Resistenza forte alle collisioni*

E' computazionalmente impossibile trovare due qualsiasi valori  $M1$  e  $M2$  tali che  $fh(M1) = fh(M2)$ .

## 1.3 Sul funzionamento degli algoritmi di hash

Abbiamo visto che  $fh$  prende in input un qualsiasi oggetto digitale (documento, foglio di lavoro, script, foto...), ovvero una qualsiasi sequenza binaria. Le funzioni hash lavora così: divide l'input in una serie di blocchi di uguale misura e opera su questi usando un [funzione one-way di compressione](#). Le funzioni hash più diffuse utilizzano la [costruzione Merkle-Damgård](#), che prevede, come mostrato in figura, l'aggiunta di un ulteriore blocco ([length padded](#)).



Costruzione Merkle-Damgård ([Wikipedia](#))

## 1.4 Hashing crittografico

Oggi l'hashing è utilizzato nei più diversi campi e la sua importanza è in costante crescita. Se inizialmente l'utilizzo era limitato in ambito crittografico, oggi le applicazioni sono ben più numerose.

Possiamo subito fare una distinzione tra campi d'applicazione crittografici e non. Per i primi intendiamo quelli in cui le funzioni hash sono impiegate per garantire la sicurezza delle informazioni, siano esse messaggi, email, foto, dati di un db, etc. Per campi non crittografici, dei quali ci occuperemo nel

prossimo paragrafo, intendiamo invece quelli in cui le funzioni hash non sono utilizzate per proteggere delle informazioni (ma ad esempio per il recupero efficiente delle informazioni).

Prima di tutto vediamo alcune delle proprietà di cui deve godere un messaggio  $M$  trasmesso nell'ambito di un sistema crittografico:

- *Riservatezza*: il messaggio deve essere letto solo dal mittente e dai legittimi destinatari.
- *Integrità*: il messaggio non deve subire alterazioni. Nelle reti di telecomunicazione esistono già dei meccanismi per assicurare l'integrità dei dati trasmessi, ma in generale si tratta di semplici algoritmi di checksum. I checksum sono adatti per rilevare guasti nella rete ma non forniscono le stesse garanzie di un valore hash.
- *Autenticità*: il messaggio deve essere quello originale, ovvero non deve essere sostituito nel percorso tra mittente e destinatario.
- *Non ripudio*: mittente e destinatario non possono disconoscere di aver inviato e ricevuto il messaggio.

Di seguito mostriamo alcuni esempi di utilizzo delle funzioni hash.

### **Chi ha risolto per primo il problema?**

Tizio e Caio devono risolvere un problema la cui soluzione indichiamo con  $S$ . Tizio sostiene di averlo risolto, e Caio vorrebbe comunque risolverlo e nel frattempo assicurarsi che Tizio non stia bluffando. Che fare? Tizio concorda con Caio una funzione hash  $fh$ , sceglie un numero casuale  $N_c$  e calcola  $fh(S|N_c) = h$ . A questo punto comunica a Caio  $h$ , tenendo per sé  $S$  e  $N_c$ . Quando, successivamente, Caio arriva alla soluzione, potrà verificare se effettivamente Tizio ci era arrivato prima: sarà sufficiente che Tizio comunichi  $N_c$ , e a questo punto dovrà essere verificata la condizione  $fh(S|N_c) = h$ . Ovviamente il valore  $h$  sarà lo stesso solo se entrambi sono arrivati alla stessa soluzione  $S$ .

### **Integrità**

Verifica dell'integrità di un messaggio  $M$ . Determinare  $fh(M) = h$  prima dell'invio del messaggio "fotografa" l'identità del testo prima del passaggio, e quindi lo tutela da un eventuale alterazione durante la trasmissione. Il destinatario potrà infatti verificare l'integrità calcolando a sua volta  $h$ . Questo schema non garantisce tuttavia l'autenticità e non consente il non ripudio. Vedremo quindi come garantirli con la firma digitale.

## Identificazione di file

Le funzioni hash vengono utilizzate anche nell'ambito di sistemi per l'identificazione di file. Ad esempio molti sistemi di gestione del codice sorgente ([source code management](#)), come [Git](#), [Mercurial](#) e [Monotone](#), determinano lo SHA-1 di diversi tipi di contenuti (file, directory, etc) per identificarli univocamente. Quando questi sistemi non hanno a che fare con la sicurezza l'identificazione di un file rientra anche negli usi non crittografici delle funzioni hash.

## Gestione dell'autenticazione

I metodi tramite i quali un essere umano può [autenticarsi](#) ad un sistema informatico sono numerosi: [impronte digitali](#), [impronta vocale](#), modello retinico, chiavi hardware, password, pin, etc. Le funzioni hash sono di notevole importanza nel caso dell'autenticazione tramite user name e password, processo che vedremo in pratica con PHP nel secondo capitolo.

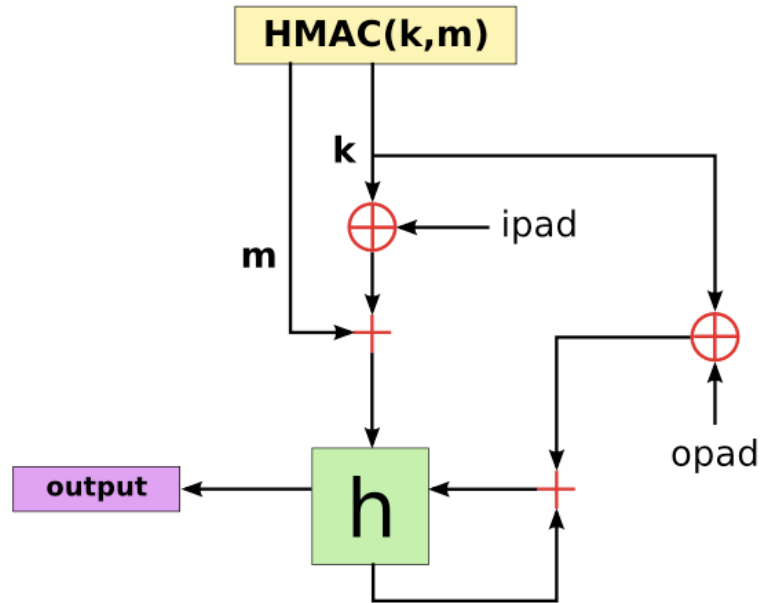
## Derivazione di una chiave crittografica

Derivare una chiave crittografica significa definire una [funzione di derivazione della chiave](#) (in inglese *Key derivation function* o KDF) che fa derivare una o più [chiavi segrete](#) da informazioni che invece sono note. Le funzioni di derivazione di chiavi sono spesso usate come componenti di protocolli di [key-agreement](#) tra più parti.

## Autenticazione di un messaggio

Le funzioni hash vengono inoltre applicate ai sistemi di autenticazione di messaggio (MAC), realizzando così gli [HMAC](#).

HMAC (*keyed-hash message authentication code*) è una tipologia di codice per l'autenticazione di messaggi basata appunto sulle funzioni hash. Tramite HMAC è infatti possibile garantire sia l'integrità che l'autenticità di un messaggio. HMAC utilizza infatti una combinazione del messaggio originale e una chiave segreta per la generazione del codice.



Funzionamento di HMAC ([Wikipedia](#))

### Firma digitale

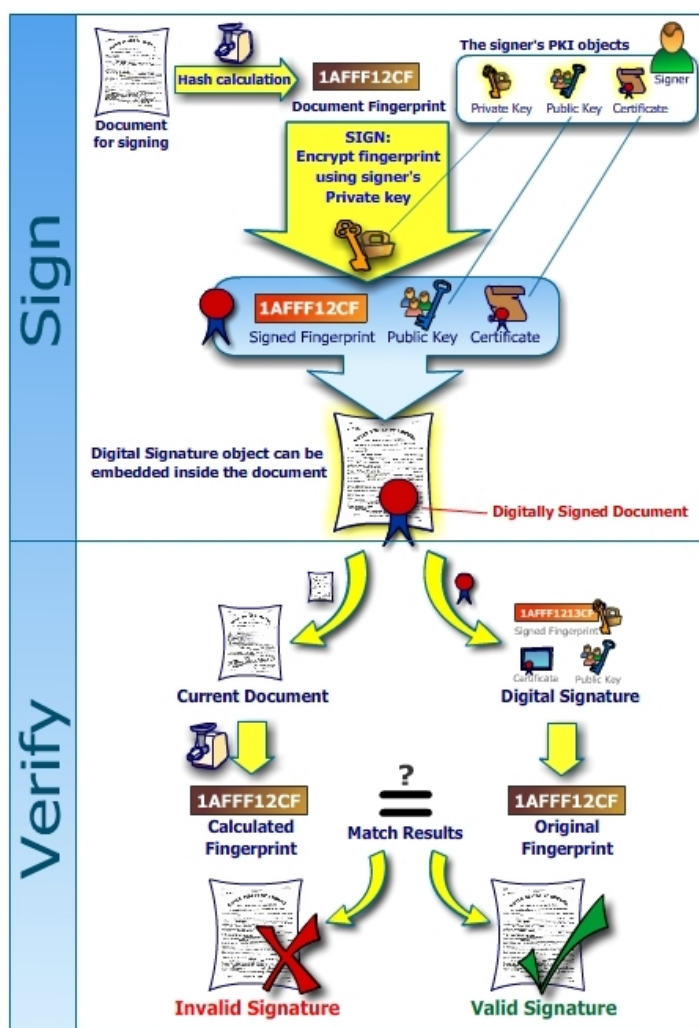
Il sistema per la creazione e la verifica di firme digitali sfrutta le caratteristiche dei [sistemi crittografici asimmetrici](#). Un [sistema crittografico](#) garantisce la riservatezza del contenuto dei messaggi, rendendoli incomprensibili a chi non sia in possesso di una "chiave" per interpretarli. Nei sistemi crittografici a chiave pubblica o asimmetrici, ogni utente ha una coppia di chiavi: una [chiave privata](#), da non svelare a nessuno, con cui può decifrare i messaggi che gli vengono inviati e firmare i messaggi che invia, e una [chiave pubblica](#), che altri utenti utilizzano per cifrare i messaggi da inviargli e per decifrare la sua firma e stabilirne quindi l'autenticità. Per ogni utente, le due chiavi vengono generate da un apposito algoritmo con la garanzia che la chiave privata sia la sola in grado di poter decifrare correttamente i messaggi codificati con la chiave pubblica associata. Lo scenario in cui un mittente vuole spedire un messaggio ad un destinatario in modalità sicura è il seguente: il mittente utilizza la chiave pubblica del destinatario per la codifica del messaggio da spedire, quindi spedisce il messaggio cifrato al destinatario; il destinatario riceve il messaggio cifrato e adopera la sua chiave privata per ottenere il messaggio "in chiaro".

Grazie ad un'ulteriore proprietà delle due chiavi, inversa rispetto a quella descritta, un sistema di questo tipo è adatto anche per ottenere dei documenti firmati, infatti: la chiave pubblica di un utente è la sola in grado di poter decifrare correttamente i documenti codificati con la chiave privata di quell'utente. Se un utente vuole creare una firma per un documento, procede nel modo seguente: con l'ausilio di una funzione hash ricava l'impronta digitale del documento, il message digest, un file di dimensione fissa che riassume le informazioni contenute nel documento, dopodichè

utilizza la propria chiave privata per cifrare quest'impronta digitale: il risultato di questa codifica è la creazione di una firma. La firma prodotta dipende dall'impronta digitale del documento e, quindi, dal documento stesso, oltre che dalla chiave privata dell'utente. A questo punto la firma viene allegata al documento.

Chiunque può verificare l'autenticità di un documento: per farlo, decifra la firma del documento con la chiave pubblica del mittente, ottenendo l'impronta digitale del documento, e poi confronta questa con quella che si ottiene applicando la funzione hash, pubblica, al documento; se le due impronte sono uguali, l'autenticità del documento è garantita.

In precedenza abbiamo detto che la firma digitale assicura il "non ripudio": infatti il firmatario di un documento trasmesso non può negare di averlo inviato, né può il ricevente negare di averlo ricevuto. Detta in altre parole significa che l'informazione non può essere disconosciuta, come nel caso di una firma "convenzionale" su un documento cartaceo in presenza di testimoni ([Wikipedia](http://it.wikipedia.org)).



Funzionamento della firma digitale ([Wikipedia](http://it.wikipedia.org))

Di seguito riportiamo le principali differenze tra firma digitale e firma convenzionale.

	<b>Firma autografa</b>	<b>Firma digitale</b>
<b>Creazione</b>	manuale	mediante algoritmo di creazione
<b>Apposizione</b>	sul documento: la firma è parte integrante del documento	come allegato: il documento firmato è costituito dalla coppia (documento, firma)
<b>Verifica</b>	confronto con una firma autenticata: metodo insicuro	mediante algoritmo di verifica pubblicamente noto: metodo sicuro
<b>Documento copia</b>	distinguibile	indistinguibile
<b>Validità temporale</b>	illimitata	limitata
<b>Automazione dei processi</b>	non possibile	possibile

Differenze tra firma digitale e firma convenzionale ([Wikipedia](#))

Quale funzione hash utilizzare per realizzare la firma digitale? E' interessante sottolineare che una delle funzioni maggiormente utilizzate, MD5, non è prevista dalla normativa italiana e in particolare dalle *Regole tecniche per la formazione, la trasmissione, la conservazione, la duplicazione, la riproduzione e la validazione, anche temporale, dei documenti informatici* del DPCM del 13 gennaio 2004. Tali regole impongono invece l'uso delle funzioni SHA-1 o RIPEMD-160, considerate, non a torto, più sicure. Una valutazione tecnica questa che però andrebbe aggiornata: come vedremo in seguito anche lo SHA-1 presenta alcune vulnerabilità. Il CNIPA ha poi prodotto le [Linee guida per l'utilizzo della Firma Digitale](#) (Versione 1.1 – maggio 2004).

## SSL e TLS

HTTPS (*Hypertext Transfer Protocol over Secure Socket Layer*) è uno schema [URI](#) (*Uniform Resource Identifier*) usato per indicare una connessione HTTP sicura: a differenza dell'HTTP gli accessi vengono effettuati sulla porta 443 e tra il protocollo [TCP](#) e [HTTP](#) si interpone un livello di crittografia/autenticazione.

In pratica viene creato un canale di comunicazione cifrato tra il client e il server attraverso lo scambio di certificati; una volta stabilito questo canale al suo interno viene utilizzato il protocollo HTTP per la comunicazione. Vediamo cosa succede in pratica con un esempio. Quando, navigando, si incontra un link gestito con HTTPS: il protocollo di comunicazione tra client (il browser) e il server è il *Secure Socket Layer* (SSL). SSL opera tra il livello application e quello transport e protegge l'intero flusso di dati che passa sulla connessione TCP, per cui può essere usato sia con HTTP che con FTP o TELNET.

Vediamo ora l'esempio di una connessione ad un sito che presenta una form per l'ordine di un bene attraverso l'immissione del numero di una carta di credito (<https://www.server.com/order.html>).

Collegandosi a questa URL accadono i seguenti passi:

1. il client web richiede al server web la form order.html
2. il server web, oltre a restituire la form richiesta, invia il proprio certificato
3. il client utilizza la chiave pubblica del server per cifrare la chiave segreta di sessione
4. a questo punto può iniziare la transazione sicura cifrata con chiave segreta

In SSL il livello di trasporto include un check dell'integrità del messaggio basato su un apposito MAC (*Message Authentication Code*) che utilizza funzioni hash. In questo modo si verifica che i dati spediti tra client e server non siano stati alterati durante la trasmissione.

In questo paragrafo ci siamo riferiti a SSL, predecessore dell'attuale [TLS](#). La modalità di impiego delle tecniche di hash è rimasta immutata, mentre la tipologia di algoritmo è passata da MD5/SHA-1 per SSL a HMAC per TLS. Osserviamo poi che molte funzioni hash (in particolare SHA-1) sono utilizzate per prodotti e protocolli come [PGP](#), [S/MIME](#), e [IPsec](#).

### **Congelamento dei dati nelle attività di computer forensics**

La [computer forensics](#) è una disciplina che si occupa della tutela, dell'identificazione, dell'analisi del contenuto di un computer o di un sistema informativo, al fine di individuare prove per scopi di indagine giudiziaria. Ad esempio tecniche di CF sono usate per analizzare i computer di persone indagate (in caso di reati) o parti in causa (cause civili). Ovviamente durante un'investigazione informatica è necessario che le prove digitali raccolte non vengano alterate. Un file, ad esempio, deve rimanere integro dal momento della raccolta fino a quando verrà valutato da un perito. E allora le funzioni hash diventano uno strumento indispensabile. Prima di ogni analisi, un investigatore congela i dati mediante la loro impronta digitale. Esistono numerosi software adatti per questa operazione, il cui impiego è immediato. Ad esempio con [SlavaSoft FSUM](#) l'istruzione con cui effettuare un hash (SHA-512) dei file di testo presenti sul disco C è la seguente:

```
C:\test>fsum.exe -sha512 *.txt > impronta_digitale_file_testo.txt
```

### **Catena Hash**

Una catena hash è un metodo per produrre chiavi one-time (monouso) a partire da una singola chiave o password.

Le applicazioni sono molto diffuse: ad esempio molte banche inseriscono nel kit dell'home banking un dispositivo che genera chiavi da usare una sola volta in fase di autenticazione e/o di operazioni sul conto.

Da un punto di vista formale una catena hash è definita dall'iterazione di una funzione hash  $fh$  su una stringa; con  $fh(fh(fh(x)))$  ad esempio indichiamo una catena di dimensione 3. Un'altra notazione per indicare la stessa catena è  $fh^3(x)$ .

## 1.5 Hashing non crittografico

Riportiamo ora alcuni utilizzi non crittografici delle funzioni hash.

### Generazione numeri pseudocasuali

Nell'ambito di alcuni algoritmi per la generazione di numeri pseudocasuali vengono utilizzate funzioni hash. Molti algoritmi prevedono che la funzione di calcolo  $seed(x)$ , che inizializza il generatore di base dei numeri casuali, si basi appunto sul valore di hash di  $x$ . Laddove  $x$  non è presente o non è fornito dall'utente, viene preso in considerazione il tempo di sistema.

### Gestione file audio e video

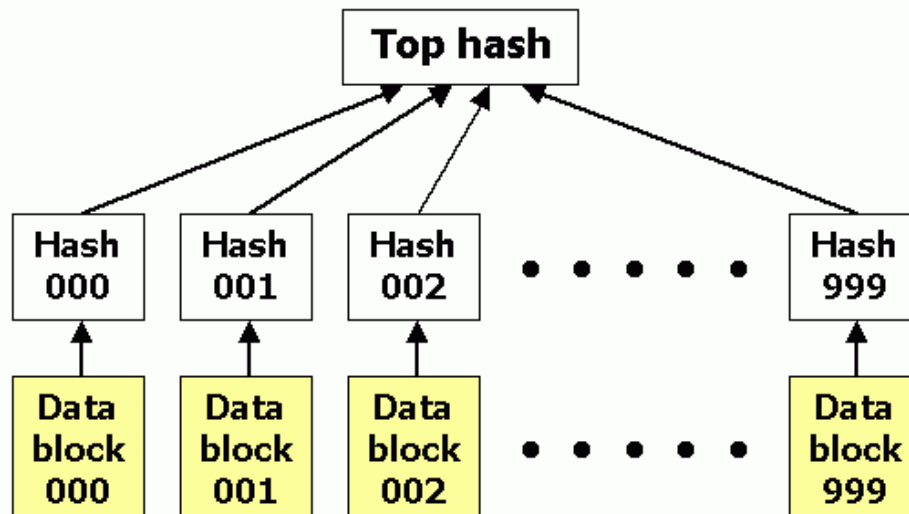
Le funzioni hash, nell'ambito della gestione di video, possono essere utilizzate per diversi scopi:

- Gestione del download di file. Ad esempio nel servizio [ed2k link](#), una funzione hash basata su MD4 viene utilizzata per identificare l'origine del file, scaricarlo e verificarne il contenuto. Occorre notare che, non essendo in un ambito crittografico, si necessita di algoritmi veloci anche se non sicuri.
- Monitoraggio della qualità/integrità di un file dopo che è stato trasmesso, sia nella sua totalità che in streaming.
- Sincronizzazione audio-video quando l'audio e il video sono trasmessi su canali separati.
- Identificazione e confronto. Generalmente per confrontare due immagini/audio/video si considerano i rispettivi valori di hash (valori di alcuni componenti del file): se coincidono allora si considerano i due contenuti uguali. Tuttavia spesso accade che due video, pur essendo uguali, presentano una piccola differenza dei file (e.g. video codec), piccola differenza che, come sappiamo, si risolve in un enorme cambiamento del valore di hash. Ecco quindi che il solo confronto tra hash non basta, anzi è fuorviante. Ecco allora che vengono prese in

considerazione anche funzioni - non adatte ai sistemi crittografici - come la [perceptual hash](#), il cui valore di hash cambia solo se il valore di input (l'entrata) ha avuto determinati cambiamenti. In tal caso il valore di hash resta lo stesso se cambiano alcune proprietà del file e/o se, ad esempio, l'ingresso ha subito solo un cambiamento del contrasto o sono andati persi alcuni frames. E proprio sulle funzioni di perceptual hash si basa il [digital video fingerprinting](#), una tecnica molto recente per l'identificazione di video che può essere considerata come l'evoluzione dei sistemi basati su confronto dei valori hash. I campi di applicazione del digital video fingerprinting sono il [Digital Rights Management](#) (DRM), il monitoraggio della pubblicità e dei media online e anche il controllo dei video trasmessi in TV (servizio sempre più richiesto sia dai provider di contenuti che dai produttori dei video).

### Lista hash

Una lista hash è una lista di valori hash di blocchi dati relativi a uno o più file. Questa tipologia di lista è spesso utilizzata per le tabelle hash, di cui parleremo tra breve.



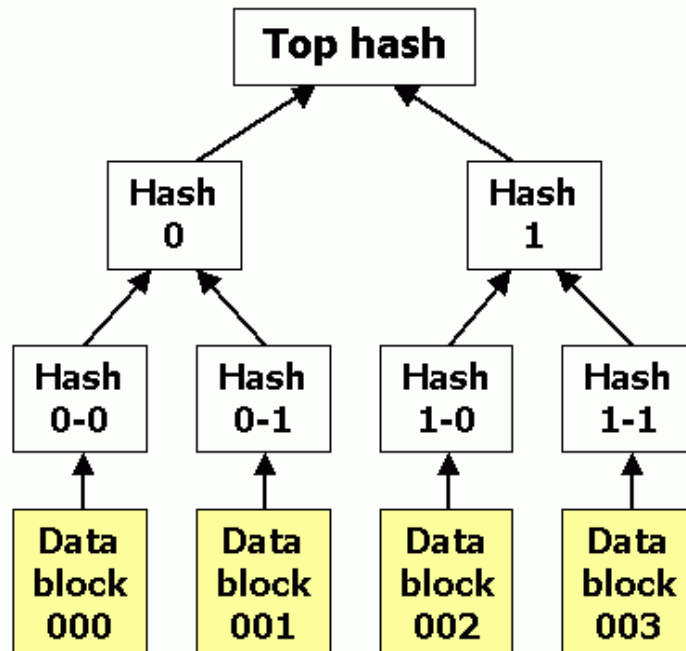
Una lista hash con un hash top ([Wikipedia](#))

Un'evoluzione delle liste hash è data dagli alberi di hash.

### Albero hash

Un albero hash è una struttura dati costituita da un albero i cui nodi rappresentano una sintesi dei dati dei nodi figli. Tale sintesi viene appunto generata mediante l'applicazione di funzioni hash. La principale applicazione degli alberi hash è la verifica dei contenuti di uno o più file.

Quando la funzione hash è Tiger, la struttura prende il nome di albero Tiger.



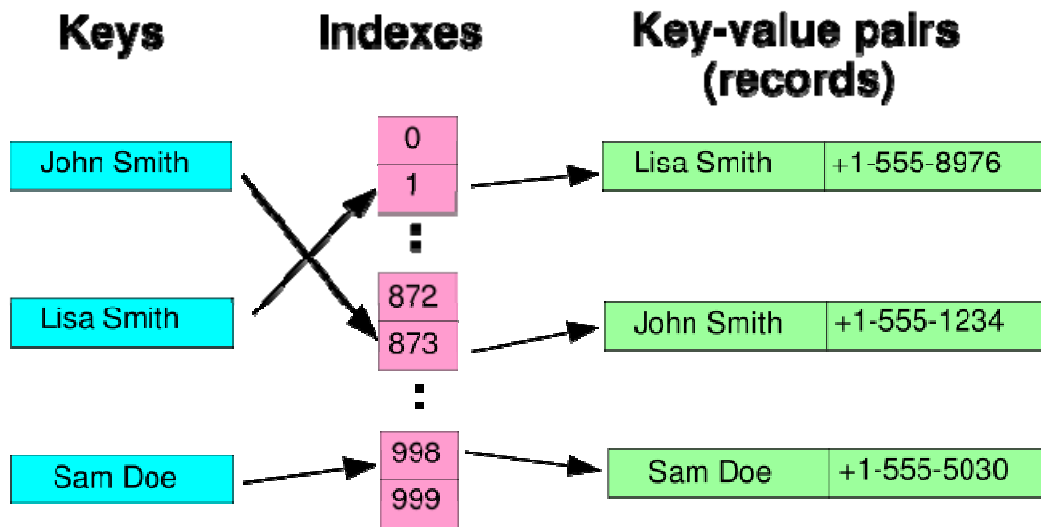
Albero hash ([Wikipedia](#))

### **Tabelle Hash e tabelle hash distribuite**

Una [tabella hash](#) è una struttura dati utilizzata per l'implementazione di [strutture dati astratte](#) che associano chiavi a valori.

Una delle applicazioni più immediate è quella della ricerca di informazioni: data una chiave K (ad esempio il nome di una persona) occorre trovare il corrispondente valore (il numero di telefono). Come funziona la ricerca basata su una tabella hash? Il funzionamento, illustrato nella figura sottostante, si articola in tre passi:

1. Trasformazione della chiave tramite una funzione hash:  $fh(K) = h$
2. Trasformazione di h in m tramite  $m = \text{mod}(h)$  o tramite [bit masking](#)
3. Utilizzo di m come indice in un vettore che identifica la corretta posizione del valore corrispondente



Un'agenda telefonica strutturata come una tabella hash ([Wikipedia](#))

Le tabelle hash sono impiegate quindi per la ricerca ma anche per l'inserimento e cancellazione di record in modo rapido (complessità in tempo  $O(1)$ ) ed efficiente. Anche Google le impiega, ad esempio per la [libreria SparseHash](#).

La voce inglese di Wikipedia ([Hash table](#)) presenta degli ottimi approfondimenti sulla gestione delle collisioni, sul ridimensionamento delle tabelle e sulla scelta della particolare funzione hash da impiegare.

A partire dalle tabelle hash esistono poi altre strutture tra le quali le [tabelle hash distribuite](#), in inglese *distributed hash tables*, indicate anche come DHTs.

Le DHTs sono una classe di sistemi distribuiti decentralizzati che partizionano l'appartenenza di un set di chiavi tra i nodi partecipanti, i quali possono inoltrare in maniera efficiente i messaggi all'unico proprietario di una determinata chiave. Ciascun nodo è l'analogo di un *array slot* di una tabella hash.

Le DHTs sono tipicamente progettate per gestire un vasto numero di nodi, anche nei casi in cui ci siano continui ingressi o improvvisi guasti di alcuni di essi. Questo tipo di infrastruttura può essere utilizzata per implementare servizi più complessi, quali [file system distribuiti](#), sistemi peer-to-peer di file sharing, [web caching](#) cooperativo, [multicast](#), [anycast](#) e [domain name services](#).

E in effetti tra i servizi più noti che utilizzano le DHTs ci sono [Napster](#), [Gnutella](#) e [Freenet](#), [BitTorrent](#), [eMule](#) e nel [Coral Content Distribution Network](#).

## Steganografia

Il termine steganografia deriva dal greco *stèganos*, nascosto, e *grafèin*, scrivere. E' un sistema per nascondere le informazioni. Mentre con la crittografia si rende incomprensibile un messaggio, con la steganografia il messaggio viene nascosto (ma non cifrato). Possiamo dire che la steganografia è l'arte di nascondere l'esistenza stessa della comunicazione. Non c'è alcun messaggio da decifrare, perché il messaggio stesso è reso invisibile, o quasi. La steganografia, rispetto alla cugina crittografia, ha il vantaggio di proteggere il messaggio dalla sua apparente inesistenza, a differenza di un messaggio cifrato che invece è illeggibile, ma visibile. Se infatti si vuole nascondere un piccolo documento tra milioni di pagine ci si trova davanti al famoso ago nel pagliaio, difficilissimo da trovare perché difficilmente riconoscibile da tutto il resto. Tuttavia oggi la steganografia viene utilizzata in numerosi campi: per nascondere del software in alcuni programmi (una sorta di marchio di sicurezza per comprovare l'autenticità del prodotto o per tutelare il diritto d'autore), per lo scambio di informazioni segrete tra esponenti della criminalità organizzata (e informatizzata) e del terrorismo internazionale. E per migliorare alcuni sistemi di crittografia. Infatti una delle tecniche di steganografia più raffinate consiste nel modificare un file inserendovi delle informazioni segrete.

Riportiamo ora un esempio, a cura di Umberto Cerruti, di steganografia applicata ad un'immagine. Qui sotto vedete due icone di Elia. Una delle due immagini (la prima) contiene un messaggio segreto, inserito con il programma [steghide](#) (per chi fosse curioso e volesse scoprirlo: la password è *sonocurioso*; nella directory di lavoro si troverà un file dal nome testo.txt).



Immagine Elia (uno)



Immagine Elia (due)

Alla base degli strumenti per verificare l'integrità delle immagini ed eventualmente identificare messaggi nascosti ci sono appunto le funzioni hash.

Da notare, infatti, che seppur le immagini siano praticamente uguali, il loro valore hash è sensibilmente diverso (il c.d. effetto valanga di cui abbiamo parlato in precedenza):

Hash (tiger192) di elijahUNO.jpg:

7EA561EE8A82B4C4E06C2C9FED319B9DD4DE946F70FAC045

Hash (tiger192) di elijahDUE.jpg:

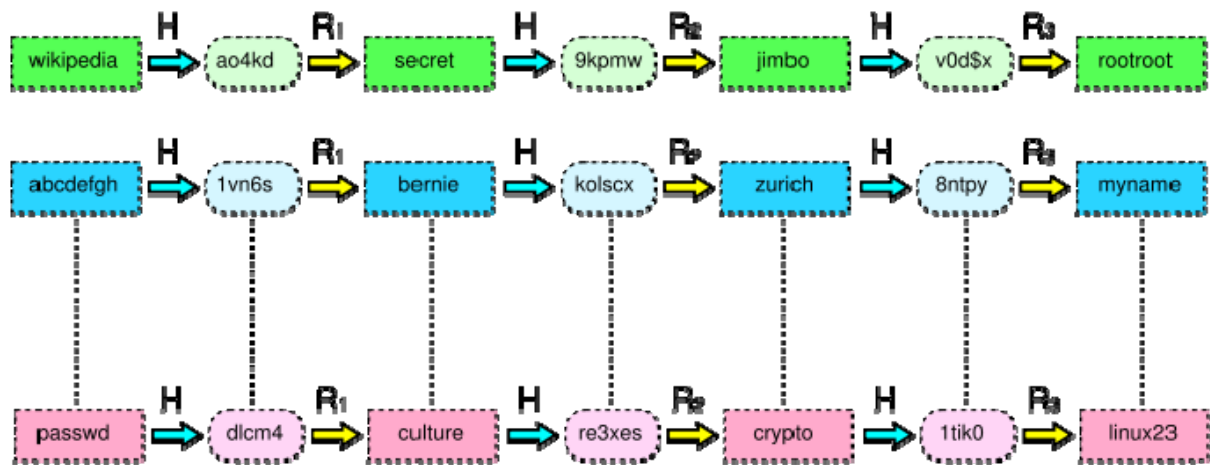
112F29EEC73A4910307688E73156D7AAAF68138F1A8C9BC9

## 1.6 Possibili attacchi ai sistemi protetti da funzioni hash

Esistono diverse tipologie di attacco informatiche che riguardano le funzioni hash. Le principali sono:

- Generazione di collisioni (detto anche [attacco del compleanno](#))  
Abbiamo visto che il tallone d'Achille di diverse funzioni hash è la non resistenza alle collisioni. Questo tipo di attacco ha come obiettivo l'identificazione di due distinti input che generino lo stesso valore di hash, ovvero sia l'individuazione della coppia (M1, M2) tale che  $fh(M1) = fh(M2)$ . In altre parole, occorre dimostrare che la funzione oggetto di attacco non goda della proprietà di resistenza forte alle collisioni.
- Preimmagine  
Quando invece l'attacco è finalizzato ad identificare un input M2 che generi lo stesso message digest di un altro input M1 sconosciuto, si parla di preimage attack. In altre parole, occorre dimostrare che la funzione oggetto di attacco non goda della proprietà di resistenza debole alle collisioni. Questo punto debole è ovviamente molto gradito ai cracker di password: se da un lato non è possibile invertire un valore di hash, dall'altro, generando una collisione, è possibile trovare un'altra stringa il cui hash corrisponda al message digest memorizzato. Esistono due tipi di attacco preimmagine:
  - *First preimage attack*: dato un valore di hash  $h$ , trovare un input  $M$  tale che  $fh(M) = h$
  - *Second preimage attack*: dato un input  $M1$ , trovare un input  $M2$  diverso dal primo tale che  $fh(M1) = fh(M2)$

- Forza bruta  
In generale un attacco a forza bruta è un metodo per rompere uno schema crittografico che prevede l'utilizzo di un grande numero di tentativi. Ad esempio si potrebbero provare tutte le possibili combinazioni alfanumeriche per trovare una password. Come è ovvio alcuni di questi tentativi hanno un costo computazionale troppo elevato, e allora sono preferibili sistemi di attacco più rapidi, come ad esempio quello a dizionario o rainbow table.
- Dizionario  
L'attacco a dizionario si basa sull'utilizzo di uno o più dizionari. Si potrebbe infatti presumere (e non a torto) che una password di posta elettronica sia composta da uno o più termini della propria lingua, e allora, invece di compiere un attacco a forza bruta, si restringe l'insieme di tentativi a quelli presenti del dizionario. Generalmente questo tipo di attacco prevede uno o più dizionari e un insieme di regole di trasformazione delle parole che estenda ulteriormente il dizionario. I programmi software di password cracking eseguono un confronto tra i message digest dei termini presenti nella lista (con le variazioni indotte dalle trasformazioni) e il valore di hash di cui non conosciamo il testo in chiaro (password). L'attacco a dizionario è decisamente più veloce di quello a forza bruta ma non garantisce il raggiungimento dell'obiettivo.
- Tabelle rainbow  
L'attacco basato sulle tabelle rainbow è un'evoluzione del metodo a forza bruta e a dizionario. L'elemento innovativo consiste nel precalcolare tabelle di hash e poi confrontare i valori di hash delle tabelle con quello da crackare. In questo modo il costo computazionale è ridotto al solo confronto tra valori, avendo già applicato la funzione hash in precedenza. E' chiaro quindi come un attacco a dizionario con i termini già trasformati in message digest possa garantire una performance migliore. La creazione di tabelle rainbow va progettata con cura e può richiedere - per la generazione - diversi giorni. Dato che le dimensioni sono spesso nell'ordine dei GB, esistono diversi servizi online (gratuiti quelli con tabelle "base", a pagamento quelli che offrono rainbow di grandi dimensioni) che consentono di scaricare o utilizzare tabelle precalcolate.



Schema del funzionamento di una tabella rainbow (con 3 funzioni di riduzione) ([Wikipedia](#))

## 1.7 Miglioramento della sicurezza (contromisure per le collisioni)

La prima considerazione da fare è di evitare innanzitutto quegli algoritmi di hashing per i quali siano stati generati collisioni. E' una premessa meno banale di quello che sembra, dato che uno degli algoritmi più utilizzati è l'MD5.

Un modo per ridurre praticamente a zero le probabilità di una collisione è, come indicato da [Bruce Schneier](#), utilizzare due funzioni hash in cascata, vale a dire applicare la stessa funzione al message digest del messaggio ( $\text{hash}(\text{hash}(m))$ ), oppure anche al message digest concatenato col messaggio stesso  $\text{hash}(\text{hash}(m)|m)$ . Ovviamente ciò comporta un costo computazionale maggiore e non sempre può essere applicato.

Vediamo ora qualche indicazione per la protezione delle password memorizzate in un database. In molti sistemi esiste una o più tabelle contenenti le password degli utenti, dati particolarmente importanti che devono essere tutelati adeguatamente. Nell'ipotesi che un intruso riesca ad accedere a queste informazioni è necessario quindi dotarle di un sistema di autodifesa insito nella loro stessa rappresentazione. Se è chiaro che le password non vanno scritte in chiaro, anche l'utilizzo di un semplice hash può esporre il sistema ad uno degli attacchi visti in precedenza. In alcuni casi potrebbe essere sufficiente un solo tentativo: in una grande azienda dove molti dipendenti utilizzano la stessa password (e.g. il nome dell'azienda) la tabella dove sono conservati i valori hash presenta numerosi valori identici e quindi facilmente riconoscibili.

Come difendere la tabella delle password? Un metodo molto efficace è aggiungere ai valori hash delle password un insieme di bit casuali,

soprannominato salt (il sale nell'immagine di copertina!), e progettare i sistemi in modo tale che lavorino sui valori di hash concatenati ai salt. In altre parole l'hash non viene applicato solo alla password, bensì alla concatenazione di questa con il salt. Utilizzando la notazione vista in precedenza,  $h(\text{password}|\text{salt})$ . Il valore del salt viene poi memorizzato in chiaro nel valore hash così calcolato. Anche se due utenti hanno la stessa password, il valore archiviato differisce tra i due. Quando l'utente effettua l'autenticazione, in base al suo user name ottengo il salt e poi eseguo il comando  $h(\text{password}|\text{salt})$ . In questo caso, con un semplice salt di 16 bit, l'attacco a forza bruta deve provare tutte le parole del dizionario per ogni password. Ciascun bit del salt ha quindi l'effetto di raddoppiare la quantità di memoria e onere computazionale richieste. Per fare un esempio supponiamo di avere come password un vocabolo appartenente al dizionario e che il sistema usi un salt a 32 bit. Sia N il numero di vocaboli presenti nel dizionario. L'intruso deve così calcolare l'hash di ciascuna delle N parole del dizionario concatenata a ciascuno dei possibili salt ( $2^{32}=4.294.967.296$ ). La complessità dell'algoritmo è dell'ordine di  $N \times 2^{32}$ , invece della complessità di ordine N che richiesta in assenza del salt. L'utilizzo del salt è d'obbligo nei sistemi in cui i requisiti in termini di sicurezza siano elevati (e.g. home banking, tutela di dati sensibili, etc).

## 1.8 Panoramica sugli algoritmi di hash

Vediamo ora una breve panoramica di alcuni algoritmi di hash, per poi passare alle applicazioni pratiche con PHP.

### MD

La famiglia delle funzioni MD (Message Digest) rappresenta l'inizio della storia degli algoritmi di hash. Senza soffermarsi sulle prime (e superatissime) versioni, l'MD5 venne progettata nel 1991 da [Ronald Rivest](#), uno dei padri della [crittografia asimmetrica](#), già autori delle precedenti versioni MD. L'MD5 nasce dall'esigenza di migliorare l'[MD4](#), veloce ma inaffidabile. Tuttavia anche l'MD5, oggi, è da considerarsi insicura. La storia dei suoi fallimenti è legata alle collisioni: nel 1993 fu individuata una [pseudo collisione](#), nel 1996, nel 2004 e nel 2005 furono generate delle collisioni, mentre nel 2006 è stato realizzato un algoritmo che trova collisioni in meno di un minuto. Nonostante ciò l'MD5 rimane una delle funzioni più utilizzate.

### SHA

Con [Secure Hash Algorithm](#) (SHA) si indicano i cinque algoritmi di cui il primo, nato nel 1993, è passato sotto la supervisione della [National Security Agency](#) (NSA) che, nel 1996, ha rilasciato una versione corretta, chiamata SHA-1. Il suo funzionamento è molto simile a quelle dell'MD5 ma è più

resistente agli attacchi. I cinque algoritmi sono chiamati *SHA-1*, *SHA-224*, *SHA-256*, *SHA-384*, e *SHA-512*. Le ultime quattro varianti sono talvolta chiamate SHA-2. SHA-1 produce un *message digest* di 160 bit. Il numero nel nome delle varianti indica il numero di bit prodotti. Le funzioni SHA-2 sono coperte da brevetto. La sicurezza di SHA-1 è stata in parte compromessa in diversi esperimenti. Nel 2007 il [National Institute of Standards and Technology](#) (NIST) ha lanciato un [concorso pubblico per la realizzazione di una nuova funzione SHA-3](#): "Il NIST sta avviando un concorso per lo sviluppo di uno o più algoritmi di hashing aggiuntivi attraverso una competizione pubblica". Le iscrizioni si concluderanno il 31 ottobre 2008 e la proclamazione del vincitore e la pubblicazione del nuovo standard sono previste per il 2012. Nel frattempo il NIST suggerisce l'utilizzo di SHA-256 o superiori.

## **RIPEMD**

[RIPEMD](#) ([RACE Integrity Primitives Evaluation Message Digest](#)) è nato nel 1992 nell'ambito del progetto europeo RIPE ([RACE Integrity Primitives Evaluation](#)) ad opera dalle stesse persone che avevano condotto attacchi efficaci contro l'MD5. Pur essendo meno diffuso degli SHA, RIPEMD vanta il pregio di non esser coperto da alcun brevetto. Ne esistono diverse versioni a seconda della dimensione del message digest e del funzionamento dell'algoritmo: RIPEMD (simile all'MD4 e con performance analoghe a SHA-1) fu presto sostituito da RIPEMD-128. Poi le versioni successive furono quelle a 160 bit (1996), a 256 bit e a 320 bit. Occorre sottolineare come RIPEMD-256 e 320 diminuiscano le probabilità di collisioni casuali ma che complessivamente non forniscano un livello di sicurezza maggiore delle versioni a 128 e 160 bit. L'algoritmo più utilizzato è RIPEMD-160.

## **WHIRLPOOL**

[WHIRLPOOL](#) è una funzione hash che produce un digest di 512 bit. Esistono tre versioni: WHIRLPOOL-0, migliorata e trasformata poi in WHIRLPOOL-T e infine WHIRLPOOL. Quest'ultima versione è stata creata dopo che nel 2003 sono state individuate delle imperfezioni nella WHIRLPOOL-T. Riportiamo, a titolo di esempio, una frase sottomessa ai tre algoritmi WHIRLPOOL.

```
Whirlpool-0("The quick brown fox jumps over the lazy dog") =  
4F8F5CB531E3D49A61CF417CD133792CCFA501FD8DA53EE368FED20E5FE0248C  
3A0B64F98A6533CEE1DA614C3A8DDEC791FF05FEE6D971D57C1348320F4EB42D
```

```
Whirlpool-T("The quick brown fox jumps over the lazy dog") =  
3CCF8252D8BBB258460D9AA999C06EE38E67CB546CFFCF48E91F700F6FC7C183  
AC8CC3D3096DD30A35B01F4620A1E3A20D79CD5168544D9E1B7CDF49970E87F1
```

```
Whirlpool("The quick brown fox jumps over the lazy dog") =  
B97DE512E91E3828B40D2B0FDCE9CEB3C4A71F9BEA8D88E75C4FA854DF36725F  
D2B52EB6544EDCACD6F8BEDDFEA403CB55AE31F03AD62A5EF54E42EE82C3FB35
```

Anche in questo caso è possibile osservare l'effetto prodotto da un piccolo cambiamento, che stravolge l'intero message digest:

```
Whirlpool-0("The quick brown fox jumps over the lazy eog") =  
228FBF76B2A93469D4B25929836A12B7D7F2A0803E43DABA0C7FC38BC11C8F2A  
9416BBCF8AB8392EB2AB7BCB565A64AC50C26179164B26084A253CAF2E012676
```

```
Whirlpool-T("The quick brown fox jumps over the lazy eog") =  
C8C15D2A0E0DE6E6885E8A7D9B8A9139746DA299AD50158F5FA9EECDDEF744F9  
1B8B83C617080D77CB4247B1E964C2959C507AB2DB0F1F3BF3E3B299CA00CAE3
```

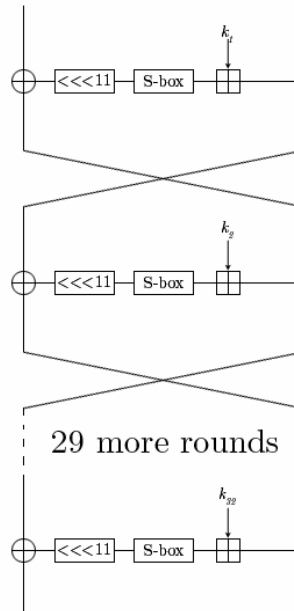
```
Whirlpool("The quick brown fox jumps over the lazy eog") =  
C27BA124205F72E6847F3E19834F925CC666D0974167AF915BB462420ED40CC5  
0900D85A1F923219D832357750492D5C143011A76988344C2635E69D06F2D38C
```

## TIGER

[Tiger](#) è una funzione hash creata nel 1995 ed ottimizzata per piattaforme a 64 bit. Produce un hash value di 192 bit ma ne esistono anche versioni a 128 e 160 bit. Con Tiger2 si indica lo stesso algoritmo con la sola differenza di utilizzare la stessa tipologia di *padding* dello SHA piuttosto che dell'MD4. Tiger viene utilizzato anche nell'ambito degli alberi hash visti in precedenza ([Tiger Tree Hash](#), TTH). TTH è impiegato nei protocolli di sharing [Gnutella](#), [Gnutella2](#), [Direct Connect P2P](#) e nelle applicazioni di file sharing come [Phex](#), [BearShare](#), [LimeWire](#), [Shareaza](#), [DC++](#) e [Valknut](#). Lo standard [OpenPGP](#) in un primo momento ha incluso Tiger per poi optare per RIPEMD-160.

## GOST

GOST è una funzione hash creata in Russia che genera message digest di 256 bit ed è basata sul celebre algoritmo [GOST block cipher](#).



Funzionamento del GOST block cipher (immagine di Decrypt3)

Nel corso dell'evento "[Crypto 2008](#)" (*28th International Cryptology Conference*) diversi ricercatori hanno mostrato i progressi raggiunti negli attacchi alle funzioni hash, ed in particolare hanno preso di mira GOST. In fatti questo algoritmo è stato fortemente criticato perchè reso obbligatorio negli uffici governativi russi. Tuttavia ancora non è possibile parlare di attacchi vincenti contro questo algoritmo a meno di un massiccio impiego di risorse computazionali.

## HAVAL

[HAVAL](#), definito nel 1992, è una funzione hash che genera valori di 128-160-192-224-256 bit. E' anche possibile specificare il numero di giri (3, 4, 5) che l'algoritmo deve effettuare. Al momento non risulta alcun attacco vincente contro questo algoritmo.

## PANAMA e RADIOGATUN

[Panama](#) è stato presentato nel 1998, genera un valore hash di 256 bit con elevate performance. Tuttavia sia nel 2001 che nel 2007 alcuni ricercatori hanno generato delle collisioni. Nel 2006 è stata quindi proposta una variante di Panama di nome RadioGatún, che al momento non risulta essere affetta da collisioni.

Sull'articolo inglese di Wikipedia [Cryptographic hash function](#) è presente una tabella riassuntiva che confronta le principali funzioni hash, dove è possibile vedere quali funzioni sono (per il momento) esenti dalle collisioni. Sono presenti anche altri parametri, tra cui l'*internal state* che indica l'*internal hash sum* dopo ciascuna compressione di blocchi.

Algorithm	Output size (bits)	Internal state size	Block size	Length size	Word size	Collision
<b>HAVAL</b>	256/224/192/160/128	256	1024	64	32	Yes
<b>MD2</b>	128	384	128	No	8	Almost
<b>MD4</b>	128	128	512	64	32	Yes
<b>MD5</b>	128	128	512	64	32	Yes
<b>PANAMA</b>	256	8736	256	No	32	Yes
<b>RadioGatún</b>	Arbitrarily long	58 words	3 words	No	1-64	No
<b>RIPEMD</b>	128	128	512	64	32	Yes
<b>RIPEMD-128/256</b>	128/256	128/256	512	64	32	No
<b>RIPEMD-160/320</b>	160/320	160/320	512	64	32	No
<b>SHA-0</b>	160	160	512	64	32	Yes
<b>SHA-1</b>	160	160	512	64	32	With flaws
<b>SHA-256/224</b>	256/224	256	512	64	32	No
<b>SHA-512/384</b>	512/384	512	1024	128	64	No
<b>Tiger(2)-192/160/128</b>	192/160/128	192	512	64	64	No
<b>WHIRLPOOL</b>	512	512	512	256	8	No

## CAPITOLO DUE

# Hashing con PHP



## 2.1 PHP e funzioni hash

PHP è un linguaggio di scripting interpretato, con licenza open source, originariamente concepito per la realizzazione di pagine web dinamiche. Attualmente è utilizzato principalmente per sviluppare applicazioni web lato server ma può essere usato anche per scrivere script a linea di comando o applicazioni standalone con interfaccia grafica ([Wikipedia](#)). Il successo di PHP è dovuto a diversi fattori tra cui la sua natura open source e freeware, la semplicità di utilizzo, la completezza e la stabilità.

A partire dalla versione 5.1.2, PHP vanta la presenza nel core del *Message Digest Hash Framework*, una libreria di funzioni che permette l'elaborazione di messaggi attraverso algoritmi hash. Questo fatto costituisce un'importante novità; innanzitutto non viene richiesta nessuna installazione, rendendo dunque alla portata di chiunque non sia esperto di installazioni PHP l'utilizzo delle funzioni hash; in secondo luogo il framework rende disponibile pressoché tutti gli algoritmi di hash attualmente di uso corrente. Infine, la semplicità di utilizzo che esso garantisce, fa sì che la scrittura del codice non crei complicazioni al programmatore.

## 2.2 Algoritmi di hash con PHP5

Per rendersi conto di quali siano gli algoritmi disponibili, è sufficiente utilizzare la funzione `hash_algos()` con una versione PHP 5.1.2 o superiore, che restituisce un array i cui elementi contengono i nomi degli algoritmi hash.

In questo modo, utilizzando l'istruzione:

```
print_r(hash_algos());
```

è possibile visualizzare i suddetti algoritmi. Infatti l'output del comando è il seguente:

```
Array (
```

```
[0] => md2  
[1] => md4  
[2] => md5  
[3] => sha1  
[4] => sha256  
[5] => sha384  
[6] => sha512  
[7] => ripemd128  
[8] => ripemd160  
[9] => ripemd256  
[10] => ripemd320  
[11] => whirlpool
```

```
[12] => tiger128,3
[13] => tiger160,3
[14] => tiger192,3
[15] => tiger128,4
[16] => tiger160,4
[17] => tiger192,4
[18] => snefru
[19] => gost
[20] => adler32
[21] => crc32
[22] => crc32b
[23] => haval128,3
[24] => haval160,3
[25] => haval192,3
[26] => haval224,3
[27] => haval256,3
[28] => haval128,4
[29] => haval160,4
[30] => haval192,4
[31] => haval224,4
[32] => haval256,4
[33] => haval128,5
[34] => haval160,5
[35] => haval192,5
[36] => haval224,5
[37] => haval256,5
)
```

Nel seguito del testo confronteremo le performance di alcuni di questi algoritmi.

## 2.3 L'hash di stringhe

Una volta scelto l'algoritmo da utilizzare, se si deve calcolare il digest di una stringa è sufficiente fare uso della funzione **hash**.

```
string hash ( string $algo , string $data [, bool $raw_output ] )
```

La funzione accetta come parametri l'algoritmo richiesto per generare l'hash (in formato stringa, così come riportato dalla funzione `hash_algos()` di cui al paragrafo precedente); la stringa da cifrare; un terzo parametro opzionale booleano (default FALSE) imposta la modalità di visualizzazione del digest calcolato, cioè binario se TRUE, esadecimale se FALSE. Ricordiamo infatti che una funzione hash restituisce sempre un valore binario.

Naturalmente la lunghezza del message digest dipende dall'algoritmo utilizzato. Ad esempio calcoliamo l'hash della stringa "le ultime lettere di Jacopo Ortis" applicando l'algoritmo SHA-1:

```

<?php
    $str = "le ultime lettere di Jacopo Ortis";
    echo($str)."<br>";
    echo("hash esadecimale: ".hash("sha1",$str,false)."<br>");
?>

```

L'output è il seguente:

```

le ultime lettere di Jacopo Ortis
hash esadecimale: 01a882e631b95b33c3871b42390d94ba7c6a6c9c

```

## 2.4 L'hash di un file

Una funzionalità che può rivelarsi utile in diversi contesti è quella che permette di cifrare il contenuto di un intero file.

La definizione della funzione è la seguente:

```

string hash_file ( string $algo , string $filename [, bool $raw_output ] )

```

La funzione viene invocata allo stesso modo di quella hash per cifrare una stringa, con la differenza del nome del file da cifrare al posto della stringa come secondo parametro.

Un esempio di uso della funzione si ha quando si deve accertare l'integrità di un file e, tramite la firma digitale, la sua autenticità.

Un intruso potrebbe modificare uno script secondo propri fini, come ad esempio quello di accedere ad aree del server non autorizzate.

Se da un lato è compito dei sistemisti proteggere in modo adeguato queste aree, dall'altro anche chi realizza applicativi web based con PHP può contribuire significativamente alla sicurezza. Infatti un intruso che riuscisse ad avere accesso alla webroot potrebbe alterare uno script PHP aggiungendo ad esempio le seguenti righe di codice:

```

<?php
    /* recupero nome del file da violare */
    $file=$_GET['file'];
    /* apertura file */
    $handle = fopen($file,"r");
    while (!feof($handle)) {
        /* lettura riga del file */
        $buffer = fgets($handle);
        /* scrittura a video della riga */
        echo $buffer."<br>";
    }
    fclose($handle);
?>

```

Queste poche righe permettono di aprire un file il cui nome e percorso relativo vengono passati direttamente attraverso il browser nella barra degli

indirizzi. Una volta aperto il file, esso viene letto riga per riga e quindi visualizzato.

In tal modo l'intruso può leggere quello che vuole semplicemente scrivendo, ad esempio:

```
http://directory_di_lavoro_php/nome_script.php?file=nome_file_da_leggere
```

ottenendo così la visualizzazione del file che si vuole leggere.

Per evitare o comunque ridurre la possibilità di simili pericoli si può calcolare l'hash dell'intero script PHP, memorizzarlo in un'opportuna area del database, e quindi verificare, ogni volta che questo sta per essere lanciato, che sia quello corretto (e cioè identico a quello memorizzato nel database). Se il file è stato modificato il suo digest sarà differente e questo indicherà inequivocabilmente una sua alterazione.

La funzione seguente si occupa di calcolare l'hash per ciascun file presente nella directory dove si trovano gli script PHP di cui calcolare l'hash, e di inserire nel db il record contenente il nome dello script e il relativo hash, in una tabella tab\_hash con due campi di tipo varchar:

```
<?php

function crea_hash_script($path_directory, $host, $user, $pass) {
    if ($handle = opendir($path_directory)) {
        $conn = mysql_connect($host, $user, $pass );
        $select_db = mysql_select_db('hash_db', $conn);
        mysql_query("START TRANSACTION");
        while (false !== ($file = readdir($handle))) {
            if (($file != ".") && ($file != "..") && eregi("^[a-z0-9_\.-
]+.php$", $file)) {
                $sql = "insert into tab_hash
values('".$file."', '".$hash_file('sha1', $file)."')";
                if (!mysql_query($sql)) {
                    mysql_query("ROLLBACK");
                }
            }
        }
        mysql_query("COMMIT");
        closedir($handle);
        mysql_close($conn);
        return true;
    } else {
        echo("IMPOSSIBILE APRIRE LA DIRECTORY !");
        return false;
    }
}

?>
```

Il codice seguente invece verifica che l'hash dello script da chiamare sia identico a quello presente nel database:

```

<?php
    $hash_preso_da_db = recupera_hash($nome_script);
    $nome_algoritmo = "sha1";
    if (hash_file($nome_algoritmo,$nome_script) != $hash_preso_da_db) {
        echo("ATTENZIONE, IL FILE DA CHIAMARE E' STATO MODIFICATO !");
        exit();
    } else {
        header("Location: ".$nome_script);
    }
?>

```

## 2.5 L'hash di dati eterogenei

Ci sono situazioni in cui si presenta la necessità di determinare il message digest di dati di origine diversa (per esempio il nome di un file e il suo contenuto, oppure più file insieme).

Per queste situazioni, il *Message Digest Hash Framework* mette a disposizione una serie di funzioni che permette di gestire l'hashing di sequenze di dati.

Prima di tutto è necessario creare un cosiddetto "contesto di hashing incrementale", attraverso la funzione `hash_init`.

La definizione è la seguente:

```
resource hash_init ( string $algo [, int $options [, string $key ]] );
```

Il primo parametro, l'unico obbligatorio, rappresenta l'algoritmo di codifica da utilizzare. Il secondo parametro è opzionale e attualmente è previsto un solo valore, cioè quello rappresentato dalla costante globale **HASH\_HMAC**. Quando questo parametro esiste, deve essere obbligatoriamente specificato anche il terzo parametro, che è la chiave segreta (in forma di stringa) da usare nella cifratura.

La funzione restituisce una risorsa, cioè un riferimento ad un contesto di hashing incrementale.

Associata alla `hash_init`, che crea il contesto necessario alla cifratura, è la funzione che invece finalizza l'operazione di cifratura, restituendo il message digest dell'input da cifrare. La dichiarazione è:

```
string hash_final ( resource $context [, bool $raw_output ] ),
```

in cui il primo parametro rappresenta il contesto di hashing, mentre un secondo parametro opzionale booleano (default FALSE) imposta la modalità dell'output restituito, cioè binario se TRUE, esadecimale se FALSE. Naturalmente questa funzione restituisce il message digest calcolato della sequenza di dati in input. Detto questo, resta da capire come fare ad associare i dati da cifrare al contesto creato; questo compito viene eseguito dall'istruzione `hash_update`, la cui dichiarazione è la seguente:

```
bool hash_update ( resource $context , string $data );
```

I parametri sono due, e sono ovviamente il contesto, restituito dalla chiamata alla funzione `hash_init`, e il messaggio da cifrare.

Nell'esempio seguente si vuole generare l'hash del nome di un file insieme al suo contenuto:

```
<?
$nome_algoritmo = "sha1";
$fp = fopen($file, "r");
$ctx = hash_init($nome_algoritmo);
hash_update($ctx, $nome_file);
while (!feof($fp)) {
    $buffer = fgets($fp, 65536);
    hash_update($ctx, $buffer);
}
$hash = hash_final($ctx, true);
fclose($fp);
?>
```

## 2.6 Confronto tra algoritmi utilizzando PHP

Abbiamo effettuato un confronto sui tempi di esecuzione per diversi algoritmi di hashing. Sotto è riportato il codice PHP di uno script che cifra una stringa secondo otto algoritmi, calcola i rispettivi tempi di esecuzione della funzione `hash` e salva i tempi in una tabella `TEMPI` appartenente ad un database MySQL `HASH_DB`. Lo script di creazione della tabella è il seguente:

```
CREATE TABLE tempi (
  id int(10) unsigned NOT NULL auto_increment,
  md5 decimal(15,14) NOT NULL,
  sha1 decimal(15,14) NOT NULL,
  sha512 decimal(15,14) NOT NULL,
  ripemd320 decimal(15,14) NOT NULL,
  whirlpool decimal(15,14) NOT NULL,
  tiger192 decimal(15,14) NOT NULL,
  gost decimal(15,14) NOT NULL,
  haval256 decimal(15,14) NOT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Il codice PHP è il seguente:

```
<?php
include ("include_hash.php");
/* apertura connessione al DB */
$con = connetti();
if ($con == false) {
    exit();
}
```

```

}
/* selezione del DB "hash_db" */
$risultato = mysql_select_db("hash_db", $con);
if (!$risultato) {
    $msg_errore = "CONNESSIONE FALLITA";
    echo($msg_errore);
    exit();
}
/* stringa da cifrare */
$stringa = "le ultime lettere di iacopo ortis";
/* preparazione transazione per inserimento dati nel db */
mysql_query("START TRANSACTION");
/* ciclo per n volte, dove n = dimensione campionaria */
/* ogni esecuzione del ciclo esegue la cifratura della stringa */
/* secondo 8 algoritmi differenti, calcola il tempo di cifratura per
*/
/* ciascuno di essi, e infine inserisce una riga nella tabella
"tempi", */
/* in cui ogni campo contiene il tempo di esecuzione per un
determinato */
/* algoritmo */
$errore = false;
$i=0;
while (($i<N) && ($errore == false)) {
    /* MD5 */
    $time_start = microtime(true);
    $criptata = hash("md5",$stringa);
    $time_end = microtime(true);
    $time = $time_end - $time_start;
    $md5 = $time;
    /* SHA1 */
    $time_start = microtime(true);
    $criptata = hash("sha1",$stringa);
    $time_end = microtime(true);
    $time = $time_end - $time_start;
    $sha1 = $time;
    /* SHA512 */
    $time_start = microtime(true);
    $criptata = hash("sha512",$stringa);
    $time_end = microtime(true);
    $time = $time_end - $time_start;
    $sha512 = $time;
    /* RIPEMD320 */
    $time_start = microtime(true);
    $criptata = hash("ripemd320",$stringa);
    $time_end = microtime(true);
    $time = $time_end - $time_start;
    $ripemd320 = $time;
    /* WHIRLPOOL */
    $time_start = microtime(true);
    $criptata = hash("whirlpool",$stringa);
    $time_end = microtime(true);
    $time = $time_end - $time_start;
    $whirlpool = $time;
    /* TIGER192,4 */
    $time_start = microtime(true);
    $criptata = hash("tiger192,4",$stringa);

```

```

$time_end = microtime(true);
$time = $time_end - $time_start;
$tiger192 = $time;
/* GOST */
$time_start = microtime(true);
$scriptata = hash("gost",$stringa);
$time_end = microtime(true);
$time = $time_end - $time_start;
$gost = $time;
/* HAVAL1256,5 */
$time_start = microtime(true);
$scriptata = hash("haval256,5",$stringa);
$time_end = microtime(true);
$time = $time_end - $time_start;
$haval256 = $time;
/* esecuzione inserimento in tabella */
$sql = "insert into tempi (md5,sha1,sha512,
ripemd320,whirlpool,tiger192,gost,haval256) values ($md5.", ".$sha1.", ".$
$sha512.", ".$ripemd320.", ".$whirlpool.", ".$tiger192.", ".$gost.", ".$haval256."
)";

$query = mysql_query($sql);
if (!$query) {
    echo("OPERAZIONE FALLITA");
    mysql_query("ROLLBACK");
    $errore = true;
}
$i++;
}
/* salvataggio nel DB */
if ($errore == false) {
    mysql_query("COMMIT");
    echo("OPERAZIONE ESEGUITA CON SUCCESSO");
}
/* chiusura connessione al DB */
mysql_close($con);
?>

```

Anche se abbiamo considerato algoritmi che producono message digest di diversa lunghezza (non esiste una lunghezza comune per tutti), il confronto ha restituito informazioni interessanti e coerenti con la natura delle funzioni. Il test è stato eseguito su un PC<sup>1</sup> con piattaforma Microsoft Windows XP, ma risultati analoghi (con tempi di elaborazione minori) - che verranno inseriti nelle versioni successive del documento - sono stati ottenuti anche su piattaforme Linux<sup>2</sup>.

Abbiamo stimato la dimensione ottimale del campione (n = numero di replicazioni degli algoritmi) con un procedimento statistico che prevede un errore di stima massimo pari al 5%. Il risultato è di n = 784. Il valore dell'algoritmo più lento è stato posto uguale all'unità; algoritmi più veloci

---

<sup>1</sup> RAM 1GB - CPU Intel Pentium 4 3GHz

<sup>2</sup> Macchina quadriprocessore con sistema operativo Red Hat Enterprise Linux AS v.3

presentano un valore inferiore all'unità. L'input delle funzioni hash è la stringa "le ultime lettere di iacopo ortis" ma sono state eseguiti analoghi test anche con file di dimensione elevata.

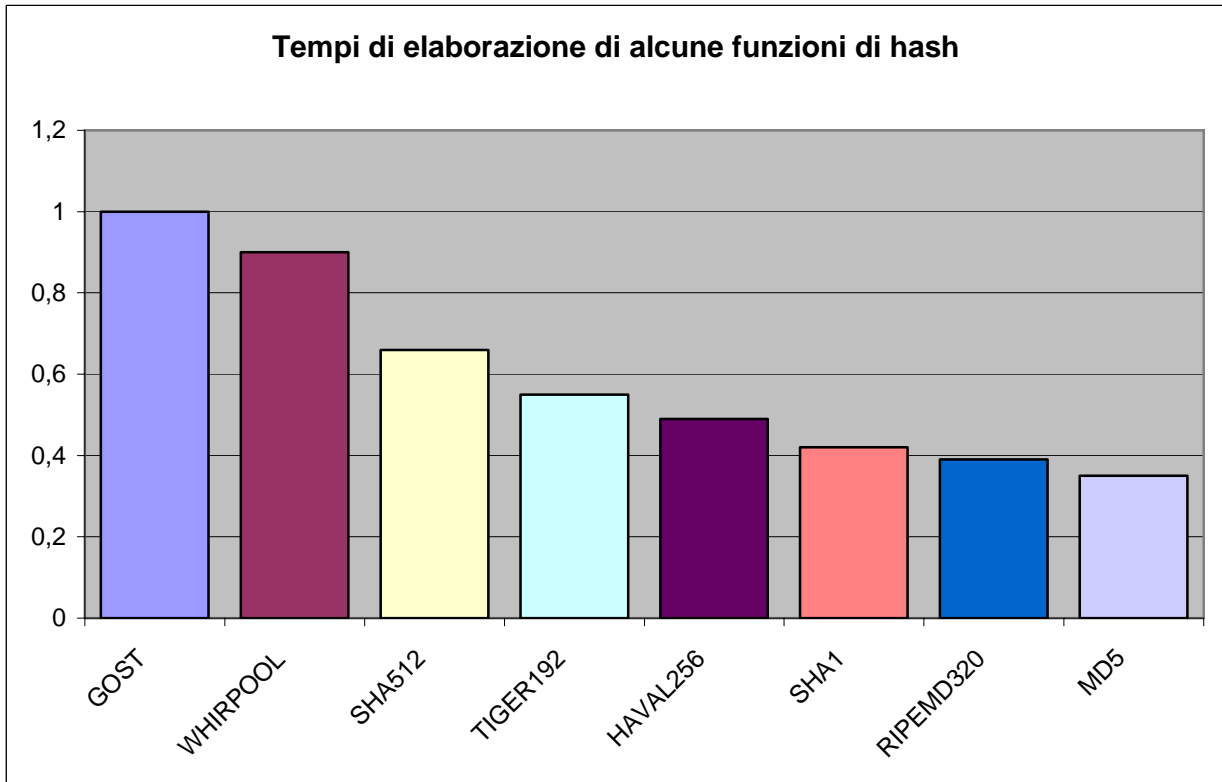
I risultati ottenuti mediante la replicazione degli algoritmi mostrano che l'algoritmo mediamente più lento è GOST, seguito da WHIRPOOL (uno dei più sicuri), la versione a 512 bit dello SHA per poi arrivare al più veloce, l'MD5. I risultati quindi non sorprendono: gli algoritmi più sicuri sono anche i più lenti mentre il più veloce è il più insicuro tra quelli considerati.

Algoritmo di hash	Tempo medio di esecuzione (sec)	Deviazione standard	Valore relativo (*)
GOST	2,49E-05	4,01E-06	1
WHIRPOOL	2,24E-05	7,19E-06	0.90
SHA512	1,64E-05	4,81E-06	0.66
TIGER192	1,39E-05	6,91E-06	0.55
HAVAL256	1,12E-05	3,03E-06	0.49
SHA1	1,04E-05	5,61E-06	0.42
RIPEMD320	9,81E-06	1,25E-05	0.39
MD5	8,86E-06	9,01E-06	0.35

(\*) = il valore più alto del tempo medio è stato posto pari a uno

Come detto in precedenza la scelta dell'algoritmo dipende da diversi fattori: se l'applicazione è crittografica o meno, il livello di sicurezza richiesto, necessità di performance elevate, l'eventuale combinazione con altri algoritmi e protocolli, etc.

Di certo possiamo individuare in MD5 un ottimo algoritmo per gli scopi non crittografici (è velocissimo), mentre per quelli crittografici RIPEMD (nelle versioni 160 e 320 bit, che offrono performance molto simili) è certamente un candidato da considerare con attenzione: produce message digest di 320 bit rapidamente, è frutto di un progetto europeo, non è coperto da brevetti, ed è (al momento) resistente alle collisioni.



Istogramma dei valori relativi

Questo testo sarà pubblicato sul [ripiano Informatica](#) di WikiBooks: sarà quindi possibile correggerlo, modificarlo ed integrarlo direttamente online. Sul blog [SegnalazioniIT](#) comunicheremo la data e le modalità di pubblicazione.

### **Approfondimenti online**

Hash in pratica: servizio web per calcolare valori di hash stringhe o da file  
<http://www.hashemall.com/>

#### *APPROFONDIMENTI MATEMATICI*

Introduction to Algorithms (MIT) sull'hashing  
<http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/index.htm>

Corso di Sicurezza A.A. 2006-2007 Paolo BIONDI - Funzioni hash crittografiche e paradosso del compleanno  
<http://www.cs.unibo.it/~margara/page2/page6/page25/assets/biondihash.pdf>

Corso sulla sicurezza del 1999 a cura di Giuseppe Abbagnale, Giovanni D'Agostino e Alfredo De Santis (gli approfondimenti matematici sono tuttora validi)

<http://www.dia.unisa.it/~ads/corso-security/www/CORSO-9900/hash/>

*MD5 e SHA-1*

Eugenio Rustico è autore di hashemall.com e della proposta di hashing parziale *md5fract: partial md5 checksum proposal*

<http://binaryunit.blogspot.com/2007/06/md5fract-partial-md5-checksum-proposal.html>

Bruce Schneier sulla crittoanalisi di SHA-1

[http://www.schneier.com/blog/archives/2005/02/cryptanalysis\\_o.html](http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html)

Sull'algoritmo MD5: Google come strumento per crackare le password

<http://www.lightbluetouchpaper.org/2007/11/16/google-as-a-password-cracker/>

Software per generare collisioni per MD5 e SHA-1

<http://www.win.tue.nl/hashclash/>

*TABELLE RAINBOW*

Sul funzionamento delle Rainbow Tables

<http://lasecwww.epfl.ch/pub/lasec/doc/Oech03.pdf>

Il programma Ophcrack e alcune tabelle rainbow

<http://ophcrack.sf.net>

Progetto e documentazione RainbowCrack

<http://www.antsight.com/zsl/rainbowcrack/>

Trovare una password in 140 secondi con le tabelle rainbow

<http://www.shannon.it/blog/trovare-una-password-in-140-secondi-con-le-rainbow-tables/>

*PHP*

HASH Message Digest Framework

<http://it.php.net/hash>

PHP 5: guida completa

<http://books.google.it/books?id=3RQOMzB5DboC>

Password Hashing di James McGlinn

<http://phpsec.org/articles/2005/password-hashing.html>

## **Testo della GNU Free Documentation License**

GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## **0. PREAMBLE**

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

### ***1. APPLICABILITY AND DEFINITIONS***

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely

XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## **2. VERBATIM COPYING**

---

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## **3. COPYING IN QUANTITY**

---

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of

that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

#### **4. MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## ***5. COMBINING DOCUMENTS***

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

## ***6. COLLECTIONS OF DOCUMENTS***

---

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## ***7. AGGREGATION WITH INDEPENDENT WORKS***

---

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## ***8. TRANSLATION***

---

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## ***9. TERMINATION***

---

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 10. FUTURE REVISIONS OF THIS LICENSE

---

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

### *How to use this License for your documents*

---

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document

under the terms of the GNU Free Documentation License, Version 1.2

or any later version published by the Free Software Foundation;

with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU

Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the

Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.