Tecniche di Progettazione Automatica **Anno Acc. 1998-99**
PROF. ALFREDO DE SANTIS

FUNZIONI HASH

a cura di G. Napolitano, F. Portolano, R. Repoli, A. Roca *12 Marzo 1999*

Indice

1 Introduzione

Le funzioni hash svolgono un ruolo di notevole importanza nella realizzazione di sistemi atti a garantire la sicurezza di dati rappresentati in formato digitale. Particolare rilevanza rivestono le applicazioni di dette funzioni nei campi delle *firme digitali* e della *certificazione del tempo*. Concettualmente una funzione hash può essere pensata come una funzione di compressione $h : X \rightarrow Z$ che ha dominio in un insieme di messaggi binari di lunghezza qualsiasi e codominio in un insieme di stringhe binarie di lunghezza finita (solitamente 128 o 160 bit).

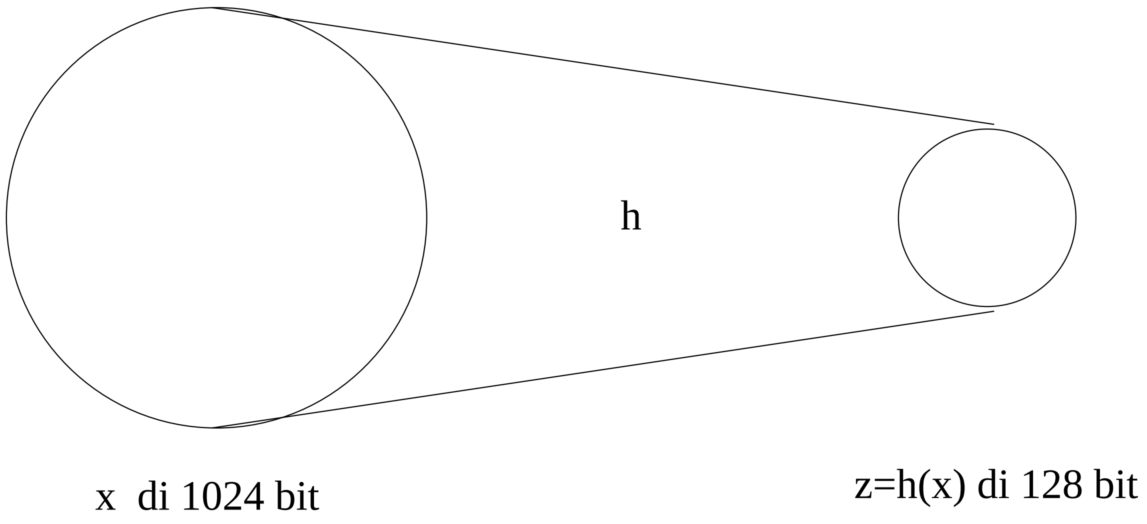
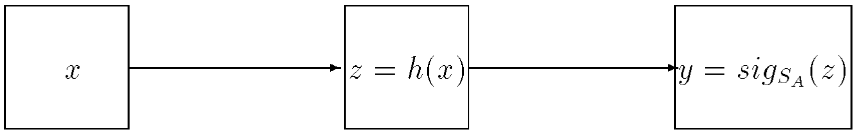


Figura 1: funzione hash

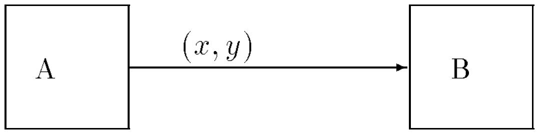
Il valore $z = h(x)$ è un'*impronta digitale* del messaggio di partenza x.
Per capirne l'utilizzo basta ripensare al problema dei messaggi lunghi emerso nella valutazione degli schemi di firme digitali proposti nelle pagine precedenti. Tutti i sistemi presentati, infatti, sono in grado di firmare messaggi di lunghezza fissata: nell' RSA questo limite è di 512 bit di messaggio per una firma di 512 bit, nel DSS è di 160 bit di messaggio per 320 bit di firma e cosí via. Una soluzione banale al problema consiste nel dividere il messaggio di partenza in blocchi e di firmare ogni blocco indipendentemente l'uno dall'altro. Gli svantaggi connessi a questo approccio sono principalmente:

- **Spazio:** l'elevata dimensione della firma che cresce proporzionalmente al numero di blocchi del messaggio.
- **Tempo:** la lentezza della verifica della firma dato che questa operazione comporta l'iterazione dell'algoritmo di verifica su tutti i blocchi del messaggio e, di conseguenza, il calcolo ripetuto di operazioni complesse quali l'esponenziazione modulare.
- **Sicurezza:** la possibilitá che un nemico in ascolto capti il messaggio, selezioni ad hoc i vari blocchi e li ricomponga in un nuovo messaggio, automaticamente firmato, di significato diverso da quello originario.

Lasciare la possibilitá di realizzare falsificazioni di questo tipo è ovviamente inaccettabile. Si supponga, per esempio, che i dati trasmessi rappresentino transazioni finanziarie o atti pubblici in genere: le conseguenze di un attacco riuscito potrebbero essere disastrose. Una soluzione efficiente a tutti questi problemi consiste nell'uso di una funzione hash pubblica molto veloce che prende in input il messaggio originario di lunghezza qualsiasi x e produce in output un messaggio succinto z *(message digest)* di lunghezza fissata. La strategia che potrebbe essere seguita dall'utente A che trasmette potrebbe essere quella di calcolare prima $z = h(x)$ e, successivamente, di firmare z al posto di x costruendo cioé $y = sig_{S_A}(z)$ tramite la sua chiave privata S_A.



Infine potrebbe trasmettere all'utente B la coppia (x, y) sul canale.



Una volta che il messaggio firmato è stato trasmesso sia il destinatario B che chiunque altro risultasse interessato alla verifica, potrebbe calcolare che y è effettivamente la firma di A su x, cioé

$$ver_{P_A}(z, y) = vero.$$

Infatti, nell'ipotesi che h è pubblica, tutti possono calcolare il valore $z = h(x)$ e quindi verificare che $y = sig_{S_A}(z)$ è proprio la firma di A per il messaggio x effettuata su z.

2 Il problema della sicurezza

La scelta di una funzione hash, che permetta di realizzare la strategia di firma di documenti appena descritta, è un'operazione critica. Occorre, infatti, assicurarsi che la funzione individuata non indebolisca la sicurezza dello schema di firma, fenomeno che si verifica quando la funzione produce facilmente collisioni.

Definizione 1 *Dati due valori x e x' ed una funzione h, diciamo che la coppia (x, x') produce una* **collisione** *per h se*

$$x \neq x' \qquad e \qquad h(x) = h(x').$$

E' facile intuire che se il nemico puó trovare agevolmente piú messaggi x, tali che $h(x) = z$ per un certo z di cui conosce la firma $y = sig_k(z)$, allora puó facilmente costruire contraffazioni. Pertanto occorre procedere con cura.

2.1 Funzioni hash libere da collisioni

Da un'attenta analisi degli attacchi che possono essere portati alla strategia di firma presentata è possibile individuare quelle proprietá che h deve soddisfare onde evitare possibili contraffazioni.

Attacco 1- Il nemico riesce a conoscere il messaggio (x, y) inviato dall'utente A dove $y = sig_k(h(x))$. Allora calcola $z = h(x)$ e cerca di trovare un $x' \neq x$ tale che $h(x) = h(x')$; se ci riesce puó costruire la coppia (x', y) che è una contraffazione.

Per prevenire questo tipo di attacco h deve soddisfare la seguente proprietà di sicurezza:

Definizione 2 (Sicurezza debole) *Una funzione hash h è debolmente priva di collisioni se dato un messaggio x è computazionalmente inammissibile trovare un messaggio x' tale che*

$$x' \neq x \qquad e \qquad h(x') = h(x).$$

Attacco 2- Il nemico cerca due messaggi $x' \neq x$ tali che $h(x') = h(x)$ e persuade l'utente a firmare $h(x)$ ottenendo $y = sig_k(h(x))$. In tal caso la coppia (x', y) è una contraffazione. Per ovviare a ció h deve soddisfare una seconda proprietá di sicurezza:

Definizione 3 (Sicurezza forte) *Una funzione h è fortemente libera da collisioni se è computazionalmente inammissibile trovare due messaggi x e x' tali che*

$$x \neq x' \qquad e \qquad h(x) = h(x').$$

Attacco 3- Il nemico conosce la firma di un messaggio ridotto z e cerca di trovare un messaggio x tale che $h(x) = z$. Se ci riesce la coppia (x, y) è una falsificazione.

Per prevenire quest'attacco h deve essere una funzione one-way, proprietá giá introdotta nello studio dei crittosistemi a chiave pubblica.

Definizione 4 (One-way) *h è una funzione hash one-way se, dato un message digest z, è computazionalmente inammissibile trovare un messaggio x tale che $h(x) = z$.*

E' possibile dimostrare che la proprietá di sicurezza forte implica sia la proprietá di sicurezza debole che la proprietá di one-way. La verifica della prima implicazione è immediata: infatti se h è fortemente sicura allora, per definizione, è computazionalmente inammissibile trovare, tra tutte le coppie di messaggi, una che collide. A maggior ragione è computazionalmente inammissibile trovare una tale coppia con uno dei messaggi fissato. D'altra parte, se h fosse fortemente sicura ma non debolmente sicura, dovrebbe essere computazionalmente possibile, dato un messaggio x, trovare un x' per cui c'è collisione. Ma ciò significherebbe che è computazionalmente ammissibile trovare una coppia che collide contrariamente a quanto stabilito dalla proprietà di sicurezza forte. Più complesso risulta, invece, dimostrare la seconda implicazione e cioè che se h è fortemente sicura allora h è una funzione one-way. Otterremo il risultato facendo vedere che qualora h fosse fortemente sicura allora non potrebbe non essere one-way. Un'ipotesi diversa, e cioè l'invertibilità di h in tempo ammissibile tramite un algoritmo A porta alla possibilità di costruire un algoritmo probabilistico di tipo Las Vegas che, servendosi di A come di un oracolo, trova collisioni *con alta probabilità in tempo ammissibile*, contrariamente a quanto stabilito dalla proprietà di sicurezza forte di h.

Questo risultato può essere ottenuto con un'assunzione molto debole sulle dimensioni relative del dominio e del codominio della funzione hash.

Supponiamo che $h : X \to Z$ sia la nostra funzione dove X e Z sono insiemi finiti e $|X| \geq 2|Z|$. Quest'ipotesi è ragionevole in quanto se pensiamo ad un elemento di X come ad una stringa di bit di lunghezza $\log_2 |X|$ e a quello di Z come ad una di lunghezza $\log_2 |Z|$ allora stiamo supponendo che la lunghezza di $z = h(x)$ sia più piccola di almeno un bit rispetto a quella di x (infatti $\log_2 |X| \geq \log_2(2|Z|) = \log_2 2 + \log_2 |Z| = 1 + \log_2 |Z|$).
È ovvio che diminuendo il numero di bit il numero di collisioni non puó che aumentare.

Vale il seguente :

Teorema 2.1 *Sia $h : X \to Z$ una funzione hash, dove $|X|$ e $|Z|$ sono finiti e $|X| \geq 2|Z|$. Sia A un algoritmo di inversione per h. Allora esiste un algoritmo probabilistico di tipo Las Vegas che trova una collisione per h con probabilità almeno $\frac{1}{2}$.*

Dimostrazione. Consideriamo il seguente algoritmo

1. scegli un $x \in X$ a caso
2. calcola $z = h(x)$
3. calcola $x' = A(z)$
4. **se** $x' \neq x$ **allora** (x', x) genera una collisione (successo)
   altrimenti ESCI (fallimento)

L'algoritmo è chiaramente di tipo Las Vegas in quanto o dá una risposta esatta o non risponde.

Per calcolare la probabilità di successo dell'algoritmo nella ricerca di collisioni operiamo come segue: definiamo una relazione di equivalenza $\sim$ su X ponendo

$$x \sim x' \Leftrightarrow h(x) = h(x')$$

e denotiamo con

$$[x] = \{x' \in X \text{ tali che } x \sim x'\}$$

la classe di equivalenza dell'elemento $x \in X$.
Indichiamo con $\mathbf{C}$ l'insieme delle classi di equivalenza. Risulta che $|\mathbf{C}| \leq |Z|$ in quanto ogni classe $[x]$ è costituita dall'immagine inversa di al più un elemento di Z.

Sia $x \in X$ l'elemento scelto nel passo 1 dell'algoritmo. Per questo x ci sono $|[x]|$ possibili valori che A puó restituire come preimmagini di z al passo 3 dell'algoritmo. Di questi solo $|[x]| - 1$ sono differenti da x e portano ad un successo nel passo 4. Risulta dunque che

$$P_r(successo \mid x) = \frac{|[x]| - 1}{|[x]|}$$

e quindi

$$\begin{aligned}
P_r(successo) &= \sum_{x \in X} P_r(successo, x) \\
&= \sum_{x \in X} p_r(x) P_r(successo \mid x) \\
&= \sum_{x \in X} \frac{1}{|X|} \frac{(|[x]| - 1)}{|[x]|} \\
&= \frac{1}{|X|} \sum_{c \in \mathbf{C}} \sum_{x \in c} \frac{(|c| - 1)}{|c|} && \text{dato che } [x] \in \mathbf{C} \\
&= \frac{1}{|X|} \sum_{c \in \mathbf{C}} \frac{|c| - 1}{|c|} |c| && \text{dato che } \sum_{x \in c} 1 = |c| \\
&= \frac{1}{|X|} \sum_{c \in \mathbf{C}} |c| - 1 \\
&= \frac{1}{|X|} \left(\sum_{c \in \mathbf{C}} |c| - \sum_{c \in \mathbf{C}} 1 \right) \\
&\geq \frac{1}{|X|} (|X| - |Z|) && \text{dato che } \sum_{c \in \mathbf{C}} |c| = |X| \; e \; \sum_{c \in \mathbf{C}} 1 = |\mathbf{C}| \leq |Z| \\
&\geq \frac{1}{|X|} (|X| - \frac{|X|}{2}) \\
&\geq \frac{1}{2}
\end{aligned}$$

$\square$

Pertanto la sicurezza forte implica necessariamente la proprietá di one-way.

2.2 Attacco del compleanno

Un altro parametro importante che occorre considerare con attenzione nell'utilizzazione di una funzione hash è la cardinalitá dell'insieme Z. E' ovvio che se la cardinalitá è troppo piccola è facile trovare collisioni. Attraverso il cosiddetto attacco del compleanno è possibile valutare rigorosamente l'incidenza di questo parametro sulla sicurezza della funzione e individuare valori opportuni per funzioni di rilevanza pratica.

La denominazione di tale attacco deriva dal paradosso del compleanno, ben noto in teoria delle probabilità, in base al quale in un gruppo di 23 individui, scelti a caso, almeno due hanno lo stesso compleanno con probabilità di almeno $\frac{1}{2}$. Come prima supponiamo che $h : X \to Z$ sia la nostra funzione hash, che $|X| = m$ e $|Z| = n$, con n ed m finiti e che $|X| \geq 2|Z|$. Banalmente il numero di collisioni è almeno n.

Un approccio ingenuo alla ricerca di collisioni consiste nello scegliere k valori casuali distinti, $x_1, \cdots, x_k$ appartenenti a X , calcolare $z_i = h(x_i)$ per $1 \leq i \leq k$ e verificare se c'è una collisione.

Usando questo metodo possiamo calcolare un limite inferiore alla probabilità di trovare collisioni. Questo limite inferiore dipenderà da k, da n ma non da m. Dal momento che siamo interessati al limite inferiore supponiamo che $|h^{-1}(z)| \simeq \frac{m}{n}$ per ogni $z \in Z$, ossia, che la dimensione delle immagini inverse di h siano alquanto bilanciate. Sotto questa ipotesi infatti la probabilità che ci siano collisioni è più bassa rispetto al caso in cui le immagini inverse non siano bilanciate, come mostrato nel seguente

Esempio 1 *Sia m=4 e n=2. Se abbiamo una situazione del genere*

$$\begin{tikzcd}
x_1 \arrow[dr] & & & x_3 \arrow[dr] & \\
& y_1 & & & y_2 \\
x_2 \arrow[ur] & & & x_4 \arrow[ur] &
\end{tikzcd}$$

allora risulta che

$$P_r(stesso\, valore\, hash) = \frac{casi\, favorevoli}{casi\, possibili} =$$

4

2

Se invece la situazione è la seguente

$$\begin{tikzcd}
x_1 \arrow[dr] & & & \\
x_2 \arrow[r] & y_1 & x_4 \arrow[r] & y_2 \\
x_3 \arrow[ur] & & &
\end{tikzcd}$$

allora

$$P_r(stesso\, valore\, hash) =$$

4

2

Dato che le immagini inverse sono tutte circa della stessa dimensione e gli x_i sono scelti a caso, anche gli z_i possono essere considerati elementi casuali di Z, non necessariamente distinti.

È semplice calcolare la probabilitá che k elementi $z_1, \cdots, z_k$ siano distinti. La scelta di z_1 è arbitraria. Per z_2 si ha

$$p(z_2 \neq z_1) = 1 - p(z_2 = z_1) = 1 - \frac{1}{n}.$$

Per z_3 si ha

$$p(z_3 \neq z_2\, e\, z_3 \neq z_1 \mid z_1 \neq z_2) = 1 - p(z_3 = z_2\, o\, z_3 = z_1 \mid z_1 \neq z_2) = 1 - \frac{2}{n},$$

e cosí via. Dunque

$$\begin{aligned}
P_r(non\, avere\, collisioni) &= P(z_2 \neq z_1)\cdot P(z_3 \neq z_1, z_3 \neq z_2 \mid z_1 \neq z_2)\dots \\
&\quad \dots P(z_k \neq z_1, \dots, z_k \neq z_{k-1} \mid z_1 \neq z_2 \neq \dots \neq z_{k-1}) \\
&= \left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\dots\left(1-\frac{k-1}{n}\right) \\
&= \prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)
\end{aligned}$$

Se x è un reale molto piccolo, dallo sviluppo in serie dell'esponenziale, troncato al secondo termine , si ottiene $e^{-x} \approx 1-x$; per cui si ha ancora che

$$\begin{aligned}
P_r(non\, avere\, collisioni) &\approx \prod_{i=1}^{k-1} e^{-\frac{i}{n}} \\
&= e^{-\sum_{i=1}^{k-1}\frac{i}{n}} \\
&= e^{-\frac{1}{n}\sum_{i=1}^{k-1} i} \\
&= e^{-\frac{k(k-1)}{2n}}
\end{aligned}$$

e di conseguenza che

$$P_r(avere\, almeno\, una\, collisione) \approx 1 - e^{-\frac{k(k-1)}{2n}}.$$

Denotando con $\varepsilon = P_r(avere\, almeno\, una\, collisione)$ si ha che:

$$\begin{aligned}
\varepsilon &\approx 1 - e^{-\frac{k(k-1)}{2n}} \\
e^{-\frac{k(k-1)}{2n}} &\approx 1-\varepsilon \\
-\frac{k(k-1)}{2n} &\approx \ln(1-\varepsilon) \\
k^2 - k &\approx 2n \ln\frac{1}{1-\varepsilon}
\end{aligned}$$

e se ignoriamo il termine $-k$ si ha:

$$k \approx \sqrt{2n \ln\frac{1}{1-\varepsilon}}.$$

Nel paradosso del compleanno scelti $\varepsilon = 0.5$ e $n = 365$ si ha che $k \approx 22.3$ da cui la relativa formulazione. Per ogni fissato ε, k varia rimanendo comunque proporzionale a $\sqrt{n}$ con un fattore di proporzionalità pari a $\sqrt{2\ln\frac{1}{1-\varepsilon}}$. Quest'attacco impone un limite inferiore sulla dimensione di Z, al fine di avere una bassa probabilità di trovare collisioni, come mostrato nel seguente

Esempio 2 *Se $n = |Z| = 2^{40}$ e $\varepsilon = 0.5$ si ha che $k \approx 1.17\sqrt{n} \approx 1.17 \cdot 2^{20}$ e quindi può aversi una collisione con probabilità $\frac{1}{2}$ scegliendo $2^{20} \approx 1.000.000$ valori hash casuali, un numero non proibitivo da calcolare. Un messaggio di 40 bit è dunque molto insicuro.*

Dunque, per la sicurezza, è richiesto un numero di bit pari almeno a 128.

3 Funzioni hash su dominio finito

3.1 Funzione hash su logaritmo discreto

Chaum, Von Heijst e Pfitzmann [3] hanno proposto una funzione hash la cui sicurezza si basa sulla difficoltá di calcolo del logaritmo discreto. Tale funzione non è abbastanza veloce da poter essere usata in pratica ma è semplice da capire e fornisce un esempio di come possano essere realizzate funzioni hash scegliendo come riferimento ben noti problemi teorici le cui soluzioni sono ritenute computazionalmente inammissibili. Questo approccio ha il vantaggio di rendere possibile una qualche forma di quantificazione della sicurezza della funzione costruita.

Sia p un primo molto grande e $q = (p-1)/2$ anch'esso primo. Siano α e β due elementi primitivi di $\mathbf{Z}_p^*$. Il valore $\log_\alpha \beta$ non è pubblico e assumiamo sia computazionalmente inammissibile calcolarlo. La funzione hash

$$h : \{0, \ldots, q-1\} \times \{0, \ldots, q-1\} \rightarrow \mathbf{Z}_p - \{0\} = \mathbf{Z}_p^*$$

è definita come segue

$$h(x_1, x_2) = \alpha^{x_1}\beta^{x_2} \bmod p.$$

Dimostreremo ora come la conoscenza di una collisione per h porta ad un metodo di computazione efficiente per $\log_\alpha \beta$.

Sia $((x_1, x_2), (x_3, x_4))$ una collisione per h. Allora

$$(x_1, x_2) \neq (x_3, x_4) \tag{1}$$

e

$$\alpha^{x_1}\beta^{x_2} \equiv \alpha^{x_3}\beta^{x_4} \bmod p. \tag{2}$$

Sia $d = \gcd(x_4 - x_2, p-1)$. Dato che $p - 1 = 2q$ e q è primo, d può assumere un valore tra $\{1, 2, q, p-1\}$ e non altri. Analizziamo i vari casi

Caso 1: $d = 1$. Sia $y = (x_4 - x_2)^{-1} \bmod (p-1)$ l'inverso moltiplicativo di $(x_4 - x_2)$ in $\mathbf{Z}_{p-1}^*$ (che esiste sempre quando $\gcd(x_4 - x_2, p-1) = 1$). Abbiamo che

$$\begin{aligned} \beta &\equiv \beta^{(x_4 - x_2)y} \bmod p \\ &\equiv \alpha^{(x_1 - x_3)y} \bmod p \qquad \textit{per la (2)} \end{aligned}$$

quindi

$$\begin{aligned} \log_\alpha \beta &= (x_1 - x_3)y \bmod (p-1) \\ &= (x_1 - x_3)(x_4 - x_2)^{-1} \bmod (p-1). \end{aligned}$$

Abbiamo quindi trovato un modo semplice per calcolare il logaritmo discreto!

Caso 2: $d = 2$. Come conseguenza abbiamo che $\gcd(x_4 - x_2, q) = 1$ perchè q è necessariamente dispari.

Analogamente al caso precedente sia $y = (x_4 - x_2)^{-1} \bmod q$. Necessariamente esisterà un intero k tale che $(x_4 - x_2)y = kq + 1$ per cui avremo

$$\begin{aligned} \beta^{(x_4-x_2)y} &\equiv \beta^{kq+1} \bmod p \\ &\equiv (\beta^q)^k \beta \bmod p \\ &\equiv (-1)^k \beta \bmod p \\ &\equiv \pm\beta \bmod p \end{aligned} \tag{3}$$

dal momento che $\beta^q \equiv (-1) \bmod p$. Risulta infatti che $\beta^q = (\beta^{p-1})^{\frac{1}{2}} = (1)^{\frac{1}{2}}$ ed 1 ha due radici: 1 e -1. Essendo β elemento primitivo di $\mathbf{Z}_p^*$ se fosse $\beta^q \equiv 1 \bmod p$ allora β non potrebbe generare tutto $\mathbf{Z}_p^*$. In definitiva da

$$\alpha^{(x_1-x_3)} \equiv \beta^{x_4-x_2} \bmod p$$

moltiplicando gli esponenti per y e tenendo in conto il risultato (3) abbiamo

$$\alpha^{(x_1-x_3)y} \equiv \pm\beta \bmod p$$

ovvero

$$\alpha^{(x_1-x_3)y} \equiv \beta \bmod p$$

oppure

$$\alpha^{(x_1-x_3)y} \equiv -\beta \bmod p$$

da cui si evince, rispettivamente:

$$\begin{aligned} \log_\alpha \beta &= (x_1 - x_3)y \bmod (p-1) \\ \log_\alpha \beta &= (x_1 - x_3)y + q \bmod (p-1) \end{aligned}$$

Per quest'ultima basta ricordare che essendo α elemento primitivo di $\mathbf{Z}_p^*$ gode delle stesse proprietà di β, quindi moltiplicando entrambi i membri dell'equazione per α^q otteniamo:

$$\begin{aligned} \alpha^q \cdot \alpha^{(x_1-x_3)y} &\equiv (-1) \cdot (-\beta) \bmod p \\ \log_\alpha(\alpha^q \cdot \alpha^{(x_1-x_3)y}) &\equiv \log_\alpha \beta \bmod p \\ \log_\alpha \beta &= q + (x_1 - x_3)y \bmod p - 1. \end{aligned}$$

Anche in questo caso possiamo calcolare facilmente il $\log_\alpha \beta$.

Caso 3: $d = q$. Questo caso è impossibile in quanto

$$\forall i \; x_i \in \{0, 1, \ldots, q-1\} \qquad \Rightarrow \qquad -(q-1) \leq x_4 - x_2 \leq q - 1$$

ma $q > q - 1$ e dunque q non può dividere $x_4 - x_2$.

Caso 4: $d = p - 1$. Questo accade solo se $x_4 = x_2$. In tal caso risulta che:

$$\begin{aligned} \alpha^{x_1}\beta^{x_2} &\equiv \alpha^{x_3}\beta^{x_4} \bmod p \\ \alpha^{x_1}\beta^{x_2} &\equiv \alpha^{x_3}\beta^{x_2} \bmod p \\ \alpha^{x_1} &\equiv \alpha^{x_3} \bmod p \end{aligned}$$

il che è vero solo se $x_1 = x_3$. In definitiva si ha che $(x_1, x_2) = (x_3, x_4)$ il che è contro l'ipotesi (1) e pertanto $d = p - 1$, è un caso impossibile.

Pertanto possiamo ritenere la funzione h strettamente priva di collisioni. Infatti, da quanto visto, se fosse possibile trovare una collisione allora sarebbe possibile calcolare facilmente il logaritmo discreto contrariamente all'ipotesi che tale problema risulta computazionalmente inammissibile.

4 Estensione di funzioni hash

Finora abbiamo considerato funzioni hash fortemente libere da collisioni definite su un dominio finito; tuttavia è sempre possibile costruire un'estensione su un dominio infinito lasciando invariata la proprietá [4]. Ció permette di firmare messaggi di lunghezza arbitraria.

Sia $h : \mathbf{Z}_2^m \to \mathbf{Z}_2^t$ con $m \geq t + 1$, la nostra funzione strettamente libera da collisioni e $h^\star : X \to \mathbf{Z}_2^t$ la sua estensione dove $X = \bigcup_{i=m}^{\infty} \mathbf{Z}_2^i$.

Consideriamo prima il caso in cui $m \geq t + 2$.

Per ogni $x, y \in X$ indicheremo con $|x| = n > m$ il numero di bit della stringa x e con $x\|y$ la concatenazione delle due stringhe.

Possiamo esprimere x come $x_1\|x_2\|\cdots\|x_k$ dove

$$|x_i| = m - t - 1 \qquad \text{se } i \in \{1, \ldots, k-1\}$$

$$\text{e } |x_k| = m - t - 1 - d \qquad \text{con } 0 \leq d \leq m - t - 2.$$

Quindi

$$k = \left\lceil \frac{n}{m - t - 1} \right\rceil$$

In effetti abbiamo suddiviso x in k blocchi di cui i primi $k - 1$ di $m - t - 1$ bit e l'ultimo con d bit in meno, dove d è un numero variabile nel range sopra indicato. Vediamo come definire $h^\star(x)$ mediante il seguente algoritmo:

1. **for** $i = 1$ **to** $k - 1$ **do**
   $y_i = x_i$
2. $y_k = x_k\|0^d$
3. sia y_{k+1} la rappresentazione binaria di d
4. $g_1 = h(0^{t+1}\|y_1)$
5. **for** $i = 1$ **to** k **do**
   $g_{i+1} = h(g_i\|1\|y_{i+1})$
6. $h^\star(x) = g_{k+1}$

Poniamo

$$y(x) = y_1\|y_2\|\cdots\|y_{k+1}.$$

Quello che facciamo è innanzitutto riscrivere il messaggio in modo tale che ogni blocco abbia $m - t - 1$ bit. A tale scopo, aggiungiamo d zeri a x_k per ottenere y_k della lunghezza voluta.

Inoltre aggiungiamo un'ulteriore sequenza di bit, y_{k+1}. La funzione $x \mapsto y(x)$ cosí costruita risulta iniettiva. Se cosí non fosse, $h^\star$ non potrebbe essere fortemente libera da collisioni in quanto se per $x \neq x'$ la funzione non risultasse iniettiva, si avrebbe $y(x) = y(x')$ e dunque $h(y(x)) = h(y(x'))$. Di conseguenza $h^\star(x) = h^\star(x')$.

Proviamo che $h^\star$ è fortemente libera da collisioni dimostrando che, qualora non lo fosse, potremmo trovare una collisione per h in tempo polinomiale. Nell'ipotesi che h sia fortemente libera da collisioni questo porta ad un assurdo e quindi, resta provato che $h^\star$ è strettamente libera da collisioni.

Siano x e x' i due messaggi tali che $x \neq x'$ e $h^\star(x) = h^\star(x')$.

Indichiamo con

$$y(x) = y_1 \| y_2 \| \cdots \| y_{k+1} \qquad y(x') = y'_1 \| y'_2 \| \cdots \| y'_{l+1}$$

la rielaborazione di x e x' (ovviamente $y(x) \neq y(x')$), con d e d' il numero di zeri aggiunti a x e x' rispettivamente, e con $g_1 \cdots g_{k+1}$ e $g'_1 \cdots g'_{l+1}$ i valori calcolati nei passi 4 e 5 dell'algoritmo precedente.

Possiamo individuare due casi a seconda del fatto che sia $|x| \equiv |x'| \bmod (m-t-1)$ o meno.

caso 1: $|x| \not\equiv |x'| \bmod (m-t-1)$.

In questo caso $d \neq d'$ e $y_{k+1} \neq y_{l+1}$. Abbiamo quindi

$$\begin{aligned} h(g_k \| 1 \| y_{k+1}) &= g_{k+1} \\ &= h^\star(x) \\ &= h^\star(x') \\ &= g'_{k+1} \\ &= h(g'_k \| 1 \| y'_{l+1}) \end{aligned}$$

ma $g_k \| 1 \| y_{k+1} \neq\ g'_k \| 1 \| y_{l+1}$ per l'ipotesi fatta su y_{k+1} e y'_{l+1},quindi si ha una collisione per h!

caso 2: $|x| \equiv |x'| \bmod m-t-1$.

Suddividiamo l'analisi in due sottocasi:

caso 2a: $|x| = |x'|$.

Conseguenza della condizione è che $k = l$ e quindi $y_{k+1} = y'_{k+1}$.

Procedendo come nel caso 1 si ha che $h(g_k \| 1 \| y_{k+1}) = h(g'_k \| 1 \| y'_{k+1})$ e se $g_k \neq g'_k$ abbiamo trovato una collisione. Supponendo invece che $g_k = g'_k$ avremo

$$\begin{aligned} h(g_{k-1} \| 1 \| y_k) &= g_k \\ &= g'_k \\ &= h(g'_{k-1} \| 1 \| y'_k) \end{aligned}$$

A questo punto possiamo o trovare una collisione per h o avere ancora $g_{k-1} = g'_{k-1}$ e $y_k = y'_k$ Supponendo di non trovare mai delle collisioni, possiamo ripetere il lavoro all'indietro fino ad arrivare alla situazione in cui

$$\begin{aligned} h(0^{t+1} \| y_1) &= g_1 \\ &= g'_1 \\ &= h(0^{t+1} \| y'_1) \end{aligned}$$

Se $y_1 = y'_1$ (altrimenti avremmo trovato una collisione), allora per ogni $1 \leq i \leq k+1$ risulta $y_i = y'_i$ e dunque $y(x) = y(x')$; ma questo implica che $x = x'$, dal momento che la funzione $x \mapsto y(x)$ è un'iniezione, il che implica un assurdo avendo supposto $x \neq x'$.

caso 2b: $l = |x'| > |x| = k$.
Si ha lo stesso procedimento del caso 2a.
Supponendo di non trovare mai una collisione arriveremmo al punto in cui

$$\begin{aligned} h(0^{t+1} \| y_1) &= g_1 \\ &= g_{l-k+1} \\ &= h(g'_{l-k} \| 1 \| y'_{l-k+1}) \end{aligned}$$

A questo punto troviamo una collisione per h in quanto $0^{t+1} \| y_1 \neq g'_{l-k} \| 1 \| y'_{l-k+1}$ come conseguenza del fatto che la stringa $0^{t+1} \| y_1$ ha il $(t+1)$-esimo bit pari a 0 mentre $g'_{l-k} \| 1 \| y'_{l-k+1}$ ha lo stesso bit pari a 1 (infatti essendo $g'_{l-k} = h : \mathbf{Z}_2^m \to \mathbf{Z}_2^t$ risulta che che $|g'| = t$).
La costruzione precedente vale solo per $m \geq t+2$.
Vediamo come costruire $h^\star$ per $m = t+1$.
Sia $|x| = n > m$. Codifichiamo x mediante una funzione f definita come segue:

$$f(0) = 0 \qquad f(1) = 01$$

L'algoritmo per la costruzione è il seguente

1. Sia $y = y_1 \cdots y_k = 11 \| f(x_1) \| \cdots \| f(x_n)$
2. $g_1 = h(0^t \| y_1)$
3. **for** $i = 1$ **to** $k - 1$ **do**
   $g_{i+1} = h(g_i \| y_{i+1})$
4. $h^\star(x) = g_k$

Osserviamo innanzitutto che la codifica $x \mapsto y(x)$ soddisfa due importanti proprietà:

1. se $x \neq x'$ allora $y(x) \neq y(x')$ ($x \mapsto y(x)$ è un'iniezione)
2. Non esistono due stringhe $x \neq x'$ ed una stringa z tale che $y(x) = z \| y(x')$. In altre parole, nessuna codifica è postfissa ad un'altra. Questo è semplice da vedere perché ogni stringa $y(x)$ inizia con 11 e non esistono due uni consecutivi nel resto della stringa.

Come prima supponiamo che possiamo trovare $x \neq x'$ tale che $h^\star(x) = h^\star(x')$ e sia $y(x) = y_1 \cdots y_k$ e $y(x') = y'_1 \cdots y'_l$.
Consideriamo due casi:

caso 1: $k = l$. Come nel caso 2a, o troviamo una collisione per h o troviamo che $y(x) = y(x')$. Ma quest'ultimo implica che $x = x'$ che è un assurdo.
caso 2: $l \neq k$.

Senza perdere di generalitá supponiamo che sia $l > k$ e che non ci siano collisioni per h. Abbiamo la seguente sequenza di uguaglianze:

$$\begin{aligned} y_k &= y'_l \\ y_{k-1} &= y'_{l-1} \\ &\vdots \\ y_1 &= y_{l-k+1} \end{aligned}$$

Ma allora y_1 è un postfisso di y'_l il che è impossibile.

Resta cosí dimostrato il seguente

Teorema 4.1 *Sia $h : \mathbf{Z}_2^m \to \mathbf{Z}_2^t$ una funzione hash fortemente libera da collisioni, definita su un dominio finito, con $m \geq t + 1$. Allora esiste una funzione hash, definita su un dominio infinito, fortemente libera da collisioni, $h^\star : \bigcup_{i=m}^{\infty} \mathbf{Z}_2^i \to \mathbf{Z}_2^t$.*

Il numero di volte che viene computata h per la valutazione di $h^\star$ è pari a $k + 1$ ed è quindi al piú pari a $\lceil \frac{n}{m-t-1} \rceil + 1$ se $m \geq t + 2$. Nel caso invece di $m = t + 1$ h viene valutata esattamente k volte dove k puó al piú valere $2n + 2$ (tanto grande è infatti la lunghezza di y quando $x = 1^n$).

5 Costruzione di funzioni hash da crittosistemi a chiave privata

I metodi finora discussi portano alla costruzione di funzioni hash che sono probabilmente lente per essere utili in pratica. Un altro approccio consiste nell'uso di un crittosistema a chiave privata per la costruzione di una funzione hash [8].

Sia $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ un crittosistema computazionalmente sicuro e supponiamo che

$\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbf{Z}_2^n$. Al fine di prevenire l'attacco del compleanno deve essere $n \geq 128$ il che implica un'esclusione a priori del DES. Data una stringa di bit

$$x = x_1 \| x_2 \| \cdots \| x_k$$

dove $x_i \in \mathbf{Z}_2^n, 1 \leq i \leq k$, e un certo Valore Iniziale, g_0=VI, costruiamo una successione di valori $g_1, g_2, \ldots, g_k$ in base alla seguente regola

$$g_i = f(x_i, g_{i-1})$$

dove f è una funzione che comprende al suo interno la funzione di cifratura del nostro crittosistema. Il messaggio finale sará $h(x) = g_k$.

Sono state proposte molte funzioni di questo tipo e di alcune è stata provata l'insicurezza (indipendentemente dal fatto che il crittosistema sottostante fosse sicuro o meno).

Quattro variazioni sul tema che sembrano essere sicure sono:

$$\begin{aligned} g_i &= e_{g_{i-1}}(x_i) \oplus x_i \\ g_i &= e_{g_{i-1}}(x_i) \oplus x_i \oplus g_{i-1} \\ g_i &= e_{g_{i-1}}(x_i \oplus g_{i-1}) \oplus x_i \\ g_i &= e_{g_{i-1}}(x_i \oplus g_{i-1}) \oplus x_i \oplus g_{i-1} \end{aligned}$$

6 Alcune proposte per funzioni hash

Presenteremo ora alcune funzioni hash che vengono utilizzate per la risoluzione di problemi pratici. Analizzeremo tre funzioni hash: l'MD4, l'MD5 e l'SHS. **MD4** è stata progettata nel 1990 da Ron Rivest. La sigla MD sta per *'message digest'* ovvero messaggio succinto; infatti l'MD4 prende come input un messaggio di lunghezza arbitraria e restituisce in output una stringa di 128 bit. MD4 é stata intensamente studiata. Bert den Boer e Antoon Bossalaers [2] crittanalizzarono con successo una parte dell'algoritmo mentre Ralph Merkle attaccó con successo i primi due round. Eli Biham discusse in seguito un possibile attacco basato su crittoanalisi differenziale contro due terzi di MD4. Infine il 18 Settembre 1995 è stata trovata una collisione per MD4.
Rivest, intravedendo le debolezze di MD4, nel 1991 modificó il vecchio algoritmo, che comunque venne ancora utilizzato, creando una nuova funzione hash piú sicura che prese il nome di **MD5**.
L' **SHS** (Secure Hash Standard) fu pubblicato sul Federal Register il 31 gennaio del 1991 e fu adottato come standard dallo Stato americano l'11 maggio del 1993. L'SHS è pi 'u lenta sia dell'MD4 che dell'MD5 ma si basa sugli stessi principi di progettazione di questi ultimi due algoritmi.

6.1 MD4

Gli obiettivi che Rivest [9] era proposto nel progetto di MD4 sono:
Sicurezza: la funzione doveva essere fortemente libera da collisioni;
Sicurezza diretta: la sicurezza di MD4 non si doveva basare su problemi teorici difficili computazionalmente.
Velocitá: MD4 doveva essere adatta per implementazioni software molto veloci;
Semplicitá e Compattezza: MD4 non doveva far uso di grandi strutture dati o programmi complicati.
Vediamo in dettaglio il funzionamento dell'algoritmo MD4.

Sia data una stringa x di bit di dimensione qualsiasi. Il primo passo dell'algoritmo costruisce, a partire da x, un array

$$M = M[0]M[1]\dots M[N-1],$$

dove $N = 0 \bmod 16$, $|M[i]| = 32$, ed $M[i]$ viene detta parola con $1 \le i \le N-1$ mediante le tre seguenti istruzioni:

1. $d = 447 - (|x| \bmod 512)$
2. Sia l la rappresentazione binaria di $|x| \bmod 2^{64}$, $|l| = 64$
3. $M = x\|1\|0^d\|l$

L'array M viene costruito concatenando la stringa x con un uno e d zeri tali da avere che $|x10\cdots0| = 448 \bmod 512$; a questi valori poi vengono aggiunti 64 bit, indicati con l, che sono la rappresentazione binaria della lunghezza di x. Nell'improbabile evento che la grandezza di x sia maggiore di 2^{64} allora vengono presi in considerazione solo i primi 64 bit di x. Quindi risulta:

$$|M| = |x10 \cdots 0l| = 0 \bmod 512.$$

Da questa espressione notiamo che $|M|$ è divisibile per 512; pertanto, scomponendo M in parole di 32 bit ciascuna, abbiamo esattamente $K \cdot 512/32 = K \cdot 16$ parole. Il secondo passo dell'algoritmo (vedi Figura 1) costruisce i 128 bit finali che rappresentano l'output ovvero il message digest. Nel passo 1 vengono inizializzate delle parole di 32 bit ciascuna che chiameremo *registri* e indicheremo con le lettere A, B, C e D. Sará la concatenazione di questi registri a fornire il valore finale dell'output. L'array M viene elaborato in gruppi di 16 parole alla volta (primo ciclo di for nel passo 2).

Algoritmo MD4

1. $A = 67452301(hex)$
   $B = efcdab89(hex)$
   $C = 98badcfe(hex)$
   $D = 10325476(hex)$
2. **for i**=0 **to** $N/16 - 1$ **do**
3.   **for** $j = 0$ **to** 15 **do**
           $X[j] = M[16i + j]$
4. $AA = A$
5. $BB = B$
6. $CC = C$
7. $DD = D$
8.   **Round1**
9.   **Round2**
10.   **Round3**
11. $A = A + AA$
12. $B = B + BB$
13. $C = C + CC$
14. $D = D + DD$

Figura 1

Ad ogni passo del secondo for (passo 3) le successive 16 parole di M vengono memorizzate in un array X. I valori dei quattro registri iniziali vengono memorizzati nelle variabili temporanee AA, BB, CC e DD (passi 4, 5, 6, 7). In seguito vengono eseguiti tre diversi round in ognuno dei quali vengono effettuate delle operazioni che modificano i valori dei registri tramite delle funzioni booleane e l'operazione d'addizione. I round sono illustrati in dettaglio nelle figure 2, 3, 4. Gli operatori che vengono adoperati nei tre round sono:

1. gli operatori logici bit a bit AND, OR, XOR e NOT.

2. l'operazione $X + Y$ che è definita come segue: le parole X ed Y rapprentano gli interi x ed y, dove $0 \leq x < 2^{32}$ e $0 \leq y < 2^{32}$. Per numeri interi positivi n ed m, sia $n \bmod m$ il resto che si ottiene dividendo n ed m. Computiamo

   $$z = (x + y) \bmod 2^{32}.$$

   Allora $0 \leq z < 2^{32}$. Convertiamo z in una parola, Z, e definiamo $Z = X + Y$.

3. un' operazione di shift circolare sinistro che sará indicata con $\lll$.

Osservando gli assegnamenti all'interno dei round notiamo che intervengono delle particolari funzioni booleane bit a bit f, g, h che prendono come input tre parole X, Y, Z e restituiscono come output una singola parola di 32 bit.

$$\begin{aligned}
f(X, Y, Z) &= (X \wedge Y) \vee ((\neg X) \wedge Z) \\
g(X, Y, Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) \\
h(X, Y, Z) &= X \oplus Y \oplus Z.
\end{aligned}$$

Si noti che la funzione f agisce come lo statement condizionale:

$$f = \textbf{if } X \textbf{ then } Y \textbf{ else } Z.$$

Tutte queste operazioni sono molto veloci; infatti, l'unica operazione aritmetica è la somma modulo 32.

Ritornando all'algoritmo, nei passi 11, 12, 13, 14, vengono introdotte nei registri A, B, C, D, le rispettive somme senza riporto fra i valori dei registri e i valori delle temporanee. Il message digest è proprio rappresentato dai quattro registri A, B, C, e D di 32 bit ciascuno, che compongono appunto l'output finale di 128 bit.

ROUND 1

1. $A = (A + f(B, C, D) + X[0])13$
2. $D = (D + f(A, B, C) + X[1]) \ll 7$
3. $C = (C + f(D, A, B) + X[2]) \ll 11$
4. $B = (B + f(C, D, A) + X[3]) \ll 19$
5. $A = (A + f(B, C, D) + X[4]) \ll 3$
6. $D = (D + f(A, B, C) + X[5]) \ll 7$
7. $C = (C + f(D, A, B) + X[6]) \ll 11$
8. $B = (B + f(C, D, A) + X[7]) \ll 19$
9. $A = (A + f(B, C, D) + X[8]) \ll 3$
10. $D = (D + f(A, B, C) + X[9]) \ll 7$
11. $C = (C + f(D, A, B) + X[10]) \ll 11$
12. $B = (B + f(C, D, A) + X[11]) \ll 19$
13. $A = (A + f(B, C, D) + X[12]) \ll 3$
14. $D = (D + f(A, B, C) + X[13]) \ll 7$
15. $C = (C + f(D, A, B) + X[14]) \ll 11$
16. $B = (B + f(C, D, A) + X[15]) \ll 19$

Figura 2

Occorre tener presente che affinché le operazioni siano eseguite in modo corretto, il computer su cui viene implementato l'algoritmo MD4 deve avere un'architettura little-endian (come nei processori Intel80xxx) in cui la parola $a_1\, a_2\, a_3\, a_4$ di 4 bytes ($a_i \in \{0 \dots 255\}$) rappresenta l'intero

$$a_4 2^{24} + a_3 2^{16} + a_2 2^8 + a_1.$$

Ricordiamo che, al contrario, nell'architettura big-endian (come sulle Sun SPARCstation), la stessa parola rappresenta l'intero

$$a_1 2^{24} + a_2 2^{16} + a_3 2^8 + a_4.$$

Anche se MD4 presuppone che ci sia un'architettura *little-endian* ció non compromette la sua indipendenza rispetto al sistema in quanto, qualora si volesse far girare MD4 su un computer *big-endian*, sarebbe solo necessario eseguire l'operazione di addizione $X + Y$ nel modo seguente:

1. Scambia x_1 con x_4; x_2 con x_3; y_1 con y_4; e y_2 con y_3.
2. Calcola $Z = X + Y \bmod 2^{32}$
3. Scambia z_1 con z_4; e z_2 con z_3.

ROUND 2

1. $A = (A + g(B, C, D) + X[0] + 5A827999) \ll 3$
2. $D = (D + g(A, B, C) + X[4] + 5A827999) \ll 5$
3. $C = (C + g(D, A, B) + X[8] + 5A827999) \ll 9$
4. $B = (B + g(C, D, A) + X[12] + 5A827999) \ll 13$
5. $A = (A + g(B, C, D) + X[1] + 5A827999) \ll 3$
6. $D = (D + g(A, B, C) + X[5] + 5A827999) \ll 5$
7. $C = (C + g(D, A, B) + X[9] + 5A827999) \ll 9$
8. $B = (B + g(C, D, A) + X[13] + 5A827999) \ll 13$
9. $A = (A + g(B, C, D) + X[2] + 5A827999) \ll 3$
10. $D = (D + g(A, B, C) + X[6] + 5A827999) \ll 5$
11. $C = (C + g(D, A, B) + X[10] + 5A827999) \ll 9$
12. $B = (B + g(C, D, A) + X[14] + 5A827999) \ll 13$
13. $A = (A + g(B, C, D) + X[3] + 5A827999) \ll 3$
14. $D = (D + g(A, B, C) + X[7] + 5A827999) \ll 5$
15. $C = (C + g(D, A, B) + X[11] + 5A827999) \ll 9$
16. $B = (B + g(C, D, A) + X[15] + 5A827999) \ll 13$

Figura 3

Le implementazioni software di MD4 sulle Sun SPARCstations raggiungono velocitá di 1.4 Mbytes/sec e anticipiamo che questo parametro è di gran lunga superiore rispetto alle prestazioni dell'MD5 e dell'SHS.

ROUND 3

1. $A = (A + h(B, C, D) + X[0] + 6ED9EBA1) \ll 3$
2. $D = (D + h(A, B, C) + X[8] + 6ED9EBA1) \ll 9$
3. $C = (C + h(D, A, B) + X[4] + 6ED9EBA1) \ll 11$
4. $B = (B + h(C, D, A) + X[12] + 6ED9EBA1) \ll 15$
5. $A = (A + h(B, C, D) + X[2] + 6ED9EBA1) \ll 3$
6. $D = (D + h(A, B, C) + X[10] + 6ED9EBA1) \ll 9$
7. $C = (C + h(D, A, B) + X[6] + 6ED9EBA1) \ll 11$
8. $B = (B + h(C, D, A) + X[14] + 6ED9EBA1) \ll 15$
9. $A = (A + h(B, C, D) + X[1] + 6ED9EBA1) \ll 3$
10. $D = (D + h(A, B, C) + X[9] + 6ED9EBA1) \ll 9$
11. $C = (C + h(D, A, B) + X[5] + 6ED9EBA1) \ll 11$
12. $B = (B + h(C, D, A) + X[13] + 6ED9EBA1) \ll 15$
13. $A = (A + h(B, C, D) + X[3] + 6ED9EBA1) \ll 3$
14. $D = (D + h(A, B, C) + X[11] + 6ED9EBA1) \ll 9$
15. $C = (C + h(D, A, B) + X[7] + 6ED9EBA1) \ll 11$
16. $B = (B + h(C, D, A) + X[15] + 6ED9EBA1) \ll 15$

Figura 4

Purtroppo il 18 Settembre del 1995 è stato applicato con successo ad MD4 un metodo per trovare efficientemente delle collisioni [5]. Implementando questo metodo, in pochi minuti si possono trovare collisioni per $MD4$ su un normale PC. Consideriamo il seguente esempio:

Esempio 3 *Il primo messaggio M è la sequenza delle seguenti parole a 32 bit:*

$$
\begin{aligned}
M_0 &= 0x13985E12 \\
M_1 &= 0x748A810B \\
M_2 &= 0x4D1DF15A \\
M_3 &= 0x181D1516 \\
M_4 &= 0x2D6E09AC \\
M_5 &= 0x4B6DBDB9 \\
M_6 &= 0x6464B0C8 \\
M_7 &= 0xFBA1C097 \\
M_8 &= 0xABE17BE0 \\
M_9 &= 0xED1ED4B3 \\
M_{10} &= 0x4120ABF5 \\
M_{11} &= 0x20771029
\end{aligned}
$$

$$\begin{aligned} M_{12} &= 0x20771027 \\ M_{13} &= 0xFDFFFBFF \\ M_{14} &= 0xFFFFBFFB \\ M_{15} &= 0x6774BED2 \end{aligned}$$

Il secondo messaggio M' è definito ponendo $M'_i = M_i (i < 16, i \neq 12)$ e

$$M'_{12} = M_{12} + 1$$

MD4 associa ad M ed M' lo stesso valore hash

$$0x711AD51B0xBBAB5E220x618B1C760x17C15892.$$

6.2 MD5

Descriveremo ora la funzione hash MD5 [10] che non è altro che un'estensione di MD4. L'MD5 rispetto ad MD4 è piú lenta del 30% (circa 0.9 Mbytes/sec su una Sun SPARCstation) ma è molto piú sicura.

L' MD5 prende in input un messaggio di lunghezza qualsiasi e produce un output di 128 bit (il message digest). In dettaglio l'algoritmo MD5 funziona al modo seguente. Supponiamo di avere un messaggio x come input di grandezza arbitraria. In cinque passi riusciamo ad ottenere il message digest.

Come per MD4 il primo passo dell'algoritmo consiste nell'espansione del messaggio originario. Viene costruito un array M concatenando la stringa x con un uno e d zeri in modo tale da avere $\mid x10 \cdots 0 \mid =$ 448 mod 512. A questi valori, in un secondo momento, vengono aggiunti 64 bit che rappresentano la lunghezza del messaggio originario x.

Quindi risulta :

$$\mid M \mid = \mid x10 \cdots 0l \mid = 0 \bmod 512$$

cioé il messaggio espanso ha una lunghezza che è esattamente un multiplo di 512 bit, quindi multiplo anche di 16 parole di 32 bit ciascuna. Indichiamo con $M = [0 \cdots N-1]$ la stringa risulante con N multiplo di 16.

Nel terzo passo dell'algoritmo si inizializzano quattro parole (A, B, C, D) che verrano poi utilizzate per computare il message digest. A, B ,C e D non sono altro che registri a 32 bit che vengono inizializzati con i seguenti valori esadecimali:

$$\begin{aligned} A &= 01234567 \\ B &= 89abcdef \\ C &= fdecba98 \\ D &= 76543210 \end{aligned}$$

Al quarto passo vengono eseguiti quattro round su ogni blocco di 16 parole di M in cui vengono utilizzate, nell'ordine, delle funzioni ausiliarie che prendono in input tre parole di 32 bit e producono come output una sola parola di 32 bit. Queste funzioni sono:

$$\begin{aligned} \mathbf{F}(X,Y,Z) &= (X \wedge Y) \vee ((\neg X) \wedge Z) \\ \mathbf{G}(X,Y,Z) &= (XZ) \vee Y(\neg Z) \\ \mathbf{H}(X,Y,Z) &= X \oplus Y \oplus Z \\ \mathbf{I}(X,Y,Z) &= Y \oplus (X \vee (\neg Z)). \end{aligned}$$

Algoritmo MD5

1. **for** $i = 0$ **to** $N/16-1$ **do**
2. **for** $j = 0$ **to** 15 **do**
   $X[j] = M[i \cdot 16 + j]$.
3. AA= A
4. BB=B
5. CC=C
6. DD=D
7. **Round 1**

$[ABCD\ \ 0\ \ 7\ \ 1]$	$[DABC\ \ 1\ \ 12\ \ 2]$	$[CDAB\ \ 2\ \ 17\ \ 3]$	$[BCDA\ \ 3\ \ 22\ \ 4]$
$[ABCD\ \ 4\ \ 7\ \ 5]$	$[DABC\ \ 5\ \ 12\ \ 6]$	$[CDAB\ \ 6\ \ 17\ \ 7]$	$[BCDA\ \ 7\ \ 22\ \ 8]$
$[ABCD\ \ 8\ \ 7\ \ 9]$	$[DABC\ \ 9\ \ 12\ \ 10]$	$[CDAB\ \ 10\ \ 17\ \ 11]$	$[BCDA\ \ 11\ \ 22\ \ 12]$
$[ABCD\ \ 12\ \ 7\ \ 13]$	$[DABC\ \ 13\ \ 12\ \ 14]$	$[CDAB\ \ 14\ \ 17\ \ 15]$	$[BCDA\ \ 15\ \ 22\ \ 16]$

8. **Round 2**

$[ABCD\ \ 1\ \ 5\ \ 17]$	$[DABC\ \ 6\ \ 9\ \ 18]$	$[CDAB\ \ 11\ \ 14\ \ 19]$	$[BCDA\ \ 0\ \ 20\ \ 20]$
$[ABCD\ \ 5\ \ 5\ \ 22]$	$[DABC\ \ 10\ \ 9\ \ 22]$	$[CDAB\ \ 15\ \ 14\ \ 23]$	$[BCDA\ \ 4\ \ 20\ \ 24]$
$[ABCD\ \ 9\ \ 5\ \ 25]$	$[DABC\ \ 14\ \ 9\ \ 26]$	$[CDAB\ \ 3\ \ 14\ \ 27]$	$[BCDA\ \ 8\ \ 20\ \ 28]$
$[ABCD\ \ 13\ \ 5\ \ 29]$	$[DABC\ \ 2\ \ 9\ \ 30]$	$[CDAB\ \ 7\ \ 14\ \ 21]$	$[BCDA\ \ 12\ \ 20\ \ 32]$

9. **Round 3**

$[ABCD\ \ 5\ \ 4\ \ 33]$	$[DABC\ \ 8\ \ 11\ \ 34]$	$[CDAB\ \ 11\ \ 16\ \ 35]$	$[BCDA\ \ 14\ \ 23\ \ 36]$
$[ABCD\ \ 1\ \ 4\ \ 37]$	$[DABC\ \ 4\ \ 11\ \ 38]$	$[CDAB\ \ 7\ \ 16\ \ 39]$	$[BCDA\ \ 10\ \ 23\ \ 40]$
$[ABCD\ \ 13\ \ 4\ \ 41]$	$[DABC\ \ 0\ \ 11\ \ 42]$	$[CDAB\ \ 3\ \ 16\ \ 43]$	$[BCDA\ \ 6\ \ 23\ \ 44]$
$[ABCD\ \ 9\ \ 4\ \ 45]$	$[DABC\ \ 12\ \ 11\ \ 46]$	$[CDAB\ \ 15\ \ 16\ \ 45]$	$[BCDA\ \ 2\ \ 23\ \ 48]$

10. **Round 4**

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

11. $A = A + AA$
12. $B = B + BB$
13. $C = C + CC$
14. $D = D + DD$

Nei round vengono addizionate anche delle costanti che sono conservate in una tavola a 64 elementi $T[1 \cdots 64]$. L' i-esimo elemento della tavola è :

$$T[i] = \lfloor 4294967296 \cdot \mid \sin(i) \mid \rfloor$$

dove i è espresso in radianti e 4294967296 rappresenta in decimale il valore 2^{32}. Se indichiamo con i il numero di iterazione che stiamo eseguendo, con s uno shift ciclico e con $x[k]$ il pezzo di messaggio che stiamo computando, l'operazione generica che viene calcolata nei quattro round è:

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

dove W sta per una qualsiasi delle quattro funzioni F, G, H, I. All'interno dell'algoritmo questa operazione nei round viene indicata solo con i paramentri $[ABCD\ k\ s\ i]$.

Nell'ultimo passo dell'algoritmo viene prodotto l'output che sará costituito dal contenuto dei registri A, B, C, e D.

i	**round 1**	i	**round 2**	i	**round 3**	i	**round 4**
0	$0xd76aa478$	16	$0xf61e2562$	32	$0xfffa3942$	48	$0xf4292244$
1	$0xe8c7b756$	17	$0xc040b340$	33	$0x8771f681$	49	$0x432aff97$
2	$0x242070db$	18	$0x265e5a51$	34	$0x6d9d6122$	50	$0xab9423a7$
3	$0xc1bdceee$	19	$0xe9b6c7aa$	35	$0xfde5380c$	51	$0xfc93a039$
4	$0xf57c0faf$	20	$0xd62f105d$	36	$0xa4beea44$	52	$0x655b59c3$
5	$0x4787c62a$	21	$0x2441453$	37	$0x4bdecfa9$	53	$0x8f0ccc92$
6	$0xa8304613$	22	$0xd8a1e681$	38	$0xf6bb4b60$	54	$0xffeff47d$
7	$0xfd469501$	23	$0xe7d3fbc8$	39	$0xbebfbc70$	55	$0x85845dd1$
8	$0x698098d8$	24	$0x21e1cde6$	40	$0x289b7ec6$	56	$0x6fa87e4f$
9	$0x8b44f7af$	25	$0xc33707d6$	41	$0xeaa127fa$	57	$0xfe2ce6e0$
10	$0xffff5bb1$	26	$0xf4d50d87$	42	$0xd4ef3085$	58	$0xa3014314$
11	$0x895cd7be$	27	$0x455a14ed$	43	$0x4881d05$	59	$0x4e0811a1$
12	$0x6b901122$	28	$0xa9e3e905$	44	$0xd9d4d039$	60	$0xf7537e82$
13	$0xfd987193$	29	$0xfcefa3f8$	45	$0xe6db99e5$	61	$0xbd3af235$
14	$0xa679438e$	30	$0x676f02d9$	46	$0x1fa27cf8$	62	$0x2ad7d2bb$
15	$0x49b40821$	31	$0x8d2a4c8a$	47	$0xc4ac5665$	63	$0xeb86d391$

Tabella 1: Tabella delle costanti T[i]

6.2.1 Differenze fra MD4 ed MD5

Le principali variazioni in MD5 rispetto a MD4 sono le seguenti:

1. è stato aggiunto un quarto round che funziona esattamente come gli altri usando la funzione $\mathbf{I}(X,Y,Z) = Y \oplus (X \vee (\neg Z))$;
2. in ogni passo i vi è un'unica costante additiva $T[i] = [4294967296 \times |\sin(i)|]$, dove i è espresso in radianti e $4294967296 = 2^{32}$; tale costante, in MD4, è presente solo nei round 2 e 3 ed è fissata indipendentemente dal passo;
3. la funzione **G** è stata cambiata
$$\mathbf{G}(X,Y,Z) = XZ \vee Y(\neg Z)$$
4. nei round i valori calcolati in ogni passo vengono addizionati con i risultati del passo precedente provocando cosí il cosiddetto *effetto valanga*;
5. lo shift in ogni round è stato approssimativamente ottimizzato per produrre un effetto di avanzamento piú veloce. Lo shift è diverso in round differenti.

L' MD5 è un algoritmo facile da implementare. Per quanto concerne la sicurezza, Rivest ha congetturato che per trovare una collisione sono richieste circa 2^{64} operazioni mentre la difficoltá di trovare una preimmagine di un dato message digest è dell'ordine di 2^{128} operazioni. Pertanto possiamo considerare l' MD5 sicura. Fino ad oggi sono stati effettuati degli attacchi solo sui singoli round ma l'algoritmo per intero non è stato ancora attaccato.

6.3 SHS

SHS (Secure Hash Standard) è dal 1993 lo standard americano, anche se nel Luglio 1994 è stato modificato. Questo standard fornisce le specifiche di un algoritmo, detto SHA (Secure Hash Algoritm) [6], che computa una rappresentazione ridotta di un messaggio o di un file di dati. Quando riceve in input un messaggio di una qualsiasi lunghezza minore di 2^{64}, l' SHA produce un output di 160 bit che chiameremo, come per l'MD4 e l'MD5, *message digest*.
L'SHS ricalca il funzionamento di MD4 anche se presenta alcune importanti differenze; la prima, lampante, sta nel fatto che per produrre un output di 160 bit abbiamo bisogno di cinque registri di 32 bit invece dei quattro utilizzati dall'MD4 e dall'MD5. Inoltre questo standard è stato progettato per essere implementato su una architettura di tipo big-endian. L' SHS è piú lento, rispetto alle funzioni hash che abbiamo visto in precedenza (0.2 Mbytes/sec su una Sun SPARCstation), ma è anche considerato piú sicuro.
5 Gli operatori applicati alle parole di 32-bit sono gli stessi usati in MD4 e MD5. L'SHA processa blocchi di 512 bit. Quindi, come nell'MD4, ad un messaggio input X si aggiunge un "1", m "0" seguiti da 64 bit che rappresentano la lunghezza di X. Si ottiene quindi un messaggio di $512 \cdot n$ bit, ovvero di $16 \cdot n$ parole. Il messaggio espanso puó essere visto come una sequenza di n blocchi $M(1), M(2), \cdots, M(n)$ dove ogni blocco contiene 16 parole. Infatti l'algoritmo lavora su 16 parole ad ogni passo.
Nell'SHA intervengono alcune funzioni logiche $f_0, f_1, \cdots, f_{79}$. Ogni funzione f_t, con $0 \leq t \leq 79$, opera su tre parole di 32 bit, che indicheremo con B, C e D, e produce come output una sola parola di 32 bit. $f_t(B, C, D)$ viene definita come segue:

$$f_t(B,C,D) = \begin{cases} (X \wedge Y) \vee ((\neg X) \wedge Z) & (0 \leq t \leq 19) \\ X \oplus Y \oplus Z & (20 \leq t \leq 39) \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) & (40 \leq t \leq 59) \\ Y \oplus (X \vee (\neg Z)) & (60 \leq t \leq 79) \end{cases}$$

Viene anche utilizzata una sequenza di costanti, $K_0, K_1, \cdots K_{79}$ che è definita al modo seguente:

$$K_t = \begin{cases} 5a827999 & (0 \leq t \leq 19) \\ 6ed9eba1 & (20 \leq t \leq 39) \\ 8f1bbcdc & (40 \leq t \leq 59) \\ ca62c1d1 & (60 \leq t \leq 79). \end{cases}$$

Un primo passo dell'algoritmo consiste nell'espandere le 16 parole di turno in 80 parole. L'espansione avviene considerando le 16 parole del messaggio originario x, che indicheremo con $X[0] \cdots X[15]$, e aggiungendo ad esse le 64 parole che vengono computate tramite la relazione di ricorrenza :

$$X[j] = X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16]$$

con $16 \leq j \leq 79$. Il risultato di questa relazione è che ognuna delle parole $X[16], \cdots, X[79]$ rappresenta lo *xor* di un predeterminato sottinsieme delle parole $X[0], \cdots, X[15]$. Per esempio, abbiamo che

$$\begin{aligned} X[16] &= X[0] \oplus X[2] \oplus X[8] \oplus X[13] \\ X[19] &= X[0] \oplus X[2] \oplus X[3] \oplus X[5] \oplus X[8] \oplus X[11] \oplus X[13] \end{aligned}$$

$$\vdots$$

La modifica che è stata apportata nel 1994 all'SHA è stata effettuata proprio in questa fase di espansione. E' stato introdotto nella relazione di ricorrenza uno shift circolare che permuta i contenuti dei registri di una posizione. Quindi lo standard attuale prevede la relazione di ricorrenza:

$$X[j] = X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16] \ll 1.$$

Durante la computazione del message digest ci serviamo di due buffer contenenti cinque parole ciascuno. Le parole del primo buffer l'indichiamo con A, B, C, D e E e le cinque parole del secondo buffer con $H_0, H_1, H_2, H_3 e H_4$.

Le parole $H_0, H_1, H_2, H_3 e H_4$ vengono inizializzate al modo seguente:

$$\begin{array}{rcl} H_0 & = & 67452310 \\ H_1 & = & EFCDAB89 \\ H_2 & = & 98BADCFE \\ H_3 & = & 10325476 \\ H_4 & = & C3D2E1F0 \end{array}$$

Per generare il message digest vengono processati in ordine i blocchi $M(1), M(2), \cdots, M(n)$. Al passo i-esimo, in cui si elabora $M(i)$, le 16 parole contenute nel blocco vengono memorizzate in $X(0), X(1), \cdots, X(15)$. I passi dell'algoritmo sono i seguenti:

Algoritmo SHS

for $t = 16$ **to** 79 **do**
    $X[t] = X[t-3] \oplus X[t-8] \oplus X[t-14] \oplus X[t-16] \ll 1$
$A = H_0$
$B = H_1$
$C = H_2$
$D = H_3$
$E = H_4$
for $t = 0$ **to** 79 **do**
    $TEMP = (A \ll 5) + f_t(B, C, D) + E + X[t] + K_t$
    $E = D$
    $D = C$
    $C = (B \ll 30)$
    $B = A$
    $A = TEMP$
$H_0 = H_0 + A$
$H_1 = H_1 + B$
$H_2 = H_2 + C$
$H_3 = H_3 + D$
$H_4 = H_4 + E$

Si noti che la variabile temporanea TEMP viene utilizzata per aggiornare ad ogni passo i registri A, B, C, D ed E. L'output finale dell'SHA è invece rappresentato dai cinque registri H_0, H_1, H_2, H_3 e H_4.

7 Certificazione del tempo

Certificare il tempo di un insieme di informazioni, rappresentate su un determinato supporto fisico significa, essenzialmente, associare all'insieme un *marchio temporale* **(time-stamp)** che collochi in modo inequivocabile, non falsificabile e controllabile in futuro l'insieme in uno specifico contesto temporale.
Sebbene questa operazione venga svolta in modo soddisfacente in molti ambiti della vita umana, la crescente tendenza alla digitalizzazione dell'informazione e delle comunicazioni spinge ad un ripensamento delle tecniche solitamente usate al fine di far fronte alle inedite e spesso subdole debolezze che i nuovi strumenti recano con sè.
Come scrivono S.Haber e W.S.Stornetta [7], inventori del primo *notaio digitale* della storia, "la prospettiva di un mondo in cui tutti i documenti testuali, grafici, audio e video esistono in formato digitale su supporti fisici facilmente modificabili pone il problema di come certificare il momento in cui il documento è stato creato o cambiato. Il problema è di come fissare il

tempo sui dati e non sul supporto fisico..." Per capire l'importanza del problema consideriamo una serie di situazioni reali che possono richiedere la certificazione del tempo.
Un primo esempio potrebbe essere quello di due scienziati impegnati nello stesso settore di ricerca. Supponiamo che lo scienziato A, nel suo laboratorio, effettui una serie di sperimentazioni dalla cui analisi perviene ad un'importante scoperta. Allora A registra tale scoperta con la data in cui è avvenuta e "qualcuno" garantisce che tale data è corretta. Se in un altro laboratorio lo scienziato B perviene allo stesso risultato in un momento diverso ed effettua la registrazione del tempo al pari di A allora dal confronto tra le date si riuscirá ad attribuire la paternitá della scoperta. In questo caso una corretta "certificazione del tempo" permette di dirimere un contenzioso.
Un'altra situazione potrebbe essere quella in cui un utente usa uno schema di firma basato su un crittosistema per firmare messaggi. Qualora il crittosistema venisse rotto chiunque potrebbe inviare messaggi contraffatti. L'utente ha allora la necessitá di provare che le sue firme da un certo momento in poi sono false. Pertanto la "certificazione del tempo" di rottura è importante per evitare contraffazioni future. D'altra parte, per motivi vari, l'utente potrebbe di spontanea volontá svelare i segreti del proprio crittosistema poichè potrebbe aver inviato messaggi di cui vorrebbe perdere la paternitá. In questo caso la "certificazione del tempo di rottura" è di fondamentale importanza anche per rendere validi tutti gli usi del crittosistema effettuati prima della sua rottura. Le varie carte di credito e di servizi oggi in uso nonché le molte forme di home banking e di commercio in rete possono essere un buon esempio di applicazione del modello di situazione appena descritto. I n questo caso la "certificazione del tempo" diviene uno strumento per validare o annullare atti pubblici.
Gli esempi potrebbero essere, insomma, molteplici. Oltre ai casi "critici" esposti, in cui la necessitá della certificazione del tempo è palese e il vantaggio di falsificazioni altrettanto, esistono moltissimi altri casi di documenti per i quali in prima istanza sembra inutile un time-stamp in quanto difficile pensare all'utilitá di falsificazioni. Tali documenti possono essere classificati come "falsificabili in tempo impredicibile". Relazioni scritte, fotografie, informazioni grafiche etc. possono rientrare in questa classe. Anche per essi, tuttavia, la certificazone del tempo puó essere importante per risolvere complesse situazioni che potrebbero venire a crearsi.
Esistono diverse forme di time-stamping non digitali usate in molte attivitá umane. Nel caso, per esempio, dei laboratori scientifici una procedura di certificazione del tempo consiste nell'annotazione giornaliera in un notebook del lavoro effettuato. Le date sulle pagine vengono segnate l'una dopo l'altra senza lasciare pagine in bianco. Tale accortezza rende difficile le manomissioni. Inoltre il notebook viene vidimato da un notaio o da un dirigente facente funzioni, operazione che aumenta la validitá della certificazione. Infatti l'esistenza del notebook e la procedura di controllo stabilita, forniscono una notevole garanzia.
Per quanto riguarda invece il caso di un singolo che necessita di certificare il tempo di un documento ma non vuole rivolgersi ad un notaio, questi potrebbe spedire il documento in una busta chiusa a se stesso e conservarla chiusa. In tal caso il timbro postale certificherebbe l'esistenza del documento giá nella data apposta sulla busta.
Altra idea interessante per rafforzare la credibilitá e la correttezza dei controlli di un time-stamping e, piú in generale dell'autenticazione di un documento è rinvenibile nelle procedure spesso usate nelle compagnie finanziarie in cui gli stessi atti vengono verificati da *una catena di persone*.
Molte delle intuizioni presenti in queste tecniche possono essere utilizzate, seppur in forme e strategie diverse, in molti schemi matematici per la certificazione del tempo per documenti digitali. Occorre peró osservare che tutte le idee presentate poggiano su due assunzioni: primo,

le registrazioni possono essere prese in esame solo per significativi segni di manomissione; secondo, c'è un'altra parte che controlla il documento la cui onestá o imparzialitá nei confronti del documento puó metterne in dubbio la validitá.

Queste ipotesi creano notevoli problemi per documenti creati e conservati in forma digitale. Questi, infatti, sono semplici da manomettere e l'operazione non lascia necessariamente segni sul mezzo fisico. Quello di cui si ha bisogno è un metodo di time-stamping che soddisfi le seguenti tre proprietá:

1. la data prodotta sul documento deve essere totalmente indipendente dal supporto fisico su cui le informazioni vengono mantenute.
2. deve essere impossibile cambiare anche un solo bit del documento senza che la falsificazione possa essere riscontrata;
3. deve essere impossibile apporre su un documento una data differente da quella effettiva. (Impossibilitá di predatare o postdatare).

Nel seguito considereremo delle soluzioni matematicamente esatte e computazionalmente pratiche al problema del time-stamping, partendo dalla descrizione di una soluzione banale e fornendo informazioni su un sistema effettivamente in uso.

7.1 L'ambiente

Lo scenario in cui analizzeremo il problema è quello di una rete distribuita di utenti (individui, compagnie, etc. a cui faremo riferimento col termine di *clienti*), ad ognuno dei quali è associato un unico codice identificativo.

Una soluzione al problema del time-stamping è costituito da piú parti: deve esistere una procedura eseguita immediatamente quando un cliente desidera un time-stamping di un documento; deve esistere un metodo che permetta al cliente di verificare che questa procedura sia stata eseguita correttamente; ci deve essere una procedura per fronteggiare la messa in dubbio, da parte di qualcuno, della validitá del time-stamping del documento.

Come in un problema di cifratura, un compito delicato è quello di caratterizzare in modo preciso la sicurezza raggiunta da uno schema di time-stamping. Una buona soluzione è quella in cui è difficile o impossibile produrre un time-stamping falso, qualora siano state fatte delle ragionevoli assunzioni sulle capacità computazionali degli utenti dello schema e sulla complessità del problema computazionale, e possibilmente, sulla fedeltá degli utenti.

7.2 Soluzione naive

Una soluzione naive (banale) è quella della *digital safety-deposit box* (cassetta di sicurezza digitale) il cui funzionamento è il seguente: quando un cliente ha un documento di cui vuole effettuare il time-stamping, spedisce il documento a un servizio di time-stamping, il TSS. Il servizio registra la data e l'ora in cui il documento è arrivato e ne trattiene una copia in custodia. Se l'integritá del documento viene messa in discussione, quest'ultimo puó essere confrontato con la copia in custodia al TSS al fine di stabilire se ha subito una modifica. I problemi sollevati da quest'approccio sono:

1. **Privatezza**: non è garantita né rispetto ad un qualsiasi altro cliente né rispetto al TSS. Il cliente deve preoccuparsi non solo della sicurezza del documento in suo possesso ma anche di quello del documento in custodia al TSS.
2. **Tempo e memoria**: sia il tempo necessario per spedire il documento, sia la quantitá di memoria richiesta al TSS dipendono dalla lunghezza del documento stesso. Se quest'ultimo è molto lungo le dimensioni di tempo e spazio potrebbero essere proibitive.
3. **Incompetenza**: la copia del documento in possesso del TSS potrebbe essere stata trasmessa in modo non corretto e quindi il TSS potrebbe aver effettuato un time-stamping del documento non corretto oppure, la copia, potrebbe venire falsificata o persa in qualsiasi momento mentre è in possesso del TSS. Ognuna di queste situazioni rende non valida la rivendicazione di time-stamping di un cliente.
4. **Fiducia**: anche in questa soluzione il problema fondamentale non è superato: niente, nello schema, previene la collusione tra il TSS ed un cliente che vuole rivendicare un time-stamping di un documento effettuato in una data e tempo diversi da quelli reali.

Ultimamente su internet si stanno avvicendando compagnie private che, senza nessuna garanzia legale, offrono un servizio di time-stamping. Il loro scopo è quello di sperimentare sistemi alternativi di certificazione del tempo. Queste compagnie si avvalgono di un server di posta elettronica al quale un utente puó inviare un documento di cui vuole certificare il possesso in quel determinato momento. Il compito di queste agenzie consiste nell' allegare un certificato temporale al documento e di riinviarlo al proprietario. Per creare il certificato temporale, le agenzie, usano generalmente PGP. Nel caso in cui il proprietario ci tenga a mantenere privati i contenuti del documento ne puó inviare una funzione hash ottenuta con MD5 o con l'utility di firma distaccata del PGP. Anche se la testimonianza di queste societá non ha nessuna valenza legale puó comunque essere presa in considerazione nel momento in cui si vuole dimostrare di aver avuto in possesso un dato documento prima di una determinata data. Uno dei server che offre questo servizio è **stamp@timestamp.mit.edu**.

Il formato del documento riinviato dal server al proprietario è una coppia documento-certificato, dove il certificato è applicato alla funzione hash del documento. Insieme al time-stamping l'agenzia appone la propria firma digitale che servirá in futuro per verificare la veridicitá del certificato. Per questo motivo sui siti di queste agenzie viene pubblicata la chiave pubblica con cui testare la firma. La sicurezza di questo schema si basa sulla bontá della funzione hash e sulla sicurezza dello schema di firme. Generalmente l'algoritmo di hashing è l' MD5 e lo schema di firma è l'RSA. Entrambi hanno una sicurezza molto elevata, almeno attualmente.

Un problema potrebbe sorgere per files destinati ad avere una lunga vita. In futuro infatti potrebbe accadere di trovare il modo per intaccare la sicurezza di MD5 e RSA, e quindi poter manomettere il timestamping. Una soluzione è quella di generare un nuovo documento formato dalla coppia vecchio documento-certificato, ed applicarvi la certificazione con un nuovo algoritmo.

7.3 Servizio di time-stamping fidato

Presentiamo ora un miglioramento alla soluzione naive proposta nella Sezione 7.2 che risolve i problemi esposti nei punti 1, 2 e 3 mentre lascia aperta la questione della fiducia.

Supponiamo che il TSS sia fidato. I miglioramenti che descriviamo si basano sulle funzioni hash e sulle firme.

7.3.1 Soluzione hash

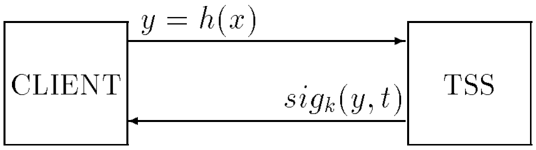
Supponiamo di usare una famiglia di funzioni hash libere da collisioni (vedi Sezione 2.1).

Il cliente invece di inviare al TSS il messaggio x manda il suo valore hash $y = h(x)$. Ai fini dell'autenticazione, effettuare un time-stamping di y equivale ad effettuare il time-stamping di x, per le proprietá di cui gode h. Questo accorgimento permette di risolvere il problema della privatezza e riduce quello del tempo e della memoria. A seconda dell'implementazione, e degli obiettivi da raggiungere, potrá essere usata una singola funzione per tutti i clienti o una per cliente.

Nel seguito parleremo sempre del time-stamping di y e supporremo che nella procedura di convalidazione del time-stamping ci sia una parte in grado di produrre il documento preimmagine x tale che $h(x) = y$; l'incapacitá a farlo invalida l'operazione.

7.3.2 Soluzione della firma

Il secondo miglioramento fa uso delle firma digitali. Avendo a disposizione uno schema di firma sicuro, il TSS, una volta ricevuto il valore hash del messaggio, vi aggiunge la data e l'ora, ottenendo un nuovo documento, lo firma e lo manda al cliente.



Verificando la firma, il cliente, è sicuro che il TSS ha realmente elaborato la richiesta, che il valore hash è stato ricevuto correttamente, e che è stato incluso il tempo corretto. Questo permette di risolvere il problema di un' incompetenza presente o futura da parte del TSS e riduce lo spazio necessario al TSS per memorizzare le registazioni.

7.4 Schemi di time-stamping

Purtroppo né l'uso della firma né quello di funzioni hash impedisce al TSS di emettere un time-stamp falso. Idealmente, quello che vogliamo è un meccanismo che garantisce, indipendentemente dal TSS, che il tempo certificato è sempre quello corretto e che, pur volendo, il

TSS risulti incapace di emettere un time-stamp non corretto.

Ci sono tre approcci piuttosto differenti: il primo, vincola un TSS centralizzato, non necessariamente fidato, a produrre un time-stamping in modo tale che la falsificazione sia difficile da produrre; il secondo, consiste nel distribuire la fiducia tra i clienti del servizio; il terzo, consiste nel rendere pubblico su mezzi ampiamente accessibili, alcune informazioni.

7.4.1 Collegamento lineare

La soluzione si basa sul fatto che la sequenza dei clienti che richiedono un time-stamping e i valori hash inviati non possono essere conosciuti in anticipo. Se dunque includiamo nel certificato firmato dal TSS, una sequenza di bit delle richieste dei clienti precedenti, siamo sicuri che il time-stamp si è avuto dopo queste richieste. Ma l'esigenza di includere bit dai documenti precedenti nel certificato, puó essere anche usata per risolvere il problema di vincolare il tempo nell'altra direzione in quanto, chi effettua il time-stamping, non puó pubblicare certificati successivi senza che abbia a disposizione la richiesta corrente.

Descriveremo due varianti di questo schema di collegamento; uno piú semplice, un altro piú complicato ma da preferirsi in pratica. In entrambi, comunque, il TSS, fa uso di una funzione hash, h, libera da collisioni per rendere sicuri i collegamenti.

Volendo essere piú precisi, una richiesta di time-stamping è costituita da una stringa y di lunghezza l e da un codice di identificazione del cliente ID. Quando il TSS riceve la richiesta n-esima, (y_n, ID_n), di time-stamping, effettua le seguenti operazioni:

1. manda al cliente il certificato firmato
$$s = sig_k(C_n)$$
Il certificato, $C_n = (n, t_n, ID_n, y_n, L_n)$, è costituito dal numero sequenziale di richiesta, n, dal tempo, t_n, dal codice del cliente, ID_n, dal valore hash del documento y_n e dall'informazione di collegamento, $L_n = (t_{n-1}, ID_{n-1}, y_{n-1}, h(L_{n-1}))$, ricavata dal certificato precedentemente pubblicato.
2. quando il TSS elabora la richiesta (n+1)-esima manda al cliente precedente ID_n l'identificativo del nuovo cliente ID_{n+1}.

Una volta che il cliente ha ricevuto s e ID_{n+1} puó verificare che s è la firma del TSS di un certificato in cui è stato attribuito un time-stamp corretto al suo documento originario. Qualora ci fossero dei dubbi, colui il quale volesse controllare, prima verificherebbe che (s, ID_{n+1}) è della forma corretta (ossia, che s è la firma di un documento che contiene il valore hash di x). Inoltre per assicurarsi che il cliente e il TSS non siano in collusione, chi sta controllando potrebbe chiamare il cliente ID_{n+1} e chiedergli di produrre il proprio time-stamp (s', ID_{n+2}). Quest'ultimo include una firma

$$s' = sig_k(n+1, t_{n+1}, ID_{n+1}, y_{n+1}, L_{n+1})$$

di un certificato contenente, nell'informazione di collegamento, L_{n+1}, una copia del valore hash y_n. L'informazione L_{n+1} è ulteriormente autenticata dall'inclusione dell'immagine $h(L_n)$

dell'informazione di collegamento L_n.

Praticamente, se si suppone che il cliente che ha effettuato la richiesta $n - esima$ di time-stamping stia barando sul tempo t_n o sul documento stesso, y_n, o sul momento in cui è stato richiesto, n, colui che effettua il controllo, con la collaborazione del cliente $(n+1) - esimo$, conoscendo $L_{n+1} = (t_n, ID_n, y_n, h(L_n))$, puó produrre le prove che il documento non è valido fornendo il tempo t_n realmente associato al documento all'atto della richiesta, il reale valore hash y_n ricevuto dal TSS all'atto della richiesta e mostrando che, per le proprietà della funzione h, la sua richiesta è avvenuta sicuramente prima del cliente ID_{n+1}.

Per assicurarsi che neanche il cliente ID_{n+1} sia in collusione col TSS, colui che controlla potrebbe fare le stesse richieste a ID_{n+2} e proseguire a piacimento. Il processo potrebbe essere effettuato anche all'indietro cominciando dal cliente ID_{n-1}.

In questo schema l'unico modo per falsificare la certificazione è tramite l'aiuto del TSS. Occorre peró osservare che quest'ultimo non puó predatare un documento perché, pur sapendo in anticipo che un certo cliente N vuole farsi predatare un documento (che il TSS non ha ancora ricevuto), nel momento in cui elabora la richiesta che dovrebbe risultare successiva al documento falsificato, non è in grado di inserire l'informazione di collegamento con quest'ultimo e cioé i bit del certificato che avrebbe dovuto produrre effettuando il calcolo di un valore hash sul falso documento. D'altra parte non ha senso supporre che N fornisca al TSS un valore "fittizzio". Il problema fondamentale è, infatti, che lo stesso N, per sua sicurezza, ha usato una funzione hash libera da collisione ed é quindi estremamente difficile anche per lui far risultare un altro documento piuttosto che quello effettivamente fornito.(Dovrebbe essere capace di trovare una collisione...). Il TSS non puó tantomeno predatare il documento all'atto dell'effettiva richiesta di time-stamping in quanto dovrebbe essere in grado di inserire un certificato in una catena giá esistente e questo richiederebbe ancora il calcolo di una collisione per h. In modo analogo il TSS non puó postdatare un documento perché, nel momento in cui sta elaborando la richiesta, non conosce ancora le richieste dopo le quali deve porre quella attuale. D'altro canto non puó postdatarlo in un secondo momento perché dovrebbe ancora una volta inserire un certificato in una catena giá esistente di certificati e, inoltre, quei certificati prodotti dal TSS dopo il reale time-stamp del documento, contengono bit d'informazione relative a quel documento e non ad altri e tali certificati sono stati giá tutti pubblicati.

L'unico modo per produrre un time-stamp falso da parte del TSS è dunque quello di farlo per una catena di certificati abbastanza lunga.

Una variante dello schema consiste nel collegare ogni richiesta alle prossime m richieste. Il TSS in risposta alla richiesta $n - esima$ compie i seguenti passi:

1. Come prima il certificato è $C_n = (n, t_n, ID_n, y_n, L_n)$ dove L_n è dato da

$$L_n = [(t_{n-m}, ID_{n-m}, y_{n-m}, h(L_{n-m})), \ldots, (t_{n-1}, ID_{n-1}, y_{n-1}, h(L_{n-1}).$$

2. Dopo che le m richieste sono state elaborate, il TSS manda al cliente la lista $(ID_{n+1}, \ldots, ID_{n+m})$.

In fase di verifica chiunque puó chiedere a uno degli m clienti successivi alla richiesta $n - esima$ di produrre il proprio time-stamp. Il time-stamp del cliente chiamato in causa,

ID_{n+i}, conterrá nella sua informazione di collegamento, L_{n+i}, una copia delle informazioni su C_n, autenticata dall'inclusione del valore hash calcolato tramite h dell'informazione di collegamento L_n del cliente su cui si sta effettuando la verifica.

Questa seconda variante ha la proprietá che un inserimento corretto di un nuovo documento in un flusso giá esistente di certificati, di cui sono stati effettuati i time-stamp, richiede il calcolo di m collisioni simultanee per h piuttosto che due.

7.4.2 Collegamento casuale

Supponiamo di avere a disposizione uno schema di firma sicuro in modo tale che ogni cliente puó firmare messaggi e un generatore pseudocasuale, G, disponibile a tutti gli utenti (un generatore pseudocasuale è un algoritmo che dato in input un valore genera una sequenza output che, per un qualsiasi algoritmo ammissibile, è indistinguibile da una sequenza casuale).

Se y è il valore hash di cui il cliente ID vuole un time-stamping, allora y sarà il valore in input a G. L'output sará interpretato come una $m-upla$ di codici di identificazione di clienti in maniera standard:

$$G(y) = (ID_1, \ldots, ID_m).$$

Il cliente manda la richiesta (ID, y) ad ognuno degli m clienti e riceve come ritorno dal cliente ID_j un messaggio firmato

$$s_j = sig_k^j(t, ID, y)$$

che contiene il tempo desiderato. In questo modo ognuno dei clienti scelti diventa testimone dell'evento.

Il time-stamp del cliente sará costituito da

$$[(y, ID), (s_1, \ldots, s_m)].$$

Chiunque puó a questo punto verificare le m firme e qualora ci siano dei dubbi sull'autenticitá del time-stamp non saranno necessari ulteriori comunicazioni.

La sicurezza dello schema si basa sul fatto che l'unico modo per produrre un time-stamp scorretto è quello di usare un valore hash y tale che G(y) dia in output m clienti disposti a cooperare per falsare il sistema.

Se in un qualsiasi momento c'è al piú una frazione costante ε di possibili clienti disonesti, il numero medio di valori y che dovrebbero essere provati prima di trovare una $m-upla$ G(y) contenente solo collaboratori fra tutti è $(\frac{1}{\varepsilon})^m$. Dato che G è pseudocasuale, non esiste nessun modo piú veloce per trovare tali y se non quello di sceglierli a caso.

Nella realtá quello su cui si punta non è sulla difficoltá di trovare un tale y ma piuttosto si cerca di scegliere un parametro m per cui la computazione stessa è inammissibile. C'è da osservare che perfino una stima molto pessimistica della percentuale dei clienti corruttibili (90%) non comporta una scelta proibitiva per m.

In questo schema non c'è bisogno di un TSS. L'unica cosa richiesta è che sia possibile comunicare con gli altri clienti, ricevere da essi le firme e che ci sia una directory pubblica di clienti in modo da interpretare in modo standard l'output di G(y).

Inoltre, in una implementazione pratica, sono richieste delle riserve per i clienti non contattabili al momento del time-stamping. Per esempio, scelto un opportuno $m' < m$, il sistema puó accettare risposte firmate da un qualsiasi gruppo di m' clienti tra gli m nominati da G(y). In tal caso è necessario un valore di m molto elevato in modo da ottenere la stessa bassa probabilità di trovare un insieme di collaboratori in modo casuale.

7.4.3 Collegamento ad albero

In questo schema [1] quello che si fa è, essenzialmente, combinare, mediante un albero binario completo, i valori hash inviati da piú utenti contemporaneamente in unico valore che verrá pubblicato. Ogni cliente conserva oltre al suo documento anche i valori hash con cui è stato combinato e per ognuno di tali valori l'informazione indicante se il valore calcolato di recente è stato posto prima o dopo il valore hash con cui è stato combinato. Quest'ultima informazione è ugualmente importante al fine del corretto calcolo del valore associato all'albero in fase di verifica. In altre parole all'utente che invia il proprio valore hash vengono fornite le informazioni per risalire alla radice dell'albero che viene resa pubblica.

Con queste informazioni il cliente puó dimostrare che il documento esisteva prima che il valore hash fosse pubblicato.

In particolare se N è il numero di richieste, ogni cliente riceve dal servizio $\lceil \log_2 N \rceil$ valori hash, uno per ogni livello dell'albero.

Esempio 4 *Sia $N = 8$ allora il cliente ID_1 riceve oltre al valore hash y_1 anche y_2 (quello immediatamente combinato con y_1), y^{II} (quello combinato con y^I), y^{IV} (quello combinato con y^{III}), più le informazioni relative all'ordine in base al quale i valori stessi vengono combinati (vedi Figura 5).*

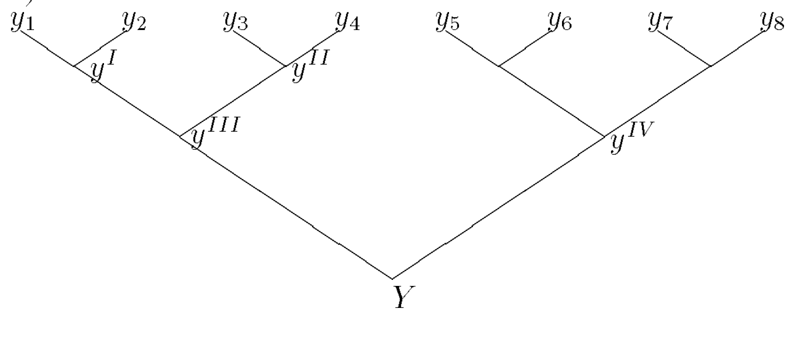


Figura 5: collegamento ad albero

7.4.4 Conclusioni

A seconda delle applicazioni uno schema è da preferirsi all'altro. Ad esempio, in un notebook di laboratorio, l'esatta sequenza delle date registrate potrebbe essere piú importante delle date stesse quindi, sarebbe da preferirsi un collegamento lineare. Per quanto riguarda gli altri due schemi c'è da dire che, nello schema degli alberi il numero di testimoni di un evento aumenta col numero di richieste pervenute, ma non garantisce un numero minimo di testimoni. Inoltre, ogni cliente, deve conservare i valori hash ai fini della validità del time-stamp. Nel collegamento casuale è l'intera popolazione di utenti a poter fare da testimone anche se solo una frazione di essi è coinvolta in ogni evento. Inoltre è la firma rilasciata da ogni cliente scelto casualmente a testimoniare l'evento quindi, la validitá del protocollo, non viene compromessa dal fatto che i testimoni abbiano o meno conservato i documenti che hanno firmato.

I tre schemi non si escludono ma possono essere combinati insieme per minimizzare lo spazio necessario e il numero di eventi da rendere pubblici. Ad esempio, un cliente puó usare un collegamento lineare per effettuare un time-stamping dei suoi record, spedire i valori accumulati in un certo intervallo di tempo ad un servizio che, a intervalli regolari, costruisce collegamenti ad albero. La radice di ognuno di questi alberi viene certificata usando lo schema di collegamento casuale tra tutti i partecipanti al sistema.

7.5 Digital Notary System

Il Digital Notary System è il primo sistema software progettato per risolvere il problema della certificazione del tempo per documenti digitali in modo sicuro, veloce ed economico (nonché per produrre "utili" ai suoi gestori).
E' stato il primo prodotto commercializzato dalla Surety Techonologies Inc., compagnia privata fondata nel 1993 da W.S. Stornetta e S. Haber, in precedenza ricercatori presso il Theoretical Computer Science Research Group alla Bellcore in Morristown, NJ. La Surety Techonologies Inc. ha ricevuto dalla Bellcore i diritti esclusivi per sviluppare e commercializzare prodotti basati sulla tecnologia messa a punto da Stornetta e Haber sulla certificazione digitale.

Il Digital Notary System fonda la sua sicurezza sulle funzioni hash one-way e su una fiducia distribuita ad una serie di testimoni. Lo schema che realizza il sistema è quello del collegamento tramite albero.
Ad un utente, per poter usufruire del servizio, basta avere a disposizione almeno un Intel 386 con 8 MB di RAM e 2 MB su disco fisso, software e hardware che permettono di collegarsi con Internet, il software del Digital Notary Services e un account presso la Surety Technologies Inc.. Il Digital Notary gira sia sotto Windows 3.1, Windows 3.11 che Windows for Workgroups 3.11.

È possibile sia integrare il software con altre applicazioni, in modo tale che ogni record generato venga certificato, oppure selezionare i documenti da certificare. In questo caso è infatti disponibile il Digital Notary Personal Edition software che per un costo di $ 49, fornisce un account presso la Surety Technologies e la possibilitá di certificare 50 documenti. Ció non ne limita l'utilizzo in quanto è possibile acquistare altri certificati a prezzi vantaggiosi per grandi quantitá.

In breve il Digital Notary System funziona al modo seguente: quando giunge una richiesta da parte dell'utente, espressa tramite comando o applicazione, il Digital Notary server software, installato sulla macchina dell'utente, crea un codice hash del documento mediante una funzione hash one-way e lo trasmette via Internet ad un coordinatore di server. Quest'ultimo combina i valori ricevuti nell'ultimo secondo da vari utenti in un albero (vedi Sezione 7.4.3) la cui radice è identificata da un unico valore hash. Tale valore viene collegato ogni secondo ad un Universal Validation Record permanente. Il coordinatore dei server trasmette poi all'utente i dati necessari per verificare il documento, che sono essenzialmente gli altri valori hash con cui quello dell'utente che fa richiesta è stato combinato nonché l'ordine in cui tali valori sono stati combinati insieme, secondo lo schema ad albero precedentemente descritto. Tali dati permettono di risalire all'Universal Validation Record e, quindi, rendono possibile la verifica della validitá del time-stamp effettuato.

Il Digital Notary software del proprio sistema costruisce con le informazioni ricevute un certificato per il documento e lo memorizza in un database proprio o in uno specificato dal cliente. Avendo a disposizione il documento e il certificato, l'utente puó verificarne la validitá ripercorrendo l'intera procedura di costruzione dell'albero da parte del coordinatore dei server e confrontando il risultato con ció che è contenuto nell'Universal Validation Record.

Inoltre frequentemente, ad intervalli regolari, i codici hash generati ogni secondo, vengono combinati creando super codici hash. Questi ultimi forniscono un ulteriore livello di sicurezza impedendo a chiunque abbia accesso al server o all'Universal Validation Record di postdatare qualsiasi documento.

Infine la Surety Technologies, ogni domenica, pubblica un avviso a pagamento sul The New York Times elencando i super codici hash piú recenti dell'ultima settimana. A questo va aggiunto che, per assicurare l'integritá e la disponibilitá dei server, dell'Universal Validation Record vengono memorizzate diverse copie, aggiornate a intervalli regolari, ed inoltre i super codici hash vengono pubblicati su Internet al fine di aumentare la sicurezza. La Surety sta anche pensando di pubblicare questi super codici su CD-ROM con aggiornamenti periodici [11].

7.6 Protocollo di time-stamping basato sul PGP

Nell'ottobre del 1995 ad opera di Matthew Richardson (matthew@itconsult.co.uk) nacque un servizio di time-stamping basato su email. Fu scelto questo tramite, in quanto costituiva la forma piú comune di connessione ad Internet. Il servizio prese il nome di Stamper ed è attualmente in uso.

Stamper è un servizio non ufficiale basato su PGP, fornito gratuitamente agli utenti di Internet da I. T. Consultancy Limited previa accettazione dei termini riportati sulla pagina Web del sito www.itconsult.co.uk/stamper-files/index.htm. L'utente invia il messaggio di cui vuole ottenere la certificazione all'indirizzo di email di Stamper. Tale messaggio viene corredato di numero di serie, firma effettuata mediante PGP e rimandato al mittente. Ogni documento firmato da Stamper ha un numero seriale unico. Questo numero viene automaticamente incrementato ogni volta che un documento viene firmato.

Ogni firma viene conservata da Stamper e puó essere ispezionata da chiunque. Gli unici dettagli che sono rivelati, sono la firma distaccata dal documento, il numero di serie e la data e l'ora in cui è stato creato il certificato. Per mantenere la privacy, infatti, nessun messaggio di testo o indirizzo di email è reso noto.

Ogni utente puó verificare la firma del documento usando PGP con la chiave pubblica di Stamper. Le chiavi pubbliche utilizzate da Stamper sono di 2046 bit e richiedono versioni di PGP recenti.

La combinazione dei numeri seriali, che sono stati collegati in ordine cronologico pubblicandone la firma, fornirá sufficienti garanzie che i timestamp non saranno né predatati né postdatati.

Per garantire l'onestá del servizio, Stamper genera due tipi di file giornalieri. Un file chiamato SIGYYYY.TXT contiene solo il numero di serie dell'ultima firma del giorno per ogni giorno dell'anno. Ed un file ZIP chiamato YYYYMMDD.ASC contiene tutte le firme (divise dal messaggio) effettuate nel giorno DD, del mese MM, dell'anno YYYY. Questi files sono disponibili presso il sito Web di Stamper o è possibile richiederli tramite email all'indirizzo

list@itconsult.co.uk mandando un messaggio contenente la linea: "GET STAMPER-FILES filename". La richiesta di piú files puó essere effettuata all'interno dello stesso file utilizzando linee separate. I file ZIP giornalieri, sono firmati da Stamper.
Allo scopo di aumentare la sicurezza del servizio le firme (divise dal messaggio) dei file ZIP giornalieri, vengono pubblicate nell'Usenet group comp.security.pgp.announce il Sabato di ogni settimana per i sette giorni precedenti. Gli utenti possono richiedere questo servizio inviando nel corpo del messaggio all'indirizzo list@stamper.itconsult.co.uk la stringa "JOIN STAMPER-ANNOUNCE".
Questi messaggi settimanali sono anche disponibili nei file WKYYYYMMN.TXT dove N rappresenta il numero del Sabato nel mese.

Stamper opera in diverse modalitá a seconda delle necessitá dell'utente e del tipo di file da gestire. Le modalitá attualmente disponibili sono:

Post. Questa modalitá viene utilizzata per mandare un messaggio (M) corredato di un time-stamp (NS+PGP(M) dove NS rappresenta il numero di serie, e PGP(M) la firma del messaggio effettuata mediante PGP).
Il messaggio dovrá essere indirizzato a post@stamper.itconsult.co.uk e dovrá contenere nel suo corpo una linea di indirizzo per ogni destinatario nella forma: X-Stamper-To: "Email Address". Destinatari multipli devono avere linee separate di X-Stamper-To. È anche importante che la X sia collocata nella prima colonna. Il messaggio firmato con un certificato di posta viene messo in coda per essere inviato agli indirizzi dati. Una copia del messaggio inviato verrá spedita anche al mittente. È importante notare che un tale certificato garantisce che qualcuno, identificato come utente, abbia chiesto allo Stamper di mandare quel messaggio ad una lista di destinatari. Questo non garantisce peró che il messaggio sia recapitato, né autentica in alcun modo l'utente.

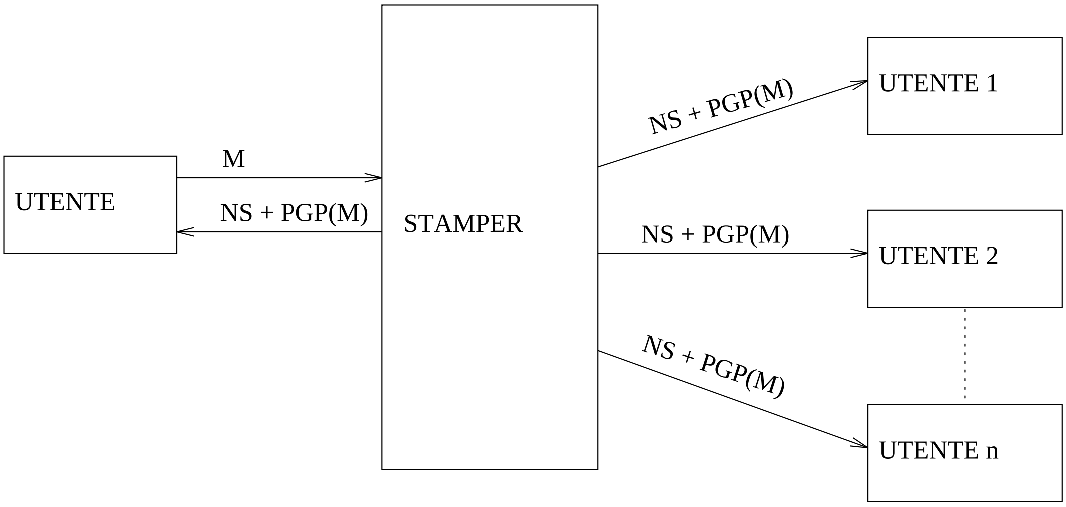


Figura 2: Modalitá Post

Clear. Questa modalitá è usata per ottenere un messaggio di testo (M), libero dalla firma (PGP(M)). Il messaggio dovrá essere inviato all'indirizzo clear@stamper.itconsult.co.uk.

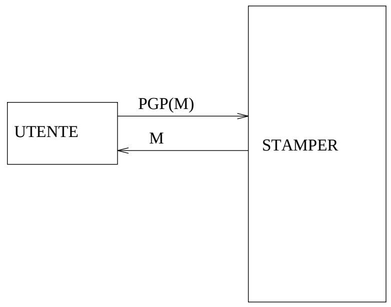


Figura 3: Modalitá Clear

Header. Questa modalitá viene usata per liberare un messaggio dall'header (numero di serie e commenti) e dalla firma. L'indirizzo a cui dovrá essere inviato è header@stamper.itconsult.co.uk. In questo caso è importante sapere che l'header del messaggio puó essere contraffatto.

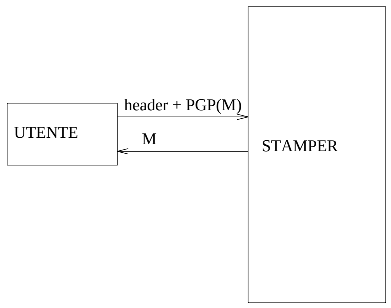


Figura 4: Modalitá Header

Text. Questa modalitá viene usata per ottenere una firma e una copia in formato ASCII (PGP(M.txt)) del messaggio di testo spedito (M.txt). L'indirizzo a cui dovrá essere inviato è text@stamper.itconsult.co.uk.

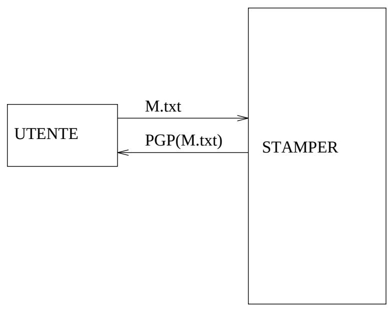


Figura 5: Modalitá Text

PGP. Questa modalitá è usata per ottenere la firma (PGP(PGP(M)) di un file esistente firmato mediante PGP (PGP(M)). Tipicamente viene utilizzata per ottenere una firma "composta" su un documento giá autenticato da Stamper o su un certificato di firma distaccato. Il file ASCII (firmato mediante PGP) dovrá essere inviato a pgp@stamper.itconsult.co.uk.

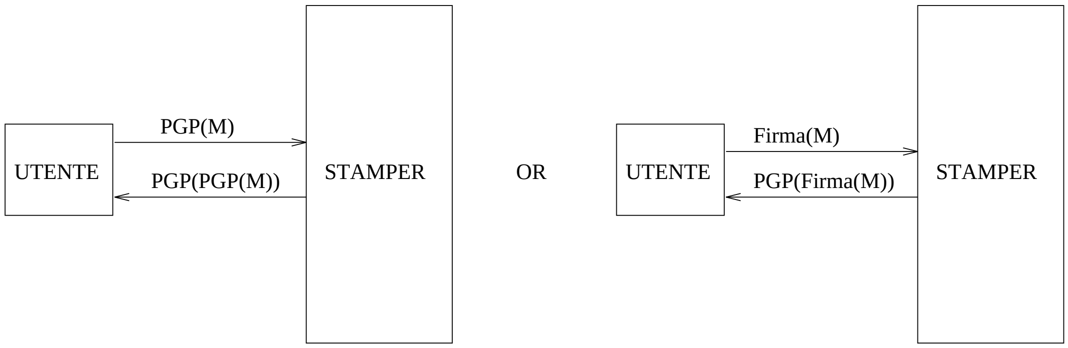


Figura 6: Modalitá del PGP

Binary. Questa modalitá viene utilizzata per ottenere un file in formato binario firmato (PGP(M.bin)). Il file dovrá essere inviato all'indirizzo binary@stamper.itconsult.co.uk.
La sola differenza tra questo e la modalitá PGP è che Stamper assume che il file ricevuto non è firmato mediante PGP (M.bin).

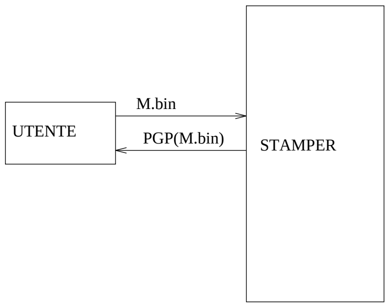


Figura 7: Modalitá Binary

Riferimenti

[1] D.Bayer, S.Haber e W.S. Stornetta, *Improving the Efficiency and Reliability of Digital Time-Stamping.*

[2] B.Den Boer e A.Bossalaers, *An attack on the last two rounds of MD4*, CRYPTO 1991.

[3] D.Chaum, E.Van Heijst e B. Pfitzmann, *Cryptographically strong undeniable signatures, unconditionally secure for the signer*, CRYPTO 1991.

[4] I.Damgard, *A design principle for hash functions*, CRYPTO 1989.

[5] Hans Dobbertin, *MD4 is not collisionfree*, Settembre 1995.

[6] FIPS PUB 180-1 *Secure Hash Standard*, Aprile 1995.

[7] S.Haber e W.S. Stornetta, *How to time stamp a digital document*, Journal of Cryptology, N.2 , pagg.99-111, 1991, Presentato anche a CRYPTO '90.

[8] B.Preneel, R.Govaerts e J.Vandewalle, *Hash functions based on block ciphers: a synthetic approach*, CRYPTO 1993.

[9] Rivest, R., *The MD4 Message Digest Algorithm*, RFC 1320, Aprile 1992.

[10] Rivest R., *The MD5 Message Digest Algorithm*, RFC 1321, Aprile 1992.

[11] la documentazione relativa è disponibile al seguente indirizzo *http://WWW.Surety.com.*