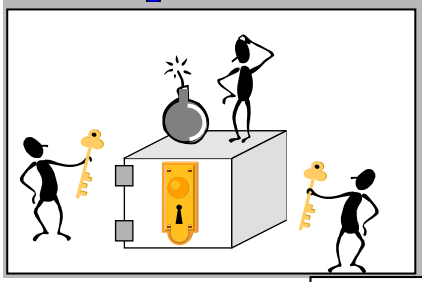
Capitolo 2



Dati sicuri

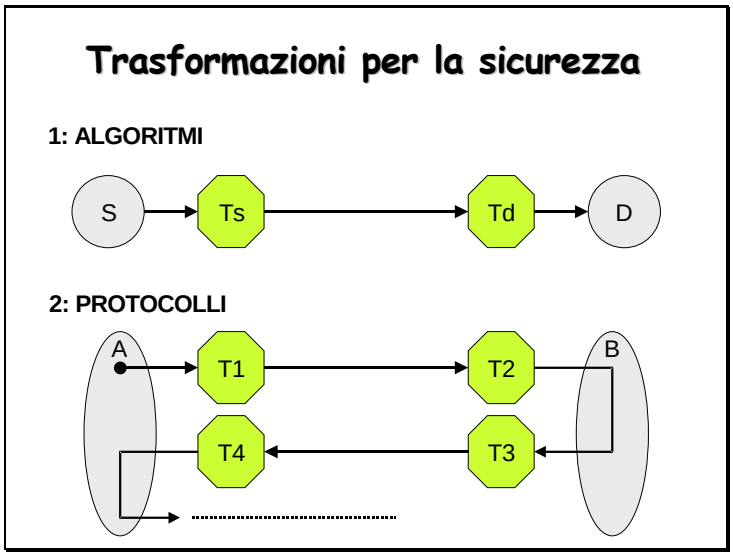
2.1 Algoritmi e protocolli

In questo corso intendiamo fondamentalmente occuparci di **sicurezza dei dati** di un sistema informatico sottoposto ad attacchi intenzionali.

Assumiamo dunque come ipotesi che i diversi componenti hardware e software del sistema sono stati in precedenza resi **sicuri** dall'adozione di tutti gli accorgimenti necessari a garantirne un corretto funzionamento.

Abbiamo già detto che questo contesto è necessario, ma non sufficiente per la **sicurezza**: tra due blocchi sicuri è, infatti, in generale interposto un canale insicuro su cui le loro comunicazioni potrebbero essere esposte agli attacchi passivi ed attivi dell'intruso.

Tale vulnerabilità può essere difesa da una contromisura in linea di principio molto semplice:
"sorgente e destinazione, in perfetto accordo, attribuiscono alle informazioni comunicate sul canale insicuro una rappresentazione nota soltanto a loro ed atta ad impedire all'intruso di comprendere il significato della comunicazione (riservatezza) e/o di spacciare come inviati dalla sorgente messaggi che lui ha modificato (integrità) o forgiato (autenticità).



A tal fine si deve predisporre un blocco di elaborazione (la forma ottagonale evidenzierà da ora in poi un meccanismo per la sicurezza) a ciascuna estremità del canale, adibendo **Ts** a trasformare una rappresentazione standard dell'informazione nella nuova rappresentazione e **Td** a ripristinarla dopo aver verificato che quella usata sul canale è congruente con gli accordi presi dai due corrispondenti.

Tali blocchi eseguono dunque funzioni definite da **algoritmi**, che tra poco vedremo dover essere dotati di particolari proprietà.

In generale questo scenario non è però sempre sufficiente. Per prendere accordi segreti sulle trasformazioni da eseguire o per fronteggiare efficacemente certi attacchi, i due corrispondenti, alternandosi nei ruoli di sorgente e di destinazione, devono spesso essere chiamati ad eseguire in sequenza ordinata più azioni secondo le indicazioni fornite da un certo **protocollo**.

Lo scenario non è ancora completo. Nella Società dell'informazione occorre, infatti, prevedere anche il caso che il malintenzionato sia uno dei due corrispondenti. Anticipiamo qui, ma lo vedremo meglio più avanti, che per fronteggiare questa minaccia è necessario il coinvolgimento nel protocollo di una **terza parte fidata**, attribuendole il compito di intervenire nel corso (l'**arbitro**) o al termine della sua esecuzione (il **giudice**).

La disciplina che studia gli **algoritmi** ed i **protocolli** da svolgere alle estremità di un canale insicuro è detta **Crittologia**. Obiettivo fondamentale della Crittologia è l'individuazione di trasformazioni efficienti, efficaci e sicure; a tal fine la Crittologia è formata da due distinte e correlate discipline:

- la **Crittografia** individua le trasformazioni idonee a proteggere una o più proprietà critiche dell'informazione;
- la **Crittanalisi** ne valuta la **robustezza**, esaminando **come**, **in quanto tempo** e **con quali risorse** è possibile **rompere** le difese.

L'avvento dei calcolatori elettronici ha segnato un punto di netta demarcazione nell'evoluzione di queste discipline: la denominazione di Crittologia **classica** si riferisce a tutto quello che è stato fatto prima, quella di Crittologia **moderna** a tutto quello che è stato fatto dopo.

La **Crittologia classica** si è preoccupata solo della riservatezza ed è stata in realtà un'arte coltivata da poche persone. L'uso di "carta e matita" o poco di più ha fortemente limitato la complessità degli algoritmi crittografici ed i metodi di crittanalisi hanno di conseguenza sempre vinto la partita. Istruttivo è comunque conoscere gli uni e gli altri, cosa che faremo nel capitolo 3.

L'enorme diffusione dei calcolatori e delle reti di calcolatori ha oggi posto al centro degli interessi di tutti il tema della sicurezza dei dati. Con i calcolatori è stato possibile mettere a punto difese di notevole complessità. L'uso dei calcolatori ha però anche consentito di mettere in atto attacchi di pari complessità.

Sotto questa spinta la **Crittologia moderna** è diventata una vera e propria Scienza, fortemente connessa alla Matematica, alla Statistica, alla Teoria degli Algoritmi ed alla Teoria dell'Informazione.

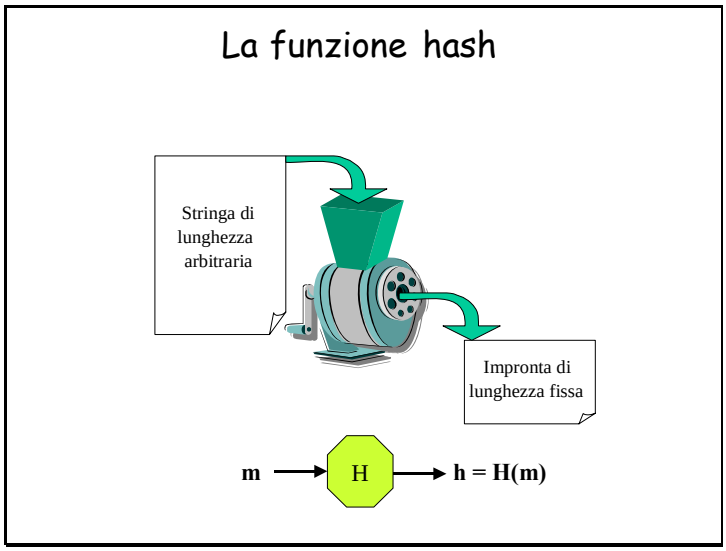
2.1.1 Accertamento dell'integrità

I disturbi presenti sui supporti fisici di comunicazione e di memorizzazione minacciano l'integrità dell'informazione che su essi fluisce. Errori possibili sono l'inserzione, la cancellazione e la modifica di valore di uno o più bit della stringa. Abbiamo però già evidenziato (v. pag. 4) che, una volta ipotizzati i disturbi più probabili, è possibile rilevare gli errori da essi indotti con una codifica ridondante dell'informazione.

Un attacco attivo presenta le stesse minacce all'integrità dell'informazione e vi aggiunge quella dell'alterazione dell'ordine dei bit nella stringa. La grossa differenza rispetto ai disturbi è che in questo caso non esistono modifiche più probabili di altre; ciononostante è ancora possibile impostare la difesa sulla ridondanza.

- R2: "*la sorgente affianca al messaggio un "riassunto" che ne rappresenti in modo univoco il contenuto; la destinazione calcola il riassunto del messaggio ricevuto e lo confronta con quello inviato dalla sorgente* ".

Per la sicurezza di tale accorgimento è necessario che **ogni modifica** apportata al messaggio originale, anche ad uno solo dei suoi bit, produca alla destinazione **un riassunto diverso** da quello inviato dalla sorgente.



La Crittografia impiega a tal fine una particolare funzione, detta **hash**, che comprime una stringa **m** di lunghezza arbitraria in una stringa **h** di lunghezza piccola e prefissata:

h = H(m).

L'uscita di una funzione hash è detta **riassunto** o **impronta** (*digest*, *fingerprint*) del messaggio d'ingresso. Un'impronta di **n** bit (tipicamente nel *range* 128 ÷ 512) suddivide l'insieme di tutte le possibili stringhe d'ingresso in $\mathbf{2^n}$ sottoinsiemi disgiunti, formati ciascuno da tutte e sole le stringhe che forniscono uguale **h**.

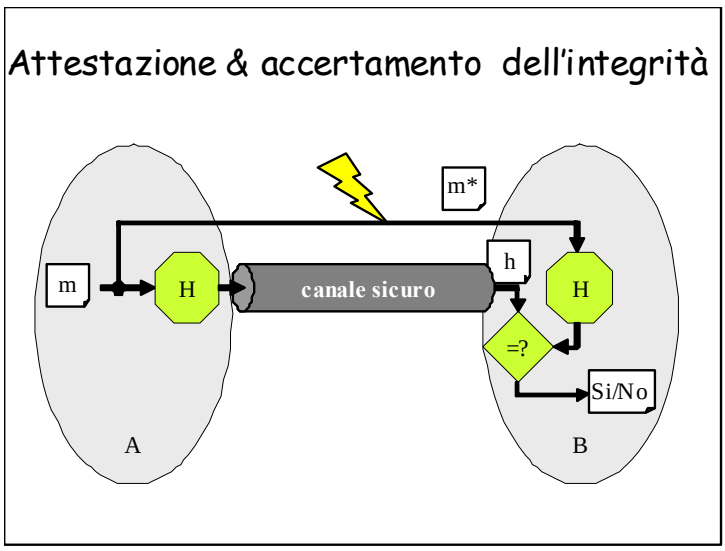
Due stringhe che hanno lo stesso hash sono dette essere in **collisione**.

Una funzione hash è detta **semplice** se l'individuazione di collisioni è un calcolo **facile**. A funzioni di questo tipo è usualmente affidato il calcolo dei bit di controllo dei codici che consentono la rilevazione d'errori indotti da disturbi aleatori.

Una funzione hash è detta **sicura**, o **crittografica**, se il suo comportamento è apparentemente **aleatorio**.

Il modello a cui ci si fa riferimento è detto "**oracolo casuale**" e prevede che fornendo in ingresso un messaggio di cui non si conosce ancora l'impronta, si riscontra sull'uscita, **con uguale probabilità,** uno qualsiasi dei 2^n valori possibili. Gli algoritmi che approssimano tale comportamento impiegano un **n** "grande" per rendere estremamente improbabile che messaggi diversi abbiano la stessa impronta e rispettano la seguente regola.

- R3 (resistenza alle collisioni): "*l'individuazione di due messaggi con la stessa impronta è un calcolo difficile*".



In figura è indicato un tipico caso d'impiego di una funzione hash sicura per l'accertamento dell'integrità di un'informazione: una sorgente **A** trasmette ad una destinazione **B** un messaggio **m** su un canale insicuro ed un riassunto **h = H(m),** tramite un canale su cui l'intruso non può svolgere (o è stato messo in condizione di non poter svolgere) attacchi attivi.

B riceve dunque un certo **m*** e **h**.

Per garantirsi che il messaggio ricevuto non sia stato alterato, è sufficiente che **B** calcoli **H(m*)** e lo confronti con **H(m)**.

Stante l'impossibilità per l'intruso di forgiare un messaggio di suo interesse ed in collisione con **m**, l'eguaglianza **H(m) = H(m*)** consente a **B** di concludere che il messaggio ricevuto è integro.

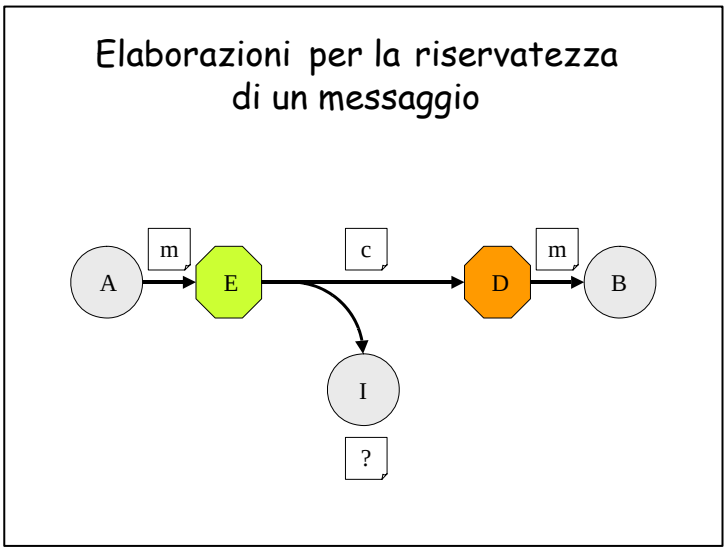
ESEMPI – Spesso i **Produttori di software** forniscono le impronte dei programmi che distribuiscono (inserendole ad es. sul contenitore del prodotto e/o nel manuale d'uso e/o in una loro pagina web). Dati di questo tipo consentono in un secondo tempo agli utenti di verificare l'integrità di ciò che stanno per installare o per impiegare.

Nel sistema di posta elettronica sicura **PGP**, ogni utente deve avere in memoria una copia autentica di un dato di ogni suo corrispondente (la sua **chiave pubblica**); il controllo dell'integrità di questo dato, formato da 1000-2000 bit, può essere ottenuto facendosi comunicare per telefono la sua sola impronta, a patto che la funzione hash impiegata sia sicura e che la voce costituisca un sicuro elemento di identificazione.

2.1.2 Difesa della riservatezza

La regola da rispettare per **comunicare in modo riservato** è molto semplice.

- R4: "*la sorgente deve trasformare la rappresentazione delle informazioni riservate in modo da renderle incomprensibili per chiunque, eccezion fatta per la destinazione*".



La trasformazione **E** operata dall'utente A, detta **cifratura** (*encryption*), deve essere **facile** da eseguire. I bit della stringa che rappresenta l'informazione **m** sono trasformati o uno per volta (in serie) o più alla volta (in serie/parallelo): ciò consente di elaborare messaggi con lunghezza arbitraria.

La trasformazione **D** operata dall'utente B, anch'essa **facile** da eseguire, è detta **decifrazione** (*decryption*) e deve essere l'**inversa** di **E**. I bit della stringa **c** sono ritrasformati o uno alla volta o più alla volta.

Una coppia di trasformazioni **E**, **D** è detta **cifrario**. La forma originaria del messaggio **m** è detta **testo in chiaro** (*plaintext*). La forma **c** con cui **m** è trasferito sul canale insicuro è detta **testo cifrato** (*ciphertext*).

Alcune osservazioni sull'uso di un cifrario:

- i due partecipanti non devono essere necessariamente *on-line*: in ricezione i messaggi cifrati possono, infatti, essere prima memorizzati e poi decifrati;
- uno stesso utente può essere sia sorgente, che destinazione: questo caso interessa chi vuole realizzare un proprio archivio di documenti riservati.

La minaccia alla riservatezza è, come sappiamo, l'intercettazione. La difesa, preventiva, è mettere l'intruso I nella condizione di non riuscire a fare ciò che fa la destinazione B. A tal fine la modalità di decifrazione deve essere tenuta **segreta**, ma ciò non è sufficiente. L'intruso potrebbe, infatti, rimettere in chiaro il testo cifrato o, tanto peggio, individuare il segreto sulla trasformazione avvalendosi di conoscenze già in suo possesso (probabili motivi della comunicazione, probabile contenuto, ecc.) e di testi cifrati intercettati in precedenza.

Un cifrario è detto **perfettamente sicuro** se l'intruso **non può imparare da un testo cifrato nulla di più di quello che sapeva già prima di intercettarlo**. Nel paragrafo 3.4 vedremo che questa condizione è ottenibile, ma molto pesante da realizzare. In pratica bisogna dunque accontentarsi del rispetto della seguente regola.

- R5: "*i calcoli per mettere in chiaro un testo cifrato senza conoscere la modalità di decifrazione devono essere **difficili** da eseguire*".

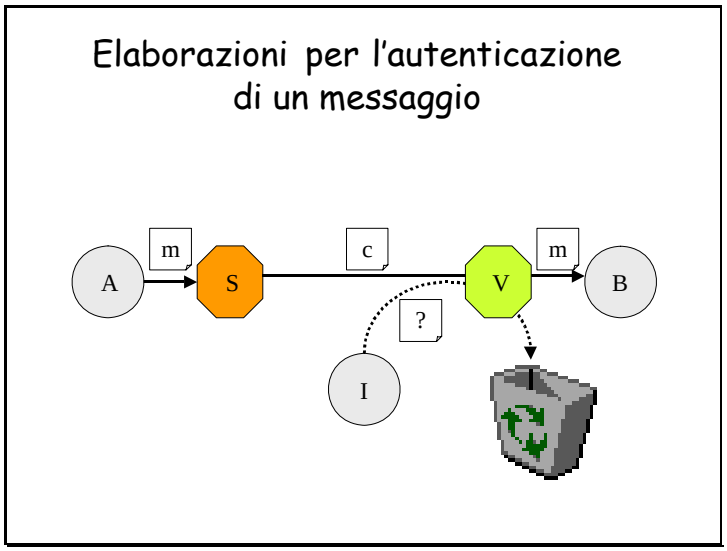
Se sul canale sono presenti disturbi e/o sono possibili attacchi attivi, il testo cifrato ricevuto dal destinatario (indichiamolo con **c***) può essere diverso da quello inviato dal mittente. Per difendere anche l'**integrità** del messaggio riservato **m** si può ricorrere al seguente protocollo:

1.	**p = m||H(m)**	*A forma un nuovo testo in chiaro p "concatenando" m e H(m)*
2.	**c = E(p)**	*A cifra p ed invia a B il risultato c*
3.	**p* = D(c*) = m*, H*(m)**	*B decifra c*, ottiene p* e separa le due componenti*
4.	**H(m*) =? H*(m)**	*B calcola l'impronta di m* e la confronta con l'impronta ricevuta: se sono uguali il messaggio è giudicato integro*

2.1.3 Accertamento dell'autenticità

E' semplice anche la regola che consente di verificare **l'origine** di un documento.

- R6: "*la sorgente aggiunge al documento informazioni non imitabili ed atte ad attestare chi l'ha predisposto; la destinazione verifica che il documento ricevuto sia stato originato proprio da chi dichiara di averlo fatto* ".



La trasformazione **S** operata dalla sorgente A, detta **prova** di autenticità del messaggio in chiaro **m**, deve essere **facile** da eseguire.

Non devono inoltre esistere vincoli sulla lunghezza della stringa che rappresenta l'informazione; a tal fine i bit di **m** possono venir prima suddivisi in "blocchetti" e poi elaborati.

La trasformazione **V** operata dalla destinazione B, detta **verifica** dell'autenticità, opera in sequenza sui "blocchetti" del testo cifrato **c** e deve essere **facile** da eseguire.

Il risultato è duplice: al termine della verifica la destinazione deve, infatti, poter disporre sia di **m**, sia dell'indicazione **SI/NO** sulla sua autenticità per poter "cestinare" tutti i messaggi che **V** ha classificato come falsi.

Alcune osservazioni sull'uso:

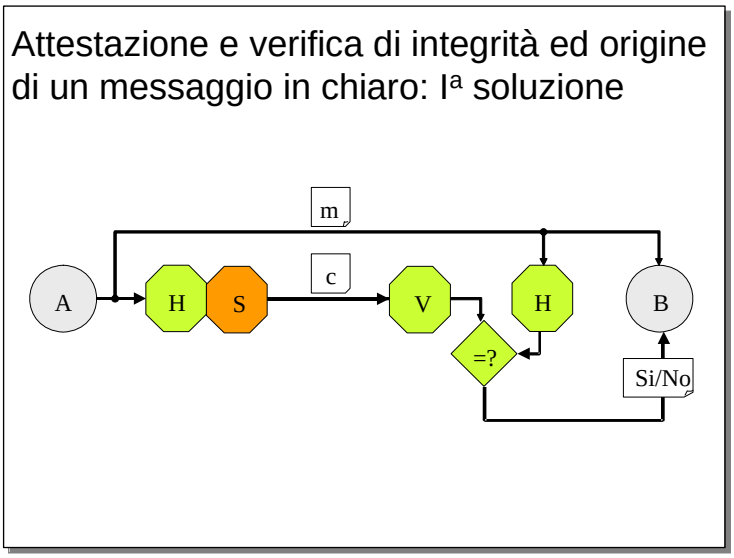
- non è necessario che i due partecipanti siano contemporaneamente attivi: i messaggi autenticati possono, infatti, essere prima memorizzati e poi verificati;
- uno stesso utente può essere sia l'autore, che il verificatore della prova di autenticità: questo caso interessa chi vuole scoprire se i suoi dati sono stati o no alterati.

La minaccia è che l'intruso **I** riesca a far accettare alla destinazione come originati dalla sorgente A messaggi che ha intercettato e modificato o che si è creato per conto suo.

La prima prevenzione è che la trasformazione **S** sia tenuta **segreta**; la seconda è che **I** non possa dedurla da testi cifrati intercettati e da altre sue conoscenze.

- R7: "*i calcoli per costruire un messaggio apparentemente autentico senza disporre della trasformazione operata dalla sorgente devono essere* ***difficili*** *da eseguire*".

Spesso è necessario trasmettere in chiaro il messaggio **m**. Per consentire a B di accertare contemporaneamente l'**integrità** e l'**origine** del messaggio ricevuto, la Crittografia ha messo a punto due differenti soluzioni.

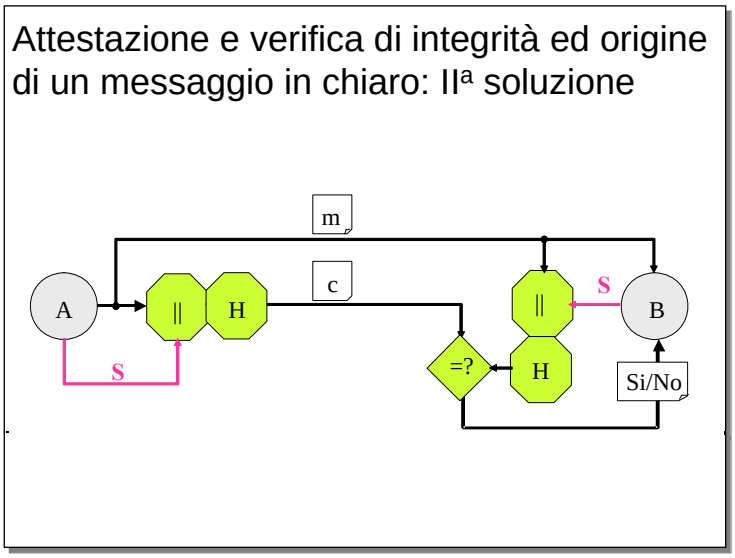


La prima soluzione è il cosiddetto **schema di firma digitale**, in cui il **messaggio in chiaro** è affiancato dalla **autenticazione del suo hash**.

La sorgente calcola **H(m)** come prova di integrità e se ne assume poi la paternità trasformando il risultato con il meccanismo **S**, che è l'unica a conoscere.

La destinazione, con il meccanismo **V**, noto a tutti, mette in chiaro il testo cifrato **c** e lo confronta con il calcolo dell'impronta del messaggio ricevuto. In questa maniera rileva ogni alterazione apportata a **m** e/o a **c**.

La minaccia è che l'intruso riesca ad alterare **m** mantenendo la compatibilità con **H(m)**. L'insuccesso dell'attacco è garantito se si impiega una funzione hash sicura, cioè atta a rendere impossibile il calcolo di una collisione.



Una seconda soluzione del problema si basa sul fatto che i due corrispondenti abbiano precedentemente **concordato in segreto un dato s**. Per consentire la verifica dell'integrità e dell'origine di un **m,** A e B eseguono il seguente protocollo:
A: calcola **H(m||s)**
A → B: c = m||H(m||s)
B: riceve **c*** ed ottiene **m*, H*(m||s)**
B: calcola **H(m*||s)**
Se e solo se **H(m*||s)=H*(m||s)**, B considera **m** integro ed originato da A.

In questa soluzione è necessario che la funzione H sia **difficile** da invertire: se così non fosse, infatti, il segreto **s** potrebbe diventare noto anche ad un malintenzionato che vuole spacciarsi per A o per B.

Dal punto di vista della sicurezza le due soluzioni **non** sono equivalenti. Nel primo caso, infatti, A non può negare di essere stato proprio lui ad autenticare m, essendo l'unico in grado di eseguire la trasformazione S: per l'informazione **m** la cui origine è stata accertata dalla corrispondente trasformazione V vale dunque la proprietà di **non ripudiabilità** (maggiori dettagli li daremo nel cap. 5). Nel secondo caso invece la generazione di **m** ed il calcolo di **H(m||s)** possono essere fatti anche da B all'insaputa di A ed un'eventuale disputa tra i due non può essere risolta. La soluzione è quindi utile solo se i due corrispondenti si fidano uno dell'altro.

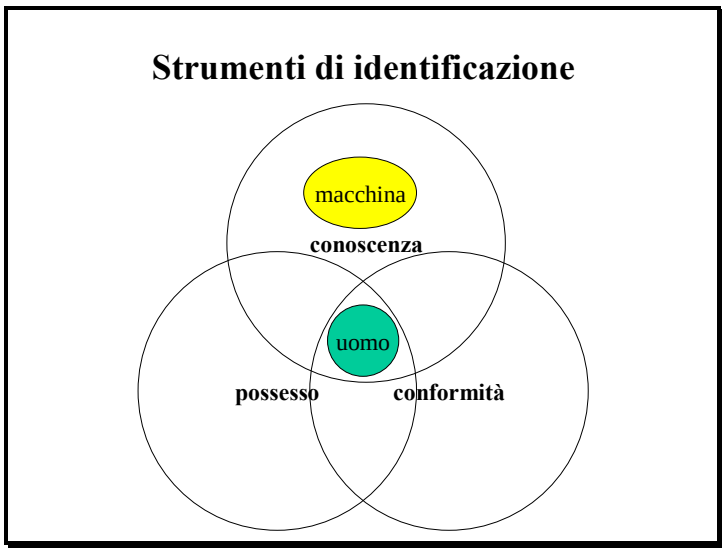
2.1.4 Accertamento dell'identità

Esistono applicazioni che ammettono l'anonimato (informativa su servizi pubblici, posta elettronica) ed altre che lo presuppongono (voto elettronico, moneta elettronica). Nella maggioranza dei casi è però richiesto che l'erogatore di un servizio conosca l'identità di chi ne usufruisce (tipicamente per delimitare e per fatturare l'uso che ne viene fatto) e che il fruitore sia certo dell'identità dell'erogatore (per non cadere ingenuamente in una truffa).

Il **processo d'identificazione** è necessariamente *real-time*: si deve, infatti, riferire ad un ben preciso istante di tempo ed entrambi gli interessati devono quindi essere *on line*. Proprietà indispensabili sono anche l'efficienza (il dichiarare la propria identità deve essere un'operazione semplice e veloce) e l'efficacia (deve essere bassissima la probabilità di scambiare un utente autorizzato per uno che non lo è ed un impostore per un utente autorizzato).

Le Entità coinvolte possono essere: due utenti remoti, un utente ed una macchina locale o remota, due macchine remote. Per un'identificazione sicura vale la seguente regola.

- R8: "*tramite il solo scambio di messaggi, l'Entità che vuole farsi riconoscere deve fornire informazioni non imitabili, atte ad individuarla univocamente in quel preciso istante di tempo, e l'Entità che effettua il riconoscimento deve potersi convincere della loro genuinità* ".

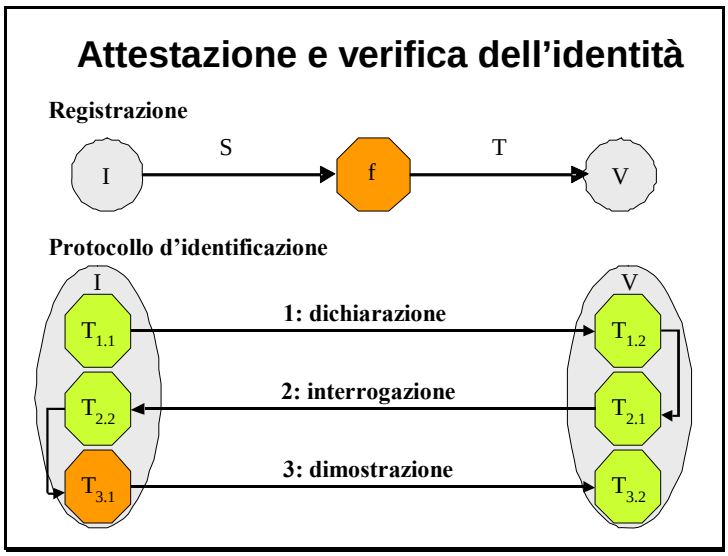


Tre sono in generale le dimostrazioni d'identità (*factors*) che una persona può fornire.
1. La **conoscenza di un'informazione** che ha in precedenza concordata (*password*).
2. Il **possesso di un oggetto** (*token*) datogli in consegna (una scheda a banda magnetica, una smart card, ecc.).
3. La **conformità di una misura biometrica** raccolta in quel momento con una misura registrata in precedenza; oggetto di misura è una caratteristica individuale, o fisiologica (l'impronta digitale, la forma della mano, la forma del volto, il timbro della voce, la forma dell'iride, la venatura della retina), o comportamentale (la dinamica della firma).

N.B. L'identificazione di una macchina si basa ovviamente solo sulla conoscenza.

Due sono i momenti topici, o fasi, di un processo d'identificazione:

- la **registrazione,**
- il **riconoscimento**.



Durante la **registrazione**, l'**identificando I** ed il **verificatore V** concordano e memorizzano rispettivamente il **dato segreto S** con cui **I** si farà riconoscere ed il **termine di paragone T=f(S)** che consentirà a V di accertare che I conosce **S**.

Quando successivamente sarà necessario procedere al **riconoscimento**, le due Entità sono chiamate a svolgere un **protocollo crittografico**.

In figura è evidenziata una semplice articolazione in tre passi, tutti **facili** da eseguire:

1. l'identificando dichiara chi è;
2. il verificatore gli chiede di dimostrarlo;
3. l'identificando fornisce la prova; se è quella concordata, il verificatore lo identifica.

La sicurezza è minacciata dal fatto che l'intruso possa spacciarsi per un utente legittimo sfruttando una prova d'identità che è riuscito o a **dedurre** da intercettazioni, o ad **indovinare**, o a **rubare** dalla memoria di uno dei due partecipanti al protocollo.

Il rischio è alto. Abbiamo già detto (v. pag. 11) che l'identificazione sta alla base di ogni politica di controllo d'accesso: se l'attacco ha successo, l'intruso ottiene tutti i diritti di un utente legittimo.

Il primo accorgimento difensivo è la non invertibilità della trasformazione **f** impiegata durante la fase di registrazione. Il secondo è la segretezza della trasformazione $T_{3.1}$ che fornisce la prova d'identità. Il terzo è espresso dalla seguente regola:

- R9: "*i calcoli che consentono di individuare una prova d'identità valida senza conoscere la trasformazione che la genera devono essere* ***difficili*** *da eseguire*".

2.1.5 Complessità computazionale di un algoritmo

Finora abbiamo usato i termini di **calcolo facile** e di **calcolo difficile** con il loro significato più intuitivo. Ciò ci ha consento di differenziare nettamente le attività computazionali richieste a chi usa legittimamente un sistema informatico sicuro, da quelle che deve invece mettere in atto chi vuole attaccarne la sicurezza.

E' però utile avere a disposizione anche una definizione rigorosa di tali termini: a tal fine si deve far riferimento alla Teoria degli algoritmi e, più in particolare, a quella branca denominata **Teoria della complessità computazionale**[6]. Il seguente glossario richiama alcune importanti definizioni.

Algoritmo	Procedimento che risolve un ben definito problema computazionale, prendendo un insieme di valori come dato di *input* e producendo un insieme di valori come *output*.
Dimensione dell'input	Numero **n** di bit che rappresentano il dato d'ingresso (*input size*).
Complessità computazionale di un algoritmo	Quantità di risorse di calcolo necessarie per eseguirlo: risorse tipicamente prese in considerazione sono il tempo di esecuzione *(running time)*, la capacità di memoria *(memory size)*, il numero e la dimensione degli stack.
Tempo di esecuzione di un algoritmo	Numero **N** di operazioni elementari, o di passi, che l'algoritmo deve eseguire per terminare. In generale **N** dipende dalla dimensione **n** dell'input: in alcuni casi **N = f(n)** è esprimibile analiticamente, in altri no. A parità di **n**, **N** dipende anche dal valore di input: per non tenerne conto si usa solo il **più grande** valore di **N** (*worst case*).
Ordine di grandezza del tempo di esecuzione	Modalità di incremento di **N** all'aumentare senza limiti di **n**. Se **f(n)** è espressa da una formula**,** si considera soltanto il termine **g(n)** che cresce più rapidamente con **n**; in caso contrario si considera una funzione **g(n)** ed una costante **c** tale che $\mathbf{0 \le f(n) \le c.\ g(n)}$ per ogni $\mathbf{n \ge n_0}$

[6] vedi S. Martello: "Lezioni di Ricerca Operativa" Esculapio 2001 o T.H. Cormen, C.E. Leiserson, R.L. Rivest "Introduzione agli algoritmi", vol.1, cap.2, *Jackson Libri* 1994

T = O(g(n))	Notazione simbolica dell'ordine di grandezza del tempo di esecuzione.
Algoritmo (con tempo) polinomiale	Algoritmo con **g(n)** polinomiale in **n.** L'ordine di grandezza del tempo di esecuzione è indicato con **T = O(n^t)**, ove **t** è l'esponente più grande presente nel polinomio.
Algoritmo (con tempo) esponenziale	Algoritmo con **g(n)** esponenziale in **n**. L'ordine di grandezza del tempo di esecuzione è indicato con **T = O(exp (n))**.
Problema facile	Problema, detto anche di tipo **P**, per il quale esiste un algoritmo polinomiale in grado di risolverlo su una **macchina di Turing deterministica**.
Problema difficile	Problema per cui non sono stati fino ad ora individuati, e probabilmente non saranno mai individuati, algoritmi di risoluzione con tempo polinomiale.

Per familiarizzarci con le notazioni della teoria della complessità computazionale, vediamo qualche semplice esempio di valutazione del tempo di esecuzione.

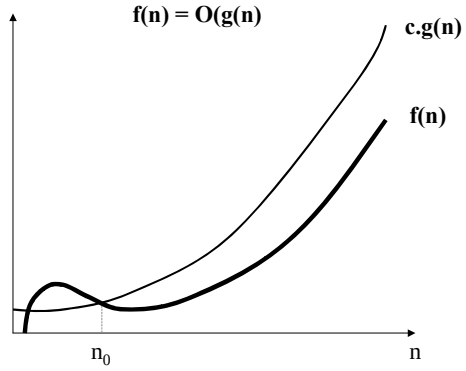
ESEMPI - Siano X e Y due numeri binari di n bit e sia p < n il n° di bit che l'unità aritmetico-logica di un calcolatore sa trattare contemporaneamente.
Addizione – Il calcolo di X+Y richiede di sommare i p bit meno significativi di X,Y, di memorizzare il riporto, di sommarlo con i p bit di X,Y di peso immediatamente superiore e di procedere nella stessa maniera fino a quando non sono stati elaborati anche i p bit di maggior peso. Tale algoritmo esegue dunque n/p operazioni elementari e la sua complessità è quindi espressa da **O(n)**.
Moltiplicazione – Il calcolo di XY può essere eseguito con l'algoritmo che prevede di sommare n prodotti parziali. Ogni prodotto parziale richiede di ripetere n/p volte il prodotto logico tra un bit di Y e p bit di X e di far scorrere verso sinistra l'intera stringa risultante di tante posizioni quant'è il peso del bit di Y preso in considerazione: per eseguirlo occorrono 2n/p operazioni della ALU. La complessità computazionale del prodotto è quindi espressa da **O(n^2).**
Esponenziazione – X^Y è per definizione uguale a (((X)X)X).. per Y volte. Nel caso peggiore Y è uguale a 2^n ed altrettante sono quindi le moltiplicazioni da eseguire: tale algoritmo ha dunque un tempo di esecuzione esponenziale.

Il problema della esponenziazione non è per questo difficile: sfruttando la rappresentazione binaria di Y, è infatti, possibile ottenere algoritmi con tempo di esecuzione polinomiale. La formula

$$X^Y = \prod_{i=0}^{n-1} X^{2^i y_i}$$

indica che occorre prima calcolare, con n moltiplicazioni, i fattori X, X^2=X X, X^4= X^2 X^2, X^8= X^4 X^4 ··· ecc. e poi, nel caso peggiore, moltiplicare tra loro tutti questi risultati parziali (e quindi ancora con n moltiplicazioni): **O(n^3)**.

La notazione del "grande O"



Prima di impiegare la notazione del "grande o" nel contesto della sicurezza informatica occorre un momento di riflessione.

La valutazione del tempo di esecuzione nel **caso peggiore** è limitante, quando si studiano algoritmi di attacco: occorre, infatti, sapersi difendere dal **caso migliore** (quello più agevole per l'intruso) ed è quindi necessario adottare ogni provvedimento atto ad impedire che l'avversario si trovi di fronte ad **istanze del problema** di facile soluzione.

La notazione del "grande o" si limita inoltre ad evidenziare solo come si incrementa il tempo di esecuzione dell'algoritmo al crescere senza limiti della dimensione dell'input.

Per ottenere sicurezza è dunque necessario saper individuare il valore di **n** al di sopra del quale il calcolo diventa impossibile.

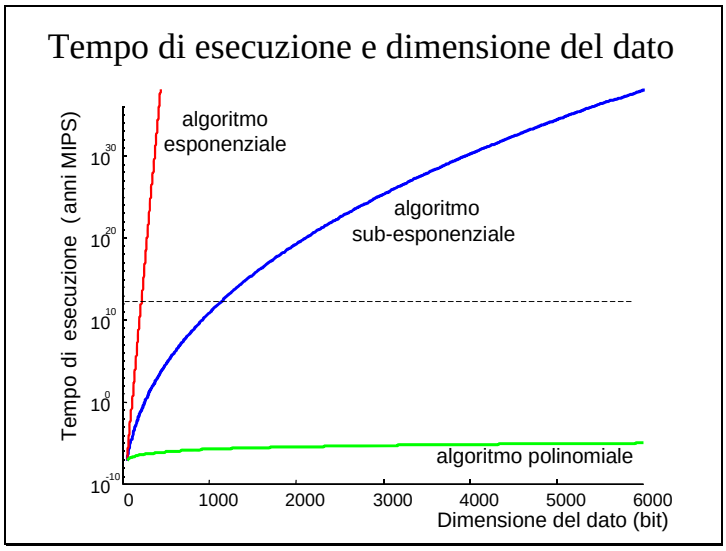
Una volta chiariti i limiti, è giusto evidenziare il contributo positivo che la Teoria della complessità può dare alla valutazione della sicurezza di un meccanismo crittografico. La prima cosa da ipotizzare è che utenti ed intrusi siano in grado di eseguire soltanto **algoritmi polinomiali.** Stabilito questo, sono sufficienti due sole regole.

- R10: "*ogni algoritmo che difende una proprietà critica dell'informazione deve avere tempo **polinomiale***".

ESEMPI - Le notazioni **O(1)**, **O(n)**, **O(n^3)** caratterizzano algoritmi polinomiali con crescente tempo d'esecuzione

- R11: "*ogni algoritmo che attacca una proprietà critica dell'informazione deve avere tempo* ***esponenziale***".

ESEMPI - **O(exp (n))**, **O(exp ($n^{1/2}$))** caratterizzano algoritmi esponenziali con tempo d'esecuzione decrescente



E' bene mettere fin d'ora in evidenza che la Crittanalisi considera **ineseguibili** anche algoritmi di attacco, detti **sub-esponenziali**, il cui tempo d'esecuzione ha una componente polinomiale ed una esponenziale:

O(exp ($(n)^{\alpha}$ $(\ln(n)^{1-\alpha})$) con $0 < \alpha < 1$.

A parità di dimensione dell'input, la loro difficoltà di calcolo è decisamente più bassa di quella di un algoritmo esponenziale, pur restando incomparabilmente più alta di quella di un algoritmo polinomiale.

Le curve di figura intendono dare un'idea qualitativa sulle modalità di crescita del tempo di esecuzione degli algoritmi polinomiali, sub-esponenziali ed esponenziali.

Anni MIPS

Per individuare la dimensione di input che determina l'**impossibilità computazionale** degli attacchi, le tecnologie per la sicurezza hanno assunto come riferimento un calcolatore in grado di eseguire **un milione di operazioni al secondo** o, più brevemente, **MIPS**.

Il tempo di esecuzione di un attacco è di conseguenza espresso in **anni MIPS** ed indica quindi il numero di anni impiegati dal calcolatore di riferimento per portarlo a termine.

Negli anni '90 si è valutato che $\mathbf{10^{12}}$ anni MIPS fossero adeguati per garantire l'impossibilità di calcolo.

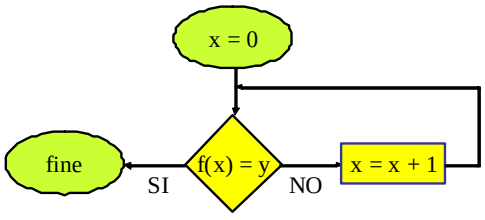
La **legge di Moore** (a parità di costo, circa ogni 18 mesi è realizzabile un calcolatore con potenza doppia del precedente) e la crescente possibilità di impiegare il calcolo parallelo (cioè di scomporre l'algoritmo in parti eseguibili contemporaneamente e di assegnare tale carico di lavoro a migliaia di macchine connesse in rete o a sistemi contenenti migliaia di processori) continueranno a spingere questo limite sempre più in alto. Il dato oggetto di un attacco dovrà di conseguenza essere rappresentato da una stringa di bit sempre più lunga.

Livelli di sicurezza

Nella valutazione della sicurezza si può però anche fare a meno degli anni MIPS, introducendo il concetto di **livello di sicurezza**. Questa diversa unità di misura della robustezza assume come riferimento il tempo d'esecuzione dell'algoritmo di **ricerca esauriente**.

L'algoritmo di ricerca esauriente

PROBLEMA – **Per quale intero x, con $0 < x < 2^n-1$, vale la proprietà f(x) = y?**



Nel caso peggiore occorrono 2^n passi: O(exp(n))

La prima cosa da capire bene è il funzionamento di tale algoritmo, cosa che possiamo fare agevolmente ponendoci il seguente problema:

"*individuare nell'intervallo* ***0 ÷ 2^n-1*** *un numero* ***x*** *tale che* ***f(x) = C*** *senza avere a disposizione un algoritmo efficiente in grado di calcolare* ***f^{-1}***".

Per trovare **x** è sufficiente generare uno dopo l'altro, a partire dal più piccolo, i numeri binari di n bit controllando ogni volta se per il valore corrente si ha **f(x) = C**: in caso affermativo l'algoritmo termina, in caso negativo si prosegue nella generazione incrementando di un'unità l'ultimo numero che non ha superato il test.

Nel caso peggiore l'algoritmo termina dopo $\mathbf{2^n}$ passi ed ha quindi complessità **O(exp(n))**.

La seconda cosa da capire è che l'algoritmo di ricerca esauriente è in grado di risolvere tutti i problemi della Crittanalisi (in questo contesto è detto **attacco con forza bruta**): una trasformazione segreta appartiene sempre ad un insieme finito di trasformazioni possibili e chi le prova tutte dispone sempre di un test per verificare se ha trovato quella buona.

La terza ed ultima cosa da osservare è che in moltissimi casi d'attacco esistono algoritmi più efficienti. Consideriamo dunque "il migliore" di questi algoritmi per un certo attacco e supponiamo di aver verificato che può essere condotto con successo eseguendo al più **N passi**.

Per misurarne la pericolosità è sufficiente individuare un **p** tale che

$$\mathbf{2^p \leq N < 2^p.}$$

ed affermare che il sistema attaccato ha **un livello di sicurezza di p bit**, cioè che è attaccabile solo con uno sforzo computazionale pari a quello richiesto dalla ricerca esauriente di un numero di p bit.

Il vantaggio di impiegare il livello di sicurezza al posto degli anni MIPS è quello di non fissare la velocità del calcolatore che esegue l'attacco e quindi di prescindere dall'evoluzione tecnologia. Fare riferimento al numero di passi piuttosto che al numero di operazioni costituisce inoltre un'utile semplificazione della valutazione di robustezza. Naturalmente è possibile passare dalla misura in anni MIPS a quella in livello di sicurezza e viceversa

ESEMPIO - $\mathbf{10^{12}}$ anni MIPS corrispondono ad un livello di sicurezza di 85 bit.

E' opinione largamente condivisa che i meccanismi progettati ed impiegati attualmente debbano avere un livello di sicurezza il più possibile vicino a **128 bit**.

2.1.6 Funzioni unidirezionali

Per ottenere sicurezza, la Crittografia chiama in causa funzioni **facili** da calcolare e **difficili** da invertire, per cui ha coniato la denominazione di **funzioni unidirezionali** (*one-way functions*).

Abbiamo già detto ad esempio:

- che la funzione hash sicura deve poter essere calcolata da un algoritmo con tempo polinomiale e che non deve esistere un algoritmo con tempo polinomiale in grado di risalire da un'impronta H(x) ad uno degli x che la generano;
- che la funzione di cifratura di un cifrario deve poter essere calcolata da un algoritmo con tempo polinomiale e che chi non conosce la funzione inversa non deve poter disporre di un altro algoritmo con tempo polinomiale in grado di risalire da E(m) a m;
- che l'algoritmo di verifica di uno schema di firma digitale deve poter rimettere in chiaro in tempo polinomiale il testo cifrato generato dal corrispondente algoritmo di firma e che chi non conosce quest'ultimo non deve poter disporre di un altro algoritmo con tempo polinomiale in grado di trasformare un testo in chiaro in un cifrato verificabile positivamente.

Non si può escludere a priori che esistano anche alcuni risultati facili da invertire: l'importante è che siano pochissimi, per rendere assolutamente trascurabile la probabilità che l'attaccante si trovi proprio in quelle situazioni.

Vale dunque la seguente definizione:

"**una funzione F è unidirezionale se è invertibile, facile da calcolare e se per quasi tutti gli x appartenenti al dominio di F è difficile risolvere per x il problema y = F(x)**".

Cominciamo con prendere atto che strumenti con "comportamento unidirezionale" sono presenti anche nella vita di ogni giorno.

ESEMPIO – La versione informatica dell'elenco telefonico è un file formato da tanti record "abbonato" contenenti ciascuno un campo "cognome-nome" ed un campo "numero telefonico".

Supponiamo ora che il campo cognome-nome sia di **n** bit e che il file sia ordinato per valori crescenti di questo numero binario.

Due sono gli intendimenti, uno inverso dell'altro, che possono spingere ad interrogare il file:

- *trovare il numero di un certo abbonato*,
- *trovare l'abbonato che ha un certo numero*.

L'algoritmo di **ricerca binaria** risolve il primo problema in un tempo **O(n)**: impiegando uno dopo l'altro i bit che codificano l'abbonato si dimezza, infatti, progressivamente l'insieme dei record da prendere in considerazione e si giunge quindi al risultato in **n** passi.

Per risolvere il problema inverso si deve ricorrere all'algoritmo di **ricerca esauriente**, il cui tempo di esecuzione ha ordine di grandezza $\mathbf{O(2^n)}$: nel caso peggiore occorre esaminare tutti i record, in media occorre esaminarne la metà e se si è molto fortunati si trova la soluzione al primo passo.

Numeri speciali (ad esempio il 113) sono immediatamente associabili a chi li possiede.

Rientriamo ora nel contesto della sicurezza informatica per dare due notizie, una cattiva ed una buona.

La cattiva è che finora la Teoria della complessità computazionale non è riuscita a dimostrare l'esistenza delle funzioni unidirezionali.

La buona. I Crittografi sono riusciti ad individuare funzioni di compressione, di cifratura e di firma calcolabili con algoritmi polinomiali ed hanno altresì verificato che, almeno allo stato attuale delle conoscenze, tali funzioni sono invertibili solo con algoritmi o esponenziali, o sub-esponenziali.

Nella pratica crittografica si può dunque oggi contare sulla disponibilità di molte funzioni **candidate** ad essere considerate unidirezionali.

Alcuni algoritmi prevedono di intervenire, con modalità particolari, sui singoli bit dei dati da elaborare: casi concreti verranno esaminati in questo e nei due successivi capitoli.

Altri algoritmi trattano i bit dei dati da elaborare come coefficienti di numeri binari, su cui eseguono operazioni proprie dell'**aritmetica modulare**. Casi significativi verranno esaminati alla fine del cap. 4 e nel cap.5 unitamente ai problemi della **Teoria dei numeri** su cui basano la loro sicurezza (la fattorizzazione di un numero intero, il logaritmo discreto, la radice quadrata modulo un numero composto, ecc.); per tali problemi, infatti, non esistono (e probabilmente non esisteranno mai) algoritmi polinomiali in grado di risolverli.

Un'ultima precisazione: nei casi della decifrazione e della firma, l'utente deve poter fare in tempo polinomiale quello che l'intruso non deve invece poter fare. Occorre dunque un ulteriore modello:
"**una funzione F è detta pseudo-unidirezionale** (*trapdoor one-way function*) **se appare come unidirezionale per chiunque non sia in possesso di una particolare informazione sulla sua costruzione".**

2.1.7 Algoritmi con chiave

Gli utenti devono tenere segrete alcune delle elaborazioni che eseguono (la decifrazione, l'attestazione di autenticità, la generazione della prova d'identità).

Per tenere segreta una trasformazione è possibile usare:

- una macchina a **funzionamento segreto**,
- una macchina nota a tutti, ma dotata di un **programma segreto**,
- una macchina nota a tutti, un programma noto a tutti ed un **parametro segreto** da fornire prima che inizi la sua esecuzione.

La sola segretezza di macchine e algoritmi non ha mai dato sicurezza.

- L'ispezione di un'apparecchiatura lasciata incustodita ha consentito in tanti casi di violare il segreto.
- Abbastanza onerose sono inoltre le azioni da intraprendere (nuovi accordi tra gli interessati, nuove macchine da installare) quando si teme o si è certi che il segreto sia stato violato.
- Realizzare, commercializzare e certificare macchine e algoritmi a funzionamento segreto è difficile.

Decisamente più sicuro è il principio[7] di usare **algoritmi di dominio pubblico** e di tenere **segreto** soltanto un loro parametro, detto **chiave**.

ESEMPIO - Tutti possono sapere come è fatta la serratura di una cassaforte: chi non conosce la combinazione non riuscirà mai, infatti, ad aprirla.

La divulgazione dell'algoritmo ha due conseguenze di notevole importanza nella Società dell'informazione:

- la sicurezza può essere valutata e certificata da tutti gli esperti della comunità scientifica internazionale;
- tutti gli utenti possono agevolmente procurarsi "sorgenti" ed "eseguibili" di meccanismi sicuri.

ESEMPI – La scelta di **AES**, il più recente standard per la difesa della riservatezza voluto dal Governo Federale degli USA, è stata preceduta da una pubblica valutazione delle proprietà di sicurezza degli algoritmi proposti.

Tutti i meccanismi per la programmazione di applicazioni sicure in linguaggio **C++** sono disponibili nella libreria **Crypto++** (www.eskimo.com/~weidai/cryptlib.html). Le funzionalità **JCE** e **JSSE** del compilatore JDK mettono le stesse cose a disposizione di chi programma applicazioni in linguaggio **Java** (www.sun.com).

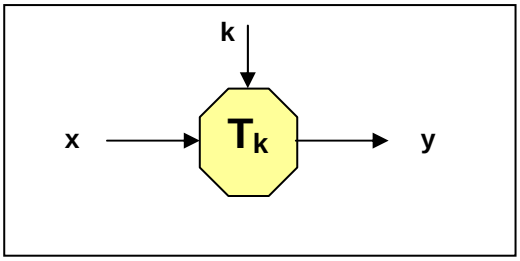
Per completare il quadro notiamo che esiste naturalmente anche la soluzione di tenere segreto tutto: la macchina, l'algoritmo e la chiave. Pochi anni fa è stata sfruttata in questo senso la tecnologia elettronica VLSI, ma il tentativo non ha avuto successo anche perché l'obiettivo non era affatto quello di incrementare la sicurezza.

[7] Sancito dall'olandese Kerckhoffs nel trattato "*La criptogràphie militaire*",1883

ESEMPI - La segretezza del chip **Clipper** e dell'algoritmo **Skipjack**, voluta all'inizio degli anni '90 dal Governo USA per rendere legale l'individuazione delle chiavi segrete usate da persone sospette, ha avuto illustri difensori[8] e numerosissimi oppositori, che la vedevano come un'inaccettabile violazione dei diritti individuali. Dopo pochi anni chip ed algoritmo sono stati declassificati. Gli algoritmi segreti inseriti nel chip **SIM** del GSM, voluti dai primi Costruttori di cellulari per eliminare la concorrenza, sono stati dopo poco divulgati su Internet.

Nella stragrande maggioranza dei casi, le trasformazioni operate alle estremità di un canale insicuro sono dunque oggi descritte da algoritmi di pubblico dominio *inizializzati,* o *parametrizzati*, da chiavi scelte dagli utenti.

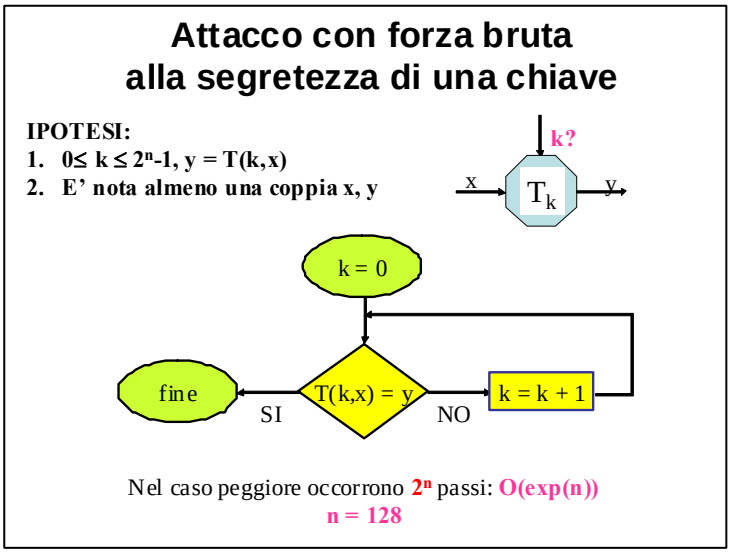
In certi algoritmi la chiave è formata da un'unica stringa di bit, in altri da due o più stringhe concatenate.



Un meccanismo con chiave è usualmente rappresentato dal simbolo grafico di figura; per indicare la sua relazione di causa/effetto nel seguito impiegheremo le notazioni

$y = T_k(x)$, o anche $y = T(k,x)$, ove

- **x** è il dato d'ingresso,
- **y** è il risultato finale,
- $T = \{t_1, t_2, \ldots, t_N\}$ è l'insieme delle trasformazioni eseguibili,
- **k** è la **chiave** che sceglie la trasformazione da eseguire; l'insieme $K = \{k_1, k_2, \ldots, k_N\}$ è detto **spazio delle chiavi**.



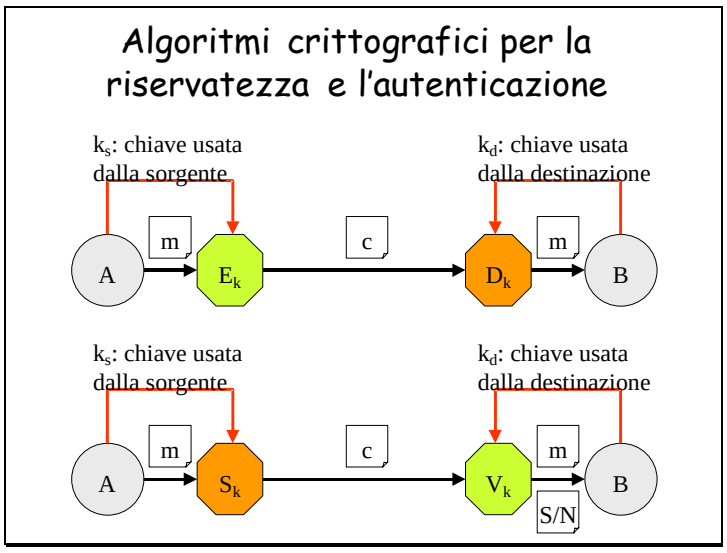
Lo spazio delle chiavi di un algoritmo crittografico deve essere molto grande. Per avere in testa una prima stima di questa dimensione basta pensare che l'**attacco con forza bruta** è la più semplice idea che può venire in testa ad un intruso in possesso di una coppia di testo in chiaro e di testo cifrato.

In figura è mostrato che occorrono al più 2^n passi per individuare una chiave di n bit. Per quanto abbiamo osservato nel paragrafo 2.1.5 oggi deve essere

$$80 < n \leq 128.$$

Più avanti vedremo che per certi algoritmi l'intruso dispone di ulteriori informazioni sulla chiave e può sfruttarle per condurre attacchi più efficienti (di complessità sub-esponenziale). In questi casi un livello di sicurezza di 128 bit richiede chiavi di più di 1000 bit.

2.1.8 Crittografia simmetrica e asimmetrica



Da quanto detto in precedenza discende che la difesa della riservatezza e della autenticità deve essere affidata ad una **coppia di algoritmi con chiave**.

Indichiamo dunque con $k_s \in K$ la chiave impiegata dalla sorgente e con $k_d \in K$ la chiave impiegata dalla destinazione.

I due valori non sono indipendenti: tramite k_d la destinazione deve, infatti, scegliere la trasformazione inversa di quella scelta dalla sorgente con k_s.

La relazione tra le chiavi è dunque una funzione: $K \rightarrow K$: $k_s = f(k_d)$.

Dalla complessità computazionale di questa funzione discende un'importante classificazione dei meccanismi per cifrare/decifrare e per autenticare/verificare.

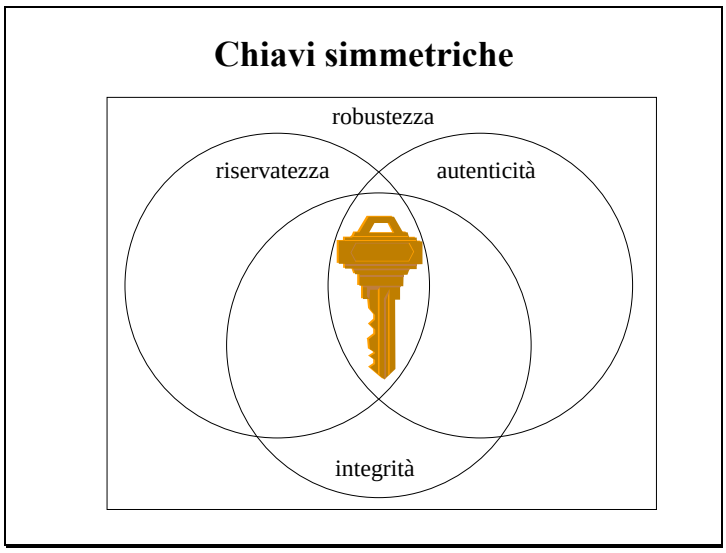
[8] D. E. Denning: "The Case for the Clipper", *Technology Review* n.86, 1995.

- **Meccanismo a chiavi simmetriche** o **simmetrico:** le chiavi k_s, k_d sono o **uguali**, o **facilmente calcolabili** una dall'altra.
- **Meccanismo a chiavi asimmetriche** o **asimmetrico:** le chiavi k_s, k_d sono **diverse** ed una delle due (k_s nella autenticazione, k_d nella riservatezza) è **difficilmente calcolabile** dall'altra.

Queste due categorie di meccanismi differiscono anche per i principi su cui si basano e per le tecniche impiegate per realizzarli. I primi sono oggetto di studio della **Crittografia simmetrica**, i secondi della **Crittografia asimmetrica**. Entrambe le categorie sono in grado di difendere la riservatezza e l'autenticità, se pure con diversa efficienza e sicurezza, come vedremo meglio nei capitoli 4 e 5.

Crittografia simmetrica: la chiave condivisa

Nei meccanismi **simmetrici** si ha di norma $\mathbf{k = k_s = k_d}$. La chiave **k**, usualmente detta **segreta** o **condivisa** perché deve essere preventivamente e segretamente concordata da ogni coppia di corrispondenti, seleziona la trasformazione che ciascuno di essi deve operare.



Quattro sono le proprietà di sicurezza che gli utenti devono assicurare alla chiave condivisa:

- **autenticità:** quando si concorda il valore della chiave in modo telematico occorre essere ben certi che il corrispondente è proprio quello voluto;
- **segretezza**: l'accordo sulla chiave deve avvenire su un canale a prova di intercettazione; la chiave deve poi essere memorizzata su un supporto a prova di intrusione;
- **integrità**: la comunicazione della chiave e la sua successiva memorizzazione devono rilevare la presenza di errori;
- **robustezza**: gli algoritmi di crittanalisi che individuano la chiave devono essere computazionalmente infattibili.

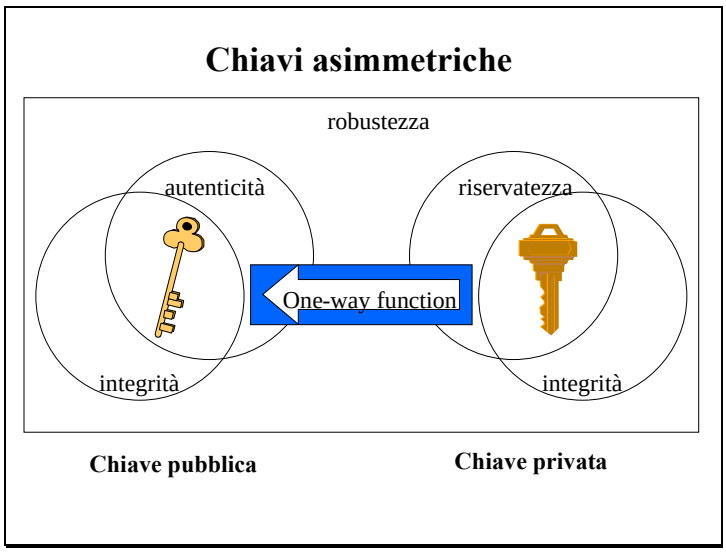
Crittografia asimmetrica: la chiave privata e la chiave pubblica

Nei meccanismi **asimmetrici** i due differenti valori di chiave da usare alle due estremità del canale insicuro sono preventivamente generati, con un procedimento di facile esecuzione, da **uno solo** dei due corrispondenti.

Ogni utente **U** di un algoritmo asimmetrico ha dunque due chiavi:

- la chiave segreta **SU**, la cosiddetta **chiave privata,** che U impiega, a seconda del tipo di algoritmo, o per decifrare un messaggio riservato, o per autenticare un suo documento, o per provare la sua identità;
- la chiave non segreta **PU**, la cosiddetta **chiave pubblica**, che U mette a disposizione di chiunque debba o inviargli messaggi riservati, o verificare l'autenticità dei suoi messaggi.

Per un uso sicuro di ogni algoritmo asimmetrico, la funzione **PU = f(SU)** deve essere **unidirezionale**.



In conclusione, le proprietà di sicurezza della chiave privata **SU** sono tre:

- **segretezza**: il valore ed il modo con cui è stato ottenuto devono essere tenuti riservati;
- **integrità**: la memorizzazione deve essere a prova d'errore;
- **robustezza**: gli algoritmi in grado di calcolare la funzione $\mathbf{SU = f^{-1}(PU)}$ devono essere come minimo sub-esponenziali e la dimensione della chiave pubblica **PU** ne deve rendere impossibile l'esecuzione.

Le proprietà di sicurezza della chiave pubblica **PU** sono due:

- **autenticità**: tutti devono poter essere certi che **PU** è proprio di **U**,
- **integrità**: la memorizzazione deve essere a prova di errore.

2.2 Crittanalisi

Solo il legittimo proprietario devono conoscere la chiave condivisa di un meccanismo simmetrico, o la chiave privata di un meccanismo asimmetrico, o l'informazione con cui prova la sua identità. Cominciamo a prendere confidenza con la Crittanalisi, individuando cosa può fare l'intruso e come ci si deve difendere.

2.2.1 – Dimensionamento del segreto

L'intruso che vuole carpire un dato segreto può tirare ad indovinarlo; il problema è sapere se ci ha azzeccato, ma è tutt'altro che difficile mettersi in questa situazione.

ESEMPIO – Nello schema di firma digitale l'intruso può intercettare il messaggio **m** ed il cifrato $\mathbf{c = S_{SA}(H(m))}$ che la sorgente **A** gli ha affiancato per autenticarlo. Gli algoritmi H e S sono per ipotesi di pubblico dominio. L'intruso può quindi scegliere una chiave **x**, calcolare $\mathbf{S_x(H(m))}$ e verificare se il risultato è **c**.

La sicurezza dipende da come è stato scelto il segreto. Se l'utente, per potersela facilmente ricordare a memoria, ha usato un'informazione che l'intruso può prevedere (ad esempio il suo nome, o il suo anno di nascita, o i dati di un suo familiare, ecc.), il tirare ad indovinare ha una notevole probabilità di successo.

La prima difesa preventiva è quindi quella di usare **dati casuali**.

- R12: "*I bit della stringa che rappresenta un dato segreto devono essere equiprobabili ed indipendenti* ".

Ciò richiede la disponibilità di un nuovo meccanismo per la sicurezza, il **generatore di numeri casuali** o **RNG** *(Random Number Generator*). Ce ne occuperemo tra poco: dobbiamo, infatti, prima valutare quanto lunga deve essere la stringa che rappresenta un dato segreto.

n° di bit	n° di valori
20	$2^{20} = 1 \times 10^6$
32	$2^{32} = 4{,}3 \times 10^9$
56	$2^{56} = 7{,}2 \times 10^{16}$

Consideriamo una stringa formata da **n** bit. Se tutti i $\mathbf{2^n}$ possibili valori sono equiprobabili, la probabilità di indovinare al primo colpo è $\mathbf{2^{-n}}$ ed una stringa di una ventina di bit sembrerebbe quindi sufficientemente robusta; se però è possibile fare **k** tentativi con valori sempre diversi, la probabilità di successo è **k** volte più grande.

Una seconda difesa preventiva è quella di non concedere all'intruso il tempo di svolgere molte prove.

ESEMPIO – Il **PIN** del Bancomat è formato da cinque numeri decimali scelti a caso dal calcolatore della Banca. I valori possibili sono 10^5, ma è impossibile provarli tutti: dopo tre digitazioni errate lo sportello automatico chiude il collegamento e trattiene la scheda.

Non sempre è possibile: in moltissimi casi l'intruso ben organizzato può, infatti, impiegare un suo calcolatore per fare un attacco con forza bruta. Nel paragrafo 2.1.5 abbiamo però già segnalato che questo attacco è oggi ineseguibile se si impiega un **centinaio di bit**.

ESEMPIO – Un intruso che è in grado di decidere in un microsecondo se una certa stringa di 80 bit è o no il dato che vuole scoprire, per provarle tutte dovrebbe fare calcoli per 10^{11} anni!

Abbiamo già anche commentato il fatto che la complessità computazione non mette in conto il caso più favorevole per l'intruso; utile è dunque la contemporanea adozione di un secondo provvedimento.

- R13: "*un dato segreto deve essere frequentemente modificato*".

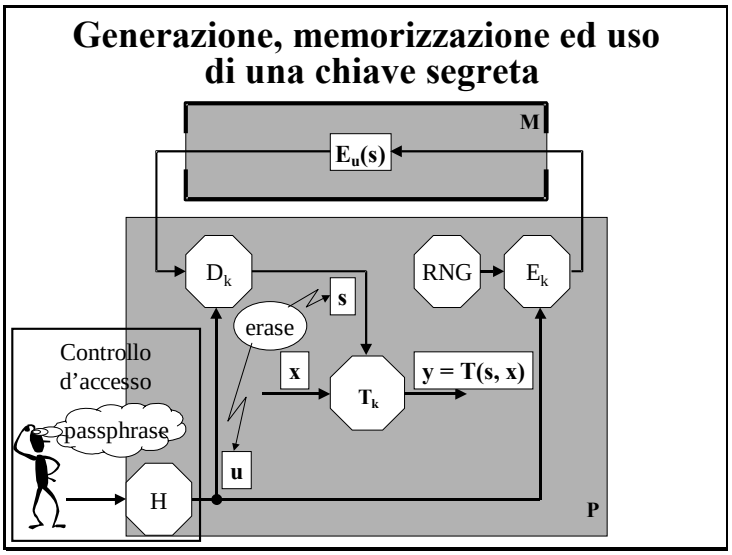
2.2.2 – Memorizzazione del segreto

Una lunga stringa di bit aleatori è difficile da imparare a memoria. Ogni utente ha dunque bisogno di una **memoria artificiale**, in cui alloggiare tutti i suoi segreti e da cui poter di volta in volta estrarre quello che serve al meccanismo da mettere in esecuzione.

Per valutare la sicurezza di questa soluzione, consideriamo il caso di una memoria non sicura **M**, in cui si deve conservare una chiave segreta **s**, e di un processore sicuro **P**, in cui deve essere eseguito un algoritmo $\mathbf{T_k}$ che trasforma un dato **x** nel dato $\mathbf{y = T_s(x)}$.

La prima minaccia da prendere in considerazione è che l'intruso possa leggere **s** su **M**. La seconda minaccia è che l'intruso possa impadronirsi della chiave o intercettandola sul canale tra M e P, o leggendola sull'area di lavoro di P. Le contromisure da adottare sono espresse dalle seguenti regole.

- R14: "*solo il proprietario della chiave segreta deve poter avere accesso (in scrittura ed in lettura) alla cella in cui è memorizzata*".
- R15: "*una chiave segreta deve essere mantenuta in memoria in una forma cifrata stabilita dal proprietario*".
- R16: "*la forma cifrata di s deve essere mantenuta anche nel trasferimento da M a P*".
- R17: "*P deve poter decifrare s solo se il proprietario gli comunica la chiave che ha scelto per cifrarla; una volta usato il testo in chiaro s, P deve cancellarlo accuratamente dalla sua memoria di lavoro* ".



Diversi sono i meccanismi chiamati in causa dalle regole precedenti: un **Controllo d'accesso** a M, un **RNG** per la generazione della chiave, un **Cifrario simmetrico** che la trasformi immediatamente in testo cifrato e che ripristini il testo in chiaro solo quando rientra nel processore, una **primitiva per cancellare** s (e non solo il suo puntatore, come fanno di solito i calcolatori non sicuri) al termine del calcolo di y.

Occorre anche preoccuparsi della chiave **u** che l'utente impiega per cifrare/ decifrare **s**: da un lato deve essere casuale, da un altro lato facilmente imparabile a memoria.

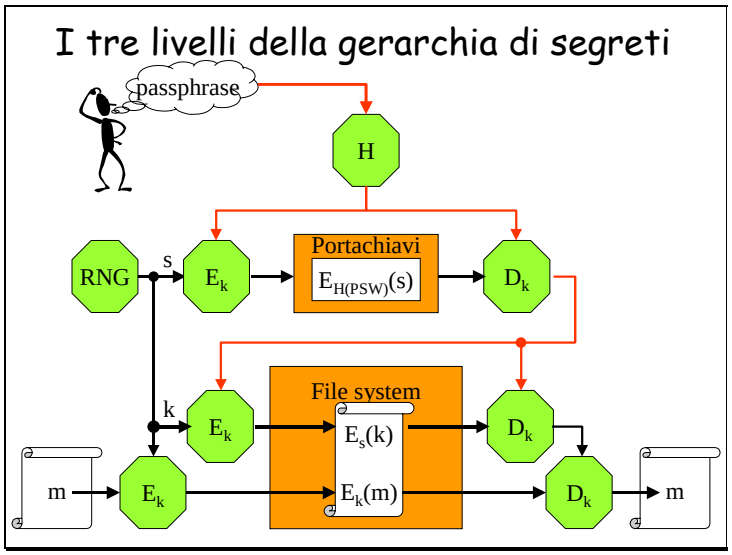
Entrambe le proprietà vengono di norma conseguite impiegando l'**impronta** di una frase, la cosiddetta **passphrase**, che l'utente impara a memoria e comunica al processore su un canale protetto.

Bisogna infine considerare la sicurezza dell'intero sistema, dato che molto spesso l'uso del processore P è condiviso da più utenti e non è quindi prudente lasciare sul hard disk la chiave segreta, anche se cifrata.

Una prima buona soluzione è quella di dotare ogni utente di una **memoria portatile** M, da inserire nel sistema soltanto quando ha la necessità di elaborare il suo dato segreto.

Una soluzione ancora più sicura è quella di dotare ogni utente di un **calcolatore portatile**, cui il sistema invia **x** e da cui ottiene **y**: in questa maniera, infatti, è possibile non fare mai uscire **s** dal suo ambito protetto.

Il caso appena discusso non è però generale. In un **file system sicuro**, un utente ha, infatti, la necessità di gestire una **gerarchia di tre dati segreti**.



Al primo livello della gerarchia dei segreti c'è la **pass phrase** che l'utente deve imparare a memoria e che impiega per accedere ai segreti di secondo livello. Il meccanismo chiamato in causa è la **funzione hash**.

Al secondo livello ci sono, cifrate dalla pass phrase, le **chiavi** (in generale sono più di una) che l'utente impiega per accedere alle chiavi che proteggono la riservatezza dei documenti riservati di terzo livello.

La memoria per le **chiavi che cifrano chiavi** è un **file**, usualmente detto **portachiavi** (*keyring*). In questo servizio sono coinvolti un RNG ed un cifrario simmetrico: nel gergo informatico tale soluzione è indicata dalla sigla **PBE** (*Password Based Encryption*).

Al terzo livello ogni file è cifrato con una chiave scelta di volta in volta dal RNG e poi

cifrata/decifrata con una chiave del portachiavi. Il testo cifrato di quest'ultima chiave è memorizzato in testa al documento che ha cifrato, unitamente all'identificativo della chiave del portachiavi che la rimette in chiaro.

La precedente organizzazione ha una pericolosa vulnerabilità: chi si dimentica la pass phrase o chi ha un portachiavi non più leggibile perde tutti i suoi dati riservati!

Determinante è dunque il prevedere una qualche forma di **recovery**. Usuale è mantenere il file system ed il suo backup nell'**hard disk**, alloggiare il keyring anche in una memoria portatile e confidare la pass phrase a qualcuno di cui ci si fida. Molti sono i prodotti commerciali in grado di realizzare un file system sicuro.

ESEMPI - **Secure Drive** per MS-DOS, **Entrust /Ice** per Windows, **PGP** per tutte le piattaforme oggi in uso.

2.2.3 – Classificazione degli attacchi

L'impossibilità di deduzione del dato segreto dal risultato della trasformazione in cui è coinvolto è una proprietà indispensabile di tutti gli algoritmi ed i protocolli crittografici. Per verificare questa proprietà, la Crittanalisi prevede di far riferimento a quattro differenti contesti in cui può trovarsi ad operare chi svolge un attacco.

ATTACCO	CONOSCENZE DELL'INTRUSO
• con solo testo cifrato	Il linguaggio del testo in chiaro e la probabilità d'occorrenza dei simboli
• con testo in chiaro noto	Coppie formate da un testo in chiaro e dal testo cifrato corrispondente
• con testo in chiaro scelto	Testi cifrati corrispondenti a testi in chiaro di sua scelta
• con testo cifrato scelto	Testi in chiaro corrispondenti a testi cifrati di sua scelta

Dall'alto verso il basso crescono sia la difficoltà per l'intruso di avere la conoscenza prevista, sia la sua probabilità di successo nei casi in cui la possiede.

ESEMPIO – La verifica dell'ipotesi che **x** sia la chiave impiegata in una firma digitale (v. esempio a pag. 26) è un attacco con testo in chiaro noto.

Per gli attuali meccanismi crittografici l'impossibilità di deduzione è sempre verificata. Un accorgimento ha un ruolo fondamentale nell'ottenimento di "robustezza" agli attacchi:
"**il legame tra il testo in chiaro ed il testo cifrato deve essere così ben nascosto da far apparire quest'ultimo all'intruso come una variabile aleatoria che assume con uguale probabilità tutti i suoi possibili valori**".

Con questa ipotesi sembrerebbe lecito concludere che l'intruso abbia a disposizione soltanto la forza bruta, attacco che abbiamo già visto poter essere fronteggiato attribuendo alle variabili aleatorie un'adeguata dimensione. In realtà le cose non stanno proprio cosi. Un intruso che ha a disposizione un numero elevato di testi cifrati può condurre attacchi più efficaci mettendo in conto le **proprietà statistiche delle variabili aleatorie**.

2.3 Meccanismi di base

2.3.1 Generatori di numeri casuali

Un **RNG** (**Random Number Generator**) è chiamato in causa, in un modo o nell'altro, da tutti i meccanismi ed i servizi per la sicurezza. Alle volte è richiesto al generatore di fornire un **singolo bit**, altre volte un **numero di n bit**, altre volte ancora **sequenze di bit** di grande e variabile lunghezza.

Alle stringhe di simboli d'uscita del generatore di numeri casuali è richiesto il rispetto della proprietà di **casualità**: ogni valore deve

1. **avere la stessa probabilità di verificarsi**,
2. **essere statisticamente indipendente da tutti gli altri** (la probabilità di una qualsiasi coppia di valori nella stringa deve essere il prodotto delle loro probabilità).

Per verificare la casualità di una sequenza di bit sono stati definiti diversi **test statistici.**

Un riferimento internazionalmente riconosciuto è lo standard FIPS 140-2 (definito dal NIST[9] Random Number Generation Technical Working Group), che elenca 16 differenti test su sequenze di 20.000 bit e le relative gamme di valori entro le quali devono ricadere i risultati per poterle classificare come casuali.

[9] *National Institute of Standards and Technology*

I primi quattro test, da superare tutti obbligatoriamente, sono basati sulla distribuzione "chi quadro":

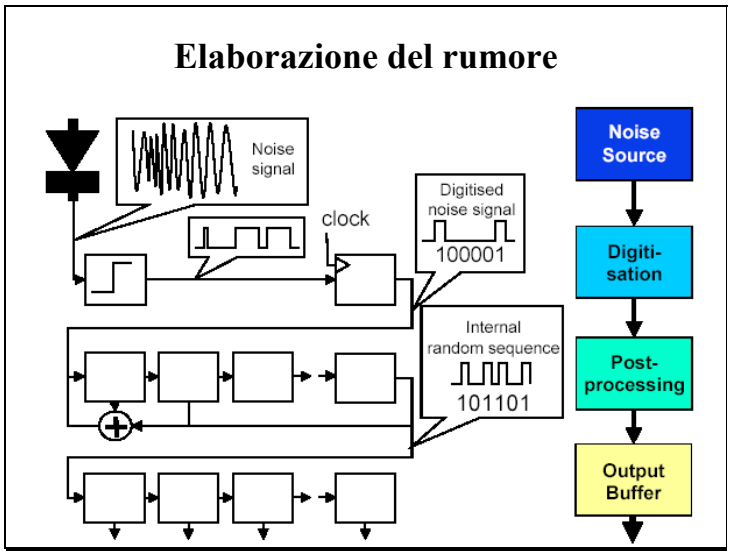
- il **monobit** valuta se il n° di "uni" e di "zeri" presenti all'interno della sequenza è approssimativamente lo stesso;
- il **poker** suddivide la sequenza in blocchi di 4 bit consecutivi e valuta se il numero di ciascuna delle 2^4 configurazioni è approssimativamente lo stesso;
- il **run** considera le stringhe di bit consecutivi tutti o a "uno" (*block*) o a "zero" (*gap*), e valuta, per le varie lunghezze, se il loro numero approssima il valore atteso per una vera sequenza casuale;
- il **long run** considera il più lungo "run di uni" presente nella sequenza in esame e valuta se la sua lunghezza è compatibile con quella attesa per una vera sequenza casuale.

Altri test in uso si basano sul calcolo di particolari funzioni della stringa:

- l'**autocorrelazione** controlla, bit a bit, di quanto differiscono la stringa originaria e la stessa traslata di una certa quantità,
- la **trasformata discreta di Fourier** controlla se esistono pattern ripetitivi troppo vicini uno all'altro,
- la **compressione di Lempel-Ziv** verifica che la lunghezza della stringa non sia significativamente ridotta.

1 - TRNG

La casualità è presente in diversi fenomeni fisici (tempo medio di emissione di particelle durante il decadimento radioattivo, rumore termico ai morsetti di un diodo o di un resistore, turbolenza di fluidi, ecc.) e nei segnali d'uscita di diverse apparecchiature elettroniche (rumore presente nell'audio generato da un microfono o nel video generato da una telecamera, fluttuazioni della frequenza di un oscillatore, ecc.).



Tali sorgenti di rumore sono spesso indicate con l'acronimo TRNG (*True Random Number Generator*).

In figura è ad esempio mostrato il caso del rumore termico di un diodo per evidenziare come questi **generatori Hw** richiedano in generale

- una digitalizzazione del segnale analogico fornito dalla sorgente,
- una post-elaborazione della stringa di bit,
- una memorizzazione del dato da mettere a disposizione dell'applicazione.

Una post-elaborazione spesso necessaria è il cosiddetto *de-skewing*, che mira sia a rendere equiprobabili gli "uni" e gli "zeri" presenti nella stringa, sia a garantirne una buona indipendenza.

ESEMPIO – Sia **p** la probabilità che la sorgente generi un "uno" e quindi **1-p** la probabilità che emetta uno "zero".

Per eliminare un'eventuale polarizzazione (**p** ≠ **1-p**), Von Neumann ha suggerito un metodo semplice ed elegante: dopo aver suddiviso la stringa in coppie di bit consecutivi, se ne genera un'altra sostituendo **1** alle coppie **10**, **0** alle coppie **01** ed eliminando le coppie **11** e **00**.

Ottime caratteristiche di casualità si ottengono oggi post-elaborando m bit alla volta con una **funzione hash sicura**.

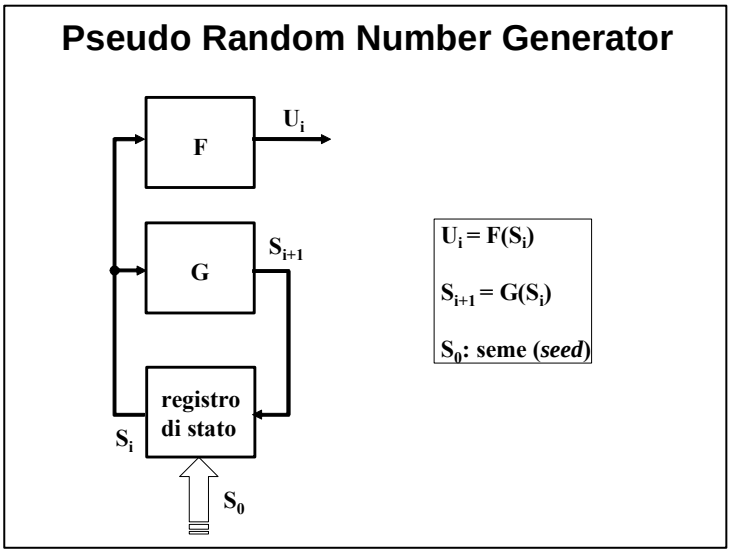
E' possibile impiegare anche **generatori Sw**, cioè programmi che estraggono "rumore" direttamente dal funzionamento di un computer (digitazione della tastiera, movimentazione del mouse, stato del processore, occupazione del hard disk, traffico di rete).

Tutte le precedenti tecniche di generazione hanno un problema non trascurabile per le tecnologie della sicurezza: la **frequenza di produzione** dei dati casuali è bassa e bisogna quindi o accontentarsi di stringhe relativamente corte o usare risultati calcolati in precedenza e poi memorizzati in un apposito *random pool*.

Un secondo problema, questa volta praticamente insormontabile, si presenta quando si usano meccanismi come **one time pad** ed i **cifrari a flusso** (li discuteremo più avanti), che richiedono la generazione di sequenze di bit casuali **identiche** in **calcolatori diversi**.

2 – PRNG

I programmi di simulazione e di elaborazione statistica dei dati hanno da sempre impiegato **algoritmi deterministici** per generare lunghe sequenze di numeri casuali a partire da un dato iniziale, detto **seme** (*seed*), di piccola dimensione.



Il modello di tali generatori algoritmici, detti **pseudocasuali** o **PRNG** (*Pseudo Random Number Generator*), è l'automa a stati finiti illustrato nella figura a lato.

Un automa a stati finiti è un PRNG solo se supera i test di casualità.

I PRNG differiscono dai TRNG per tre aspetti:

- la sequenza di numeri in uscita ad un certo punto si ripete,
- le sottosequenze ottenibili al variare del seme sono molto meno di quelle teoricamente possibili per una vera sorgente di rumore,
- in assenza di particolari accorgimenti nella costruzione delle funzioni F e/o G, il valore in uscita è prevedibile se si conosce un certo numero di valori precedenti.

ESEMPI – Il primo PRNG fu inventato da J. Von Neumann per il calcolatore ENIAC: preso un numero decimale di n cifre, lo si eleva al quadrato e si prendono le n cifre centrali del risultato per iterare il procedimento.

Nei calcolatori IBM della Serie 360 (1969) è stato impiegato il **metodo di congruenza lineare**[10] che prevede di iterare il seguente calcolo: $\mathbf{X_{i+1} = (aX_i + b) \bmod m}$[11], con **m** primo ed **a**, **b** interi.

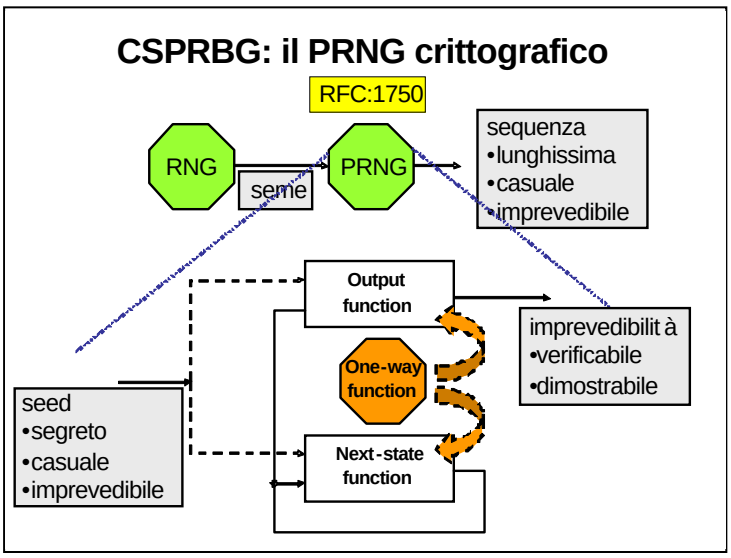
Un'ottima casualità è stata ottenuta con $a=7^5$, $b=0$, $m=2^{31}-1$.

Bastano pochi $\mathbf{X_i}$ per individuare **m**, **a**, **b** e quindi ogni altro successivo valore di **X**.

3 – CSPRBG

Nelle applicazioni crittografiche la sola **casualità** non è sufficiente. Occorre, infatti, anche l'**imprevedibilità**: un intruso che è riuscito ad intercettare l'uscita o ad individuare, in tutto o in parte, lo stato del generatore non deve poter dedurre da quale seme sono partiti i calcoli e/o quali saranno i prossimi valori generati.

Un generatore pseudocasuale che ha anche la proprietà di imprevedibilità è detto **crittograficamente sicuro** o **CSPRBG** (*Cryptographically Secure PseudoRandom Bit Generator*). Per conseguire imprevedibilità occorre che



- il periodo sia **grandissimo** (10^{50} – 10^{100}), per poterlo suddividere con il seed in moltissime sottosequenze;
- il seme sia **imprevedibile** e tenuto **segreto** (il documento RFC 1750 "*Randomness recommendation for security*" suggerisce di estrarre il seed, in modo protetto, da **una vera sorgente di rumore**);
- sia **unidirezionale** o la funzione di stato futuro, o la funzione d'uscita, per rendere impossibile ad un avversario che ha individuato uno stato il risalire agli stati precedenti ed al seme;
- sia verificato il **next-bit test**: "dati L bit della stringa non deve esistere alcun algoritmo polinomiale in grado di predire il bit L+1-esimo con probabilità maggiore di 0,5".

La funzione unidirezionale impiegata dai CSPRBG può essere un meccanismo o simmetrico (una funzione hash, una funzione di cifratura) o asimmetrico (una esponenziazione modulare).

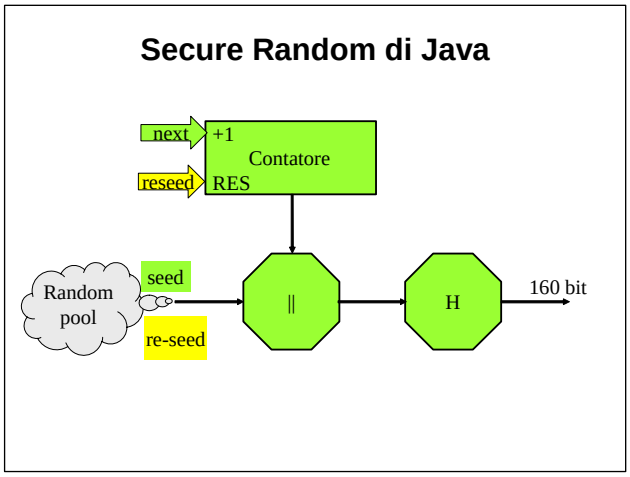
[10] Lehmer (1951)

[11] La notazione **A mod B** indica il resto della divisione di A per B

Esistono due differenze notevoli:

- i primi sono decisamente più efficienti dei secondi;
- l'imprevedibilità dei primi è stata verificata solo tramite serie di test specifici, mentre nei secondi è dimostrabile teoricamente, una volta che si sia assunto come intrattabile un certo problema della Teoria dei numeri.

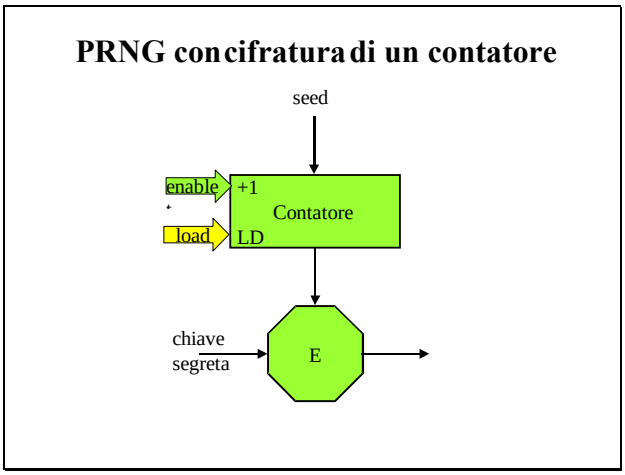
Cominciamo ad esaminare tre soluzioni della prima categoria; altre verranno discusse nel cap. 4.



SecureRandom di Java - Nel linguaggio Java è disponibile una classe che consente di istanziare un generatore pseudocasuale basato su una funzione hash non invertibile.

Il generatore rispetta lo standard IEEE P1363 : il numero casuale è generato calcolando l'hash di un seme casuale (la SUN non lo ha mai chiarito, ma sembra che sia di soli 20 bit) concatenato con l'uscita di un contatore a 64 bit; il contatore parte da zero ed è poi incrementato di 1 ad ogni operazione.

Il seed, in questo caso un ingresso della F, può essere specificato o dall'utente, o da JDK, a partire da dati casuali estratti dal calcolatore; per rendere aperiodico il comportamento, è possibile iniettare nuova incertezza ridefinendo il seme (*re-seed*) dopo un certo numero di iterazioni.

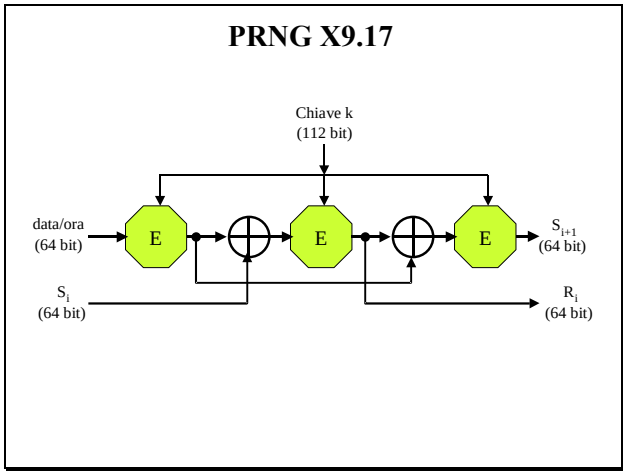


Cifrario simmetrico in modalità CTR - Abbiamo già evidenziato a pag. 22 che la funzione E di un buon Cifrario a chiave segreta può essere considerata **unidirezionale**.

Tale proprietà è stata sfruttata in molti modi diversi nella generazione di sequenze di bit pseudo-casuali.

In figura è mostrata la cosiddetta modalità CTR, che prevede di ottenere numeri a caso cifrando, con una chiave casuale e segreta (il seed in questo caso), l'uscita di un contatore il cui stato viene continuamente incrementato a partire da 0.

Il contatore è dotato anche di un comando di *load* per introdurre dall'esterno un valore di partenza casuale, o per eseguire operazioni di re-seed.



Standard ANSI X9.17 - Lo standard è stato impiegato in ambito bancario per produrre le chiavi segrete ed i vettori casuali di inizializzazione del cifrario DES (v. 4.3.3).

La generazione di un numero pseudo-casuale di 64 bit ($\mathbf{R_i}$) è ottenuta

- cifrando un'informazione oraria di 64 bit con il Cifrario EDE a due chiavi (v. 4.3.5),
- sommando mod 2 al risultato un seme di 64 bit ($\mathbf{S_i}$),
- sottoponendo infine il tutto ad una seconda cifratura.

Si noti in figura l'uso di una terza cifratura per produrre anche un nuovo seme (S_{i+1}) da utilizzare nell'iterazione successiva.

Il generatore è considerato molto robusto, dato che ciascun blocco di EDE esegue tre trasformazioni crittografiche.

Esaminiamo ora due casi significativi individuati nel contesto della Teoria dei numeri.

Prima di farlo è però utile premettere un glossario del simbolismo matematico che dovremo impiegare qui e che riprendere poi nel capitolo 5.

$Z_n = \{0,1, \ldots, n-1\}$	Insieme di tutti gli interi non negativi minori di n
b|a	L'intero b divide l'intero a, o è un divisore di a, se **a = m × b** con m intero
numero primo p	L'intero p > 1 è primo se i suoi unici divisori sono 1 e p
fattorizzazione di a>1	**$a = p_1^{\alpha 1} \times p_2^{\alpha 2} \times \ldots \times p_t^{\alpha t}$** con $p_1 < p_2 < .. < p_t$ primi e αi interi positivi
MCD (a, b)	Il massimo comun divisore degli interi a, b è il più grande intero k, tale che k|a e k|b
numeri coprimi	Gli interi a, b sono detti coprimi, o relativamente primi, se **MCD (a, b) = 1**
Φ(n) funzione *totient*	N° di interi < n e relativamente primi con esso: **$\Phi(n) = n \times (1-1/p_1) \times \ldots \times (1-1/p_t)$**
Z^*_n	Insieme degli elementi di Z_n relativamente primi con n
$Z^*_p = \{1, \ldots, p-1\}$.	Insieme di tutti gli elementi non nulli di Z_p con p primo
$a \equiv b \pmod n$	a è **congruo** a b modulo n se **a mod n = b mod n**, cioè se **a – b = k × n** con k intero
residuo quadratico	L'intero q è un residuo quadratico modulo n se esiste un x tale che **$x^2 \equiv q \pmod n$**

Il generatore di bit pseudocasuali RSA - Siano **p** e **q** due grandi numeri primi scelti a caso, **n = p×q** il loro prodotto, **Φ(n) = (p-1)×(q-1)** la funzione *totient* di Eulero ed 1 < **e** < Φ(n) un intero relativamente primo con **Φ(n)**.

Scelto un "seme" **x_0** nell'intervallo **1 ÷ n-1**, si iterano quante volte è necessario i seguenti calcoli:

1. **$x_i = x_{i-1}^e \bmod n$**,
2. **$b_i = x_i \bmod 2$**

N.B. **b_i**, l'i-esima uscita del generatore, è dunque il bit meno significativo della rappresentazione binaria di **x_i**

Se **p** e **q** sono tenuti segreti, o, ancora meglio, distrutti dopo il calcolo di **Φ(n)**, l'imprevedibilità si basa sulla assunzione che la **fattorizzazione di n** sia un problema difficile.

Per semplificare i calcoli si usa spesso **e = 3**: in questo caso, infatti, ad ogni passo occorrono soltanto un elevamento al quadrato modulo n ed una moltiplicazione modulo n. Per ottenere maggiore efficienza, è possibile estrarre più bit da ogni **x_i**, ma in questo caso manca la dimostrazione che il generatore sia ancora CSPRBG.

Il generatore di bit pseudocasuali BBS[12]- Siano **p** e **q** due grandi numeri primi **congruenti a 3 modulo 4** e **n** il loro prodotto (N.B. il numero n è detto "intero di Blum" e la condizione garantisce che ogni residuo quadratico modulo n abbia una ed una sola radice quadrata che è ancora un residuo quadratico).

Scelto a caso un seme **s** in **Z^*_n**, si calcola **$x_0 = s^2 \bmod n$** e si iterano poi i seguenti calcoli:

1. **$x_i = x_{i-1}^2 \bmod n$**,
2. **$b_i = x_i \bmod 2$**.

Se **p** e **q** sono tenuti segreti, l'imprevedibilità del "prossimo bit" è basata sull'assunzione che sia difficile **estrarre radici quadrate modulo n**. L'efficienza è più alta di quella del generatore RSA: ad ogni passo è, infatti, richiesto soltanto un elevamento al quadrato modulare.

2.3.2 Algoritmi di hash

Una seconda **primitiva crittografica** è la funzione **hash sicura,** chiamata in causa in moltissimi meccanismi e servizi. Gli algoritmi che la realizzano devono in generale presentare le seguenti quattro proprietà.

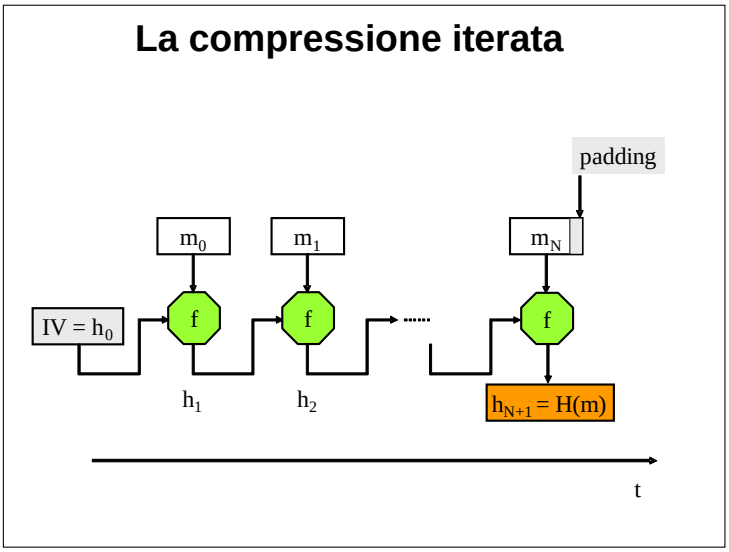
- R18 (efficienza): "*il calcolo di* **H(x)** *è computazionalmente facile per ogni* **x**".
- R19 (robustezza debole alle collisioni): "*per ogni* **x** *è infattibile trovare un* **$y \neq x$** *tale che* **H(y) = H(x)**".
- R20 (resistenza forte alle collisioni): "*è infattibile trovare una qualsiasi coppia* **x, y** *tale che* **H(y) = H(x)**".
- R21 (unidirezionalità): "*per ogni* **h** *è infattibile trovare un* **x** *tale che* **H(x) = h**".

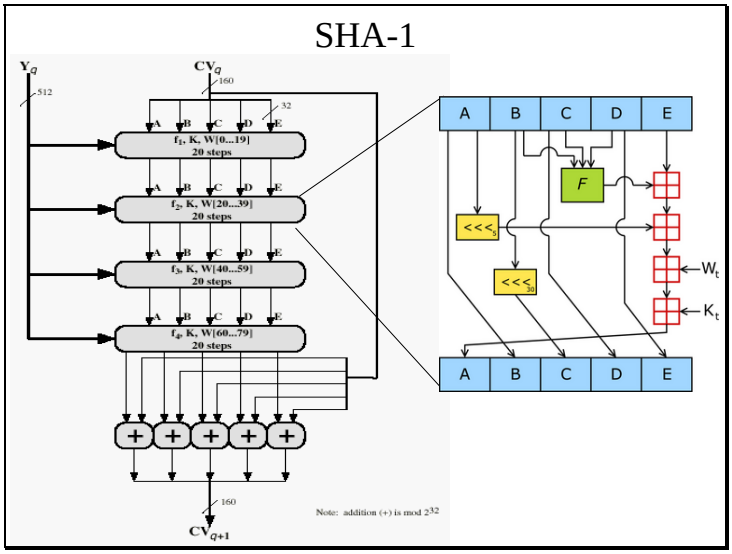
1 - Efficienza

Occorre poter comprimere agevolmente stringhe binarie arbitrariamente lunghe. A tal fine, tutti gli algoritmi di *hash* si ispirano al principio della **compressione iterata**[13].

[12] La sigla discende dai cognomi degli autori dell'articolo che ha divulgato il metodo: L.Blum, M.Blum, M.Shub, "A simple unpredictable pseudo-random number generator", *SIAM Journal on Computing*, 15 (1986), 364-383.

[13] R.Merkle "Secrecy, Authentication and Public Key Systems", *UMI Research Press*, Ann Arbor Michigan 1978; "One Way Hash functions and DES", *Proceedings of CRYPTO '89*; Springger-Verlag





Ogni iterazione impiega una funzione di compressione **f**, **resistente alle collisioni** e **facile** da calcolare, che genera **n** bit d'uscita a partire da due dati d'ingresso, uno di **n** bit ed uno di **r** bit, con **r** maggiore di **n**.

Il messaggio **m** da comprimere è suddiviso in **blocchi** $\mathbf{m_0}$, $\mathbf{m_1}$, .., $\mathbf{m_N}$, tutti di lunghezza **r**; se l'ultimo blocco è più piccolo, lo si completa con un *padding*. Nello standard SHA-1 il pudding è formato da un "uno" seguito da "zeri" e da un numero binario che indica la lunghezza originaria di **m**; il padding è presente (all'interno di un ulteriore blocco) anche nel caso che la lunghezza di **m** sia un multiplo di **r**.

L'algoritmo comprime un blocco alla volta assumendo come stato interno il risultato del passo precedente: $\mathbf{h_i = f(m_{i-1}, h_{i-1})}$.

Il dato $\mathbf{h_0}$ è una **costante** detta vettore d'inizializzazione; l'ultimo **h** è l'impronta di **m**.

Per dare almeno un'idea su come è fatto un algoritmo di hash è sufficiente fare riferimento al caso illustrato in figura[14].

In SHA-1 il messaggio da comprimere è suddiviso in blocchi da 512 bit (16 parole da 32 bit); lo stato interno è di 160 bit (5 parole di 32 bit denominate A,B,C,D,E); al termine delle iterazioni è restituita un'impronta di 160 bit.

Una volta che il blocco del messaggio è stato espanso in 80 parole da 32 bit (in figura W_i per i = 0,1,.., 79), la funzione di compressione le elabora una alla volta in 4 *round* formati ciascuno da 20 *step*. In ogni step i dati a 32 bit sono sottoposti ad operazioni logiche, rotazioni, trasposizioni ed addizioni mod 2^{32}.

Al termine del quarto round i valori di stato così ottenuti sono sommati mod 2^{32} con quelli impiegati nel primo round.

Attacco con estensione del messaggio

E' interessante prendere subito nota che il precisare la lunghezza del messaggio nell'ultimo blocco è un'indispensabile "patch" per una vulnerabilità presente in tutti gli algoritmi a compressione iterata.

Consideriamo una sorgente che, dopo aver concordato un segreto **s** con la destinazione, le invia un messaggio **m** ed il suo "autenticatore" **H(s||m)**, calcolato però senza aver aggiunto alla fine di **m** l'indicazione di quanto è lungo. L'intruso può in questo caso condurre un attacco di *length extension*: individua un'estensione **m'** di **m** di suo interesse (forma cioè un messaggio **m||m'**) e calcola **H(m')**, ponendo attenzione a fornire all'algoritmo come stato iniziale non la costante $\mathbf{h_0}$, ma il dato **H(s||m)** che ha intercettato.

L'impronta **H(m')** così ottenuta è per costruzione uguale a quella di **H(s||(m||m'))** e quindi il messaggio forgiato dall'intruso sarà giudicato dalla destinazione come generato dalla sorgente. Lo standard d'inserire la lunghezza del messaggio in coda al messaggio rende questo attacco più difficile, ma non impossibile.

Attacco con una collisione

Per parare l'attacco con estensione del messaggio, si potrebbe pensare di invertire la posizione di **s** e di **m** all'interno dell'autenticatore, ma la cosa è utile solo se per l'algoritmo di compressione non vale R19.

Supponiamo dunque che l'intruso possa trovare un messaggio **m*** in collisione con **m**.

Con questa ipotesi si ha **H(m||s) = H(m*||s)**: lo stato interno raggiunto dall'algoritmo **H** dopo la compressione delle prime componenti è, infatti, lo stesso e l'uguaglianza non si modificherà con la successiva compressione di **s**, **qualsiasi esso sia**.

In conclusione all'intruso è sufficiente inviare il messaggio **m*||H(m||s)** per spacciarsi per uno dei due corrispondenti.

[14] Uno studio più dettagliato esula dai limiti di questo corso. A chi lo vuole iniziare suggeriamo la consultazione di [6], cap. 12

L'accorgimento della doppia compressione

Esistono altri difetti, più o meno piccoli, che impediscono agli attuali algoritmi di presentare esattamente il comportamento aleatorio della funzione hash ideale. Il loro impiego come meccanismi per la sicurezza richiede dunque l'adozione di accorgimenti particolari.

Nell'attesa, forse vana, di un algoritmo di hash perfetto, i Crittografi suggeriscono di applicare al messaggio **due successive compressioni**; naturalmente ciò va a scapito dell'efficienza.

Lo standard Internet **HMAC** (RFC 2104) prevede un segreto **s** di al più 512 bit.

1. Se è più piccolo, lo si completa concatenandogli una stringa di "zeri".
2. Si calcolano due distinti vettori, **k1 = s ⊕ a** e **k2 = s ⊕ b** con a, b costanti prefissate di 512 bit.
3. Si eseguono due compressioni: **HMAC (s, m) = H(H(k1||m)||k2)**.

Il tutto può essere interpretato come una funzione hash "dotata di chiave":**$HMAC_s$(m) = HMAC (s,m)**.

Ferguson e Schneier ([4], pag. 92) sostengono che lo stesso livello di sicurezza si può ottenere calcolando l'impronta come **H(H(m||s))**, o come **H(H(m||s)||m**).

2 – Resistenza debole alle collisioni

L'impiego della funzione hash nella verifica dell'integrità di un messaggio **m** ha dunque come presupposto che nessuno sappia trovare un messaggio **m***, diverso da **m**, ma tale che sia **H(m) = H(m*)**.

Messaggi di questo tipo esistono sicuramente ed occorre quindi valutare attentamente la complessità computazionale del seguente attacco: si sceglie a caso **m***, si calcola **H(m*)** e si confronta il risultato con **H(m)**, ripetendo il procedimento fino a quando non si ha successo.

Poniamoci nel caso ideale del *random oracle* (v. pag. 15). Con questa ipotesi, la probabilità di successo di un solo tentativo è 2^{-n}, quella di insuccesso $1 - 2^{-n}$.

Dopo **k** tentativi la probabilità di successo è data dal teorema binomiale:

$$P_1(2^n, k) = 1 - (1 - 2^{-n})^k = k \times 2^{-n} - k \times (k-1) \times 2^{-2n}/2 + k \times (k-1) \times (k-2) \times 2^{-3n}/6 - \text{..}$$ ecc.

Se **n** è grande, 2^{-n} è molto piccolo e ci si può dunque arrestare al primo termine:

$$P_1(2^n, k) \cong k \times 2^{-n}$$

Fissata una desiderata probabilità di successo, il numero **k** di prove da fare è proporzionale a 2^n. L'algoritmo d'attacco è dunque esponenziale nella dimensione **n** dell'impronta: **O(exp(n))**. Per fronteggiarlo, la dimensione dell'impronta è progressivamente aumentata.

ESEMPI – Fino ad una decina di anni fa l'algoritmo di hash più usato è stato **MD5** (R. Rivest,1991), che impiega 512 bit di blocco e 64 passi di elaborazione per calcolare 128 bit d'uscita.

Successivamente, a seguito dell'individuazione di vulnerabilità in MD5 e comunque per aumentare la robustezza a fronte di attacchi *birthday* (v. pag. 34), è stato impiegato **SHA-1** (NIST, 1994, revisione del precedente standard SHA): la funzione di compressione usa 512 bit di blocco e 80 passi per calcolare un'impronta di 160 bit.

In ambito europeo ha trovato impiego anche **RIPEMD-160** (1996, revisione di RIPEM-128 del progetto europeo RIPE): la funzione di compressione usa 512 bit di blocco e 160 passi (o meglio 80 con parallelismo 2) per calcolare 160 bit d'uscita.

Nel 2005 è stato evidenziato che anche questi algoritmi sono attaccabili più facilmente di quanto previsto. L'attenzione degli sviluppatori di applicazioni sicure si è quindi spostata su **Tiger** (Anderson e Biham, 1996), che ha un'impronta di 192 bit, e su **SHA-256, -384, -512** (2002, NIST), strutturalmente simili a SHA-1, ma ancora più robusti alle collisioni avendo rispettivamente 256, 384 e 512 bit d'uscita. In Europa si sta attualmente perfezionando e valutando **Whirlpool**, che fornisce impronte di 512 bit.

Nel 2006 il NIST ha indetto una gara pubblica per individuare un nuovo standard, detto SHA-3. Nel 2011 conosceremo l'algoritmo vincitore.

3 – Resistenza forte alle collisioni

Il controllo dell'autenticità di un messaggio mediante **firma digitale** (v. pag. 17) impone di studiare un secondo contesto in cui l'attaccante può scegliere a caso anche **m**. Un impostore può, infatti, avvalersi di un famoso paradosso statistico.

Birthday paradox – *Nell'ipotesi che le date di nascita sono equiprobabili*, *è sufficiente scegliere a caso 253 persone per avere una probabilità maggiore di 0,5 che una di queste compie gli anni in un giorno prefissato. Sono invece sufficienti 23 persone se si vuole che due o più di queste compiano gli anni nello stesso giorno.*

Vediamo le conseguenze del paradosso nel caso di una funzione hash ideale con **n** bit d'uscita.

Vogliamo calcolare la probabilità $P_2(2^n, k)$ di ottenere almeno due valori d'uscita identici immettendo in ingresso, per **k** volte ($k \leq 2^n$), un valore scelto a caso.

Le sequenze così generabili sono $(2^n)^k$; quelle con valori tutti diversi sono $2^n! / (2^n - k)!$.

La probabilità di avere una stringa con almeno due simboli identici è dunque:

$$P_2(2^n, k) = 1 - 2^n! / ((2^n)^k \times (2^n - k)!) = 1 - (1-1/2^n) \times (1-2/2^n) \times .. . \times (1-(k-1)/ 2^n)$$

Per avere una formulazione più chiaramente interpretabile, ricorriamo alla disuguaglianza $(1-x) \leq e^{-x}$, che è valida per $x \geq 0$ e che evidenzia tra l'altro una buona approssimazione di **1-x**, quando **x** ha valori piccoli.

Dopo qualche passaggio[15], tenendo anche conto che **k** è molto grande, si ottiene:

$$P_2(2^n, k) \cong 1 - \exp(- 2^{-n} \times k^2/2)$$

Esplicitando per **k** si ha:

$$k = \mathrm{sqrt}(2 \ln(1/(1-P_2))) \times \mathrm{sqrt}(2^n)$$

ESEMPIO – Nel caso del paradosso, bisogna mettere **365** al posto di 2^n. Posto $P_2 = 0{,}5$ si ottiene:
$k = \mathrm{sqrt}(2 \times \ln(2)) \times \mathrm{sqrt}(365) = 22{,}54$.

Una volta fissata la desiderata probabilità di successo, l'individuazione di due coincidenze qualsiasi richiede un numero di tentativi proporzionale a $2^{n/2}$: l'algoritmo è quindi difficile, ma, a parità di **n**, enormemente più facile di quello che individua una collisione con un prefissato valore di hash.

Ritorniamo ora al problema di sicurezza posto dalla firma digitale.

L'impossibilità pratica di individuare una collisione (R19) ha portato a considerare l'impronta un sicuro **elemento identificativo del messaggio** da cui è stata calcolata e ad autenticare solo questa piccola stringa, a tutto vantaggio dell'efficienza. L'uso della coppia di trasformazioni **S, V** consente, infatti, al destinatario di rilevare e di scartare ogni messaggio generato dall'intruso.

C'è però ancora una minaccia: un mittente disonesto potrebbe voler disporre di **due messaggi con la stessa firma**, per poter in un secondo tempo sostenere di aver comunicato quello che gli conviene di più.

A tal fine il malintenzionato deve condurre il seguente attacco:

birthday attack:
1. genera $2^{n/2}$ messaggi uno diverso dall'altro, ma tutti ottenuti apportando lievi modifiche al primo messaggio;
2. calcola e memorizza l'hash di tutte queste versioni;
3. introduce lievi modifiche al secondo messaggio, calcola l'hash e controlla se è presente in memoria; in caso negativo ripete il passo 3: per il paradosso del giorno di compleanno può aspettarsi di trovare una coincidenza dopo $2^{n/2}$ iterazioni.

ESEMPIO – Nel sito del corso è disponibile un programma che consente di condurre un attacco birthday su un'impronta di piccole dimensioni (minore di 40 bit).

Si noti che il malintenzionato potrebbe essere anche il verificatore. Una volta ricevuto un messaggio autentico, con l'attacco del giorno del compleanno egli può, infatti, crearne uno di diverso significato e di uguale impronta ed escogitare un trucco (esiste e lo vedremo) per farsi firmare dal corrispondente anche questa versione.

Il rispetto di R20 richiede dunque funzioni hash crittografiche con **un numero di bit d'uscita due volte più grande del livello di sicurezza** che si vuole ottenere.

Attualmente, come si è già detto, si usano impronte di 160 bit, ma sarà opportuno passare al più presto ad algoritmi che forniscono 256 o, meglio, 512 bit d'uscita.

4 – Unidirezionalità

L'**impossibilità di inversione** è richiesta alle funzioni hash, quando sono impiegate per proteggere la segretezza di un dato di cui è resa nota la sola impronta.

Un caso tipico in cui H deve essere *one-way* si ha quando un messaggio in chiaro m è autenticato da H(m||s), come abbiamo già segnalato a pag. 18.

Se R21 non fosse vera, anche lo schema di firma digitale con appendice avrebbe una sottile vulnerabilità. L'intruso I sceglie un numero a caso **r** e calcola $\mathbf{x = V(r)}$, ove **V**, non segreta, è la trasformazione di verifica della firma di un certo utente U. A questo punto I calcola $\mathbf{y = H^{-1}(x)}$ e si costruisce così una coppia **y, r** la cui paternità non potrebbe essere ripudiata da U. Lasciando perdere il fatto che **y** quasi sicuramente è privo di significato, se U avesse voluto autenticarlo, avrebbe proprio ottenuto come etichetta $\mathbf{r = s(H(y))}$, ove **S** è la trasformazione che per ipotesi solo lui deve saper fare.

Il più efficace algoritmo di inversione deve dunque avere complessità $\mathbf{O(\exp(n))}$. Tale algoritmo esiste per qualsiasi funzione hash, appartiene alla categoria degli attacchi con forza bruta e richiede di provare uno dopo l'altro i possibili valori di **x**, fino a quando non si trova quello che ha l'hash desiderato.

[15] v. [6], Appendice 11.A; dal sito del Corso è scaricabile il notebook Birthday.nb sviluppato da Francesca Nocerino.

2.4 Servizi d'identificazione

Esistono due differenti modi per dare una dimostrazione di conoscenza. L'identificazione è detta

- **passiva**, o **debole**, quando non sono richieste elaborazioni da parte dell'identificando: al passo 3 del protocollo di pag. 19 egli deve, infatti, comunicare soltanto il **segreto S** che ha concordato con il verificatore durante la registrazione;
- **attiva**, o **forte**, quando l'identificando deve prendersi carico di calcolare e di comunicare un dato **sempre diverso** ed **imprevedibile** per farsi riconoscere.

2.4.1 Identificazione passiva

L'identificazione tramite comunicazione di una password è oggi di gran lunga il metodo più usato per controllare l'accesso ad un servizio informatico. Il protocollo è semplicissimo:

1. l'utente trasmette in chiaro il suo identificativo **ID** e la password **PSW** che ha imparato a memoria;
2. la macchina estrae da un suo **file delle password** quella che ha concordato con **ID** e confronta i due dati.

Caratteristica dell'identificazione passiva è dunque quella di far trasmettere all'identificando **sempre lo stesso dato**. L'attacco più ovvio per l'intruso è **intercettare** la password comunicata da un utente legittimo e **replicare** il messaggio quando vorrà a sua volta avere libero accesso al sistema.

Il mezzo di comunicazione delle password deve (o dovrebbe) dunque essere **a prova di intercettazione**. Qualche forma di difesa preventiva è comunque sempre presente.

ESEMPI – L'introduzione della password in un calcolatore non genera "eco" sul monitor, ma solo asterischi o spazi bianchi. Negli sportelli Bancomat la tastiera è incassata e gli utenti sono invitati a guardarsi alle spalle prima di digitare il loro PIN; il canale di trasmissione non è inoltre fisicamente accessibile dall'esterno.

Si noti che se l'utente cifrasse la password prima di trasmetterla (ma non può farlo, perché nell'identificazione passiva si presuppone che l'identificando non disponga di sue risorse di elaborazione), la pericolosità dell'attacco con intercettazione e replica non si ridurrebbe: è sicuramente vero che il testo cifrato intercettato dall'intruso non gli consente di individuare il segreto, ma ciò non gli impedisce certo di ripresentarlo tale e quale e di essere poi riconosciuto come il vero proprietario del segreto.

L'intruso ha inoltre due altre vulnerabilità da sfruttare, **la memoria dell'utente e quella della macchina**.

Cominciamo dalla **memoria dell'utente**. La password dovrebbe rispettare R12, ma in realtà l'utente che la deve imparare a memoria è spesso portato a scegliersi una stringa "corta" e "poco casuale".

L'intruso che tira ad indovinare ha dunque buone chances di riuscirci.

Una difesa potrebbe essere quella di attribuire al sistema il compito di scegliere lui password lunghe e casuali ma questo accorgimento, comunque poco gradito, si scontrerebbe poi con la difficoltà di impararle a memoria. La soluzione usuale è dunque quella di far giudicare al sistema se la password (o meglio la pass phrase di cui abbiamo già parlato e che rivedremo tra poco) scelta dall'utente è sufficientemente robusta. A tal fine sono in uso metodi interattivi[16] e regole pratiche.

- R22: "*una buona password è formata da almeno otto simboli dotati di qualche caratteristica di casualità (lettere minuscole e maiuscole alternate da cifre e da simboli di operazione/interpunzione)*".

Purtroppo R22 può non bastare. In generale, infatti, l'utente non si fida della sua memoria e si affretta quindi a trascrive la password su un **foglietto di "back-up"**: questo è un ulteriore punto di vulnerabilità ed è dunque indispensabile che l'utente adotti ulteriori provvedimenti per non farlo cadere in cattive mani.

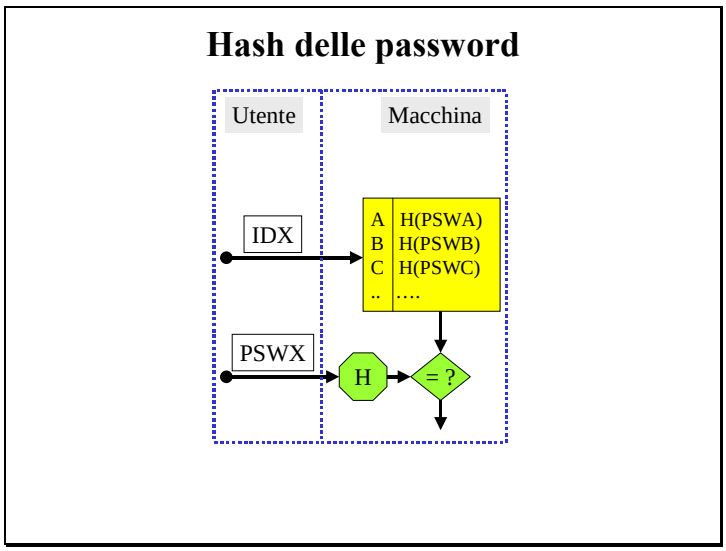
ESEMPI – Pessima è la consuetudine di apporre un foglietto con la password sul monitor del PC.
Parimenti pericoloso è inserire nel portafoglio il foglietto con il PIN e la scheda del Bancomat.

Prendiamo ora in considerazione la **memoria del sistema**. Se l'intruso riesce ad accedere al file delle password, tutto il controllo d'accesso va a farsi benedire! Chi è riuscito ad intrufolarsi può, infatti, autonominarsi utente legittimo, aggiungendo in coda al file il suo ID, l'hash della sua PSW e tutte le autorizzazioni che gli servono.

La modifica del file non è tra l'altro strettamente necessaria: basta, infatti, che l'intruso riesca a leggerlo (anche nella meno protetta versione di back-up). All'intruso, infatti, non interessa impersonare un utente specifico: gli preme solo di ottenere le credenziali di un utente qualsiasi (tanto meglio se sono quelle dell'amministratore).

Le regole R14 e R15 enunciate a pag. 27 devono dunque essere rispettate, con la sola precisazione che in questo caso il proprietario dei dati segreti è l'amministratore del sistema.

[16] Si veda in proposito come opera il PGP nella fase di installazione



Un accorgimento interessante per la riservatezza è quello di imporre al sistema l'uso dell'impronta della password come termine di paragone**.** Si noti che il calcolo dell'hash è, a ben vedere, **una forma di cifratura per la quale non esiste la decifrazione**.

All'atto della registrazione dell'utente **X**, il sistema memorizza:

IDX, H(PSWX).

Al momento della verifica, il sistema sottopone alla stessa trasformazione la prova d'identità ricevuta, estrae dalla memoria il termine di paragone corrispondente all'identificativo parallelamente dichiarato dall'identificando e confronta i due dati.

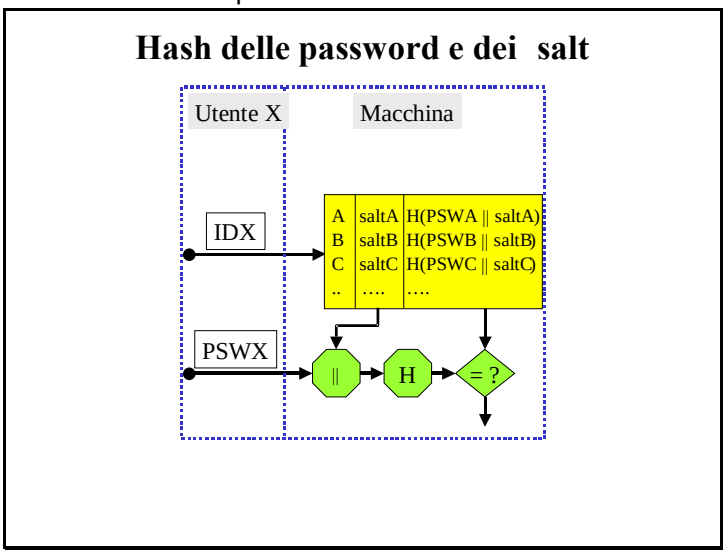
La non invertibilità (R21) impedisce, non solo all'intruso ma anche all'amministratore del sistema, di risalire da H(PSW) a PSW.

In questa maniera l'utente è veramente l'unico a conoscere il segreto che lo identifica.

La Crittanalisi ha però evidenziato che l'ostacolo può essere aggirato, se gli utenti hanno scelto password facili da ricordare: l'attaccante che è riuscito ad impadronirsi del file delle password può, infatti, calcolare l'hash di tutte le possibili varianti (permutazioni delle lettere, caratteri minuscoli e maiuscoli) delle voci di un dizionario (opportunamente ampliato con nomi di località, con nomi propri o di personaggi mitologici, ecc.) e controllare poi se c'è qualche coincidenza. Si parla in questo caso di **attacco con dizionario**.

ESEMPIO - Analisi di casi reali hanno mostrato che in media il 25% delle password di utenti registrati presso Centri di Calcolo può essere individuato da un attacco con dizionario.

All'attacco con dizionario è vulnerabile anche la passphrase (v. pag. 27); per renderlo più oneroso è usuale iterare la compressione molte volte



Il sistema operativo **Unix** prevede invece di aggiungere un numero casuale (detto *salt*) ad ogni nuova password e poi calcola e memorizza l'hash del tutto, unitamente al salt in chiaro.

Al momento della verifica, il sistema accede al file con chiave ID, estrae il salt, lo concatena alla password ricevuta, calcola l'hash e confronta il risultato con il termine di paragone in suo possesso.

In questa maniera lo spazio dei valori possibili è molto più ampio di quello dei valori in uso: l'aggiunta di un salt casuale di **p** bit prima del calcolo del hash genera uno tra $\mathbf{2^p}$ differenti termini di paragone di una stessa password e chi vuole condurre un attacco con dizionario al file delle password si trova così obbligato a provarli tutti per ogni voce.

Un'ultima osservazione sulla debolezza intrinseca dell'identificazione passiva: l'utente deve essere **certo a priori** che il verificatore è veramente quello da cui vuole farsi riconoscere.

Si potrebbe pensare di eliminare questo vincolo ricorrendo ad un **protocollo** in cui dapprima uno identifica l'altro e poi quest'ultimo identifica il primo. Purtroppo questo accorgimento è completamente insicuro.

ESEMPIO – **A**, dopo aver indirizzato il sito di **B**, presenta le sue credenziali e si mette in attesa di quelle di **B**: se un intruso è riuscito a dirottare sul suo calcolatore i messaggi destinati a **B** non solo si guarderà bene dal rispondere, ma potrà successivamente spacciarsi per **A** ed ottenere dal vero **B** tutto quello che gli serve (si racconta che un attacco di questo tipo sia stato realmente portato da un terminale Bancomat "fasullo").

2.4.2 Identificazione attiva

La vera contromisura all'attacco con intercettazione e replica è il prevedere che la prova d'identità sia **continuamente cambiata**; vedremo che ciò consente anche l'identificazione reciproca.

Sono stati individuati tre differenti principi su cui basare l'identificazione attiva:

1. la password usata una volta sola (*one-time password*),
2. il protocollo a sfida/risposta (*challenge/response*),
3. la prova d'identità a conoscenza zero (*zero-knowledge identity proof*).

Tutti cercano di soddisfare la seguente regola di sicurezza:

- R23:*"il calcolo della prova di identità da fornire di volta in volta deve essere facile per chi conosce un'informazione segreta, difficile per chi dispone solo delle prove inviate in precedenza*".

I primi due metodi sono i più usati, realizzando un buon compromesso tra sicurezza ed efficienza; il terzo è il più sicuro, ma anche il più complesso ed è per questo poco impiegato.

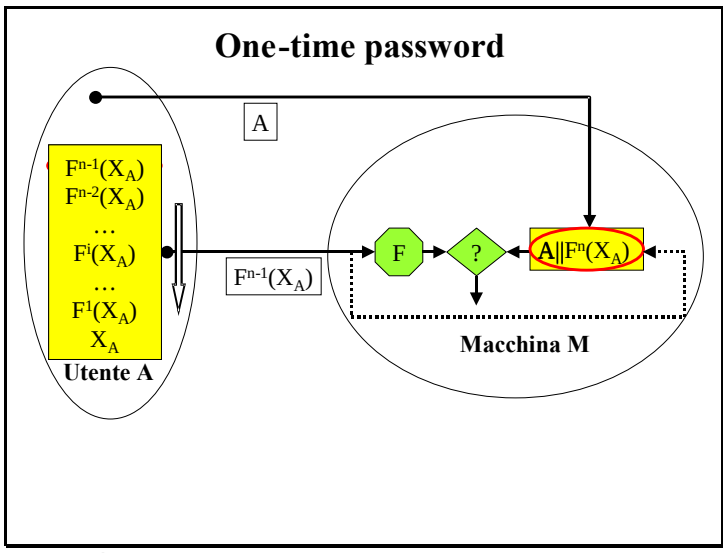
1 – La password usata una volta sola

Il metodo della **password impiegata una sola volta** può basarsi sull'uso o di una **funzione unidirezionale**, o di un **cifrario con chiave di sessione**.

Funzione unidirezionale

L'uso della funzione unidirezionale è stato proposto all'inizio degli anni '80 [17] ed è oggi oggetto di attenta riconsiderazione.

Il principio è di assegnare all'utente una **sequenza di password** da consumare una dopo l'altra.



In fase di registrazione **A** sceglie a caso un numero X_A ed impiega una funzione non invertibile **F** (ad esempio una funzione hash sicura) per calcolare la sequenza di valori:

$$X_A, F(X_A), F^2(X_A), F^3(X_A) \ldots F^{n-1}(X_A)\ F^n(X_A)$$

ove F^i indica l'i-esima applicazione di **F**.

I primi **n** dati sono memorizzati da **A** in ordine inverso. Nella memoria della macchina **M** viene inserito solo $F^n(X_A)$.

Alla prima connessione **A** invia

$$\mathbf{Prova(1) = F^{n-1}(X_A)}$$

ed il sistema calcola **F(Prova(1))**.

Se il risultato è proprio $F^n(X_A)$, **A** è identificato e la one-time password che **M** ha ricevuto è memorizzata al posto del precedente termine di paragone; A invece la depenna dalla sua lista, per poter usare il secondo dato come **Prova(2)**.

Il comportamento di **A** e di **M** si ripete identico fino a quando il primo non invia X_A, cosa che impone di rinnovare l'accordo sul segreto.

Una perdita di sincronismo, causata ad es. da un crash di **M**, può essere agevolmente recuperata.

L'ipotesi di non invertibilità della F rende del tutto inutile sia l'intercettazione delle prove già inviate, sia il furto del termine di paragone memorizzato da **M**.

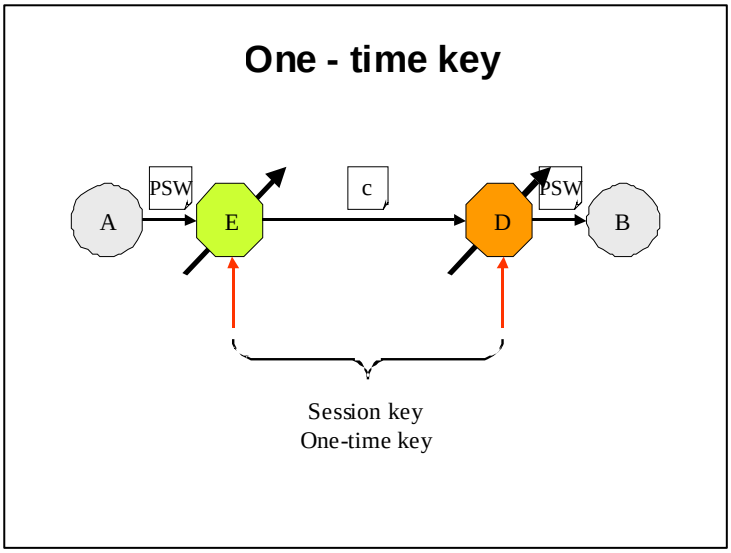
Cifrario con chiave di sessione

A pag. 36 si è osservato che la trasmissione di una password cifrata non impedisce l'attacco con replica.

Questo non è più vero se l'identificando ed il verificatore, all'inizio di ogni loro **sessione** di lavoro, concordano in segreto una sempre diversa chiave di cifratura: la prova d'identità è di conseguenza **sempre diversa** e non serve a nulla intercettarla e replicarla.

Una chiave di cifratura sempre diversa è detta **chiave di sessione** (*session key*, *one-time key*).

[17] L. Lamport, "Password Identification with Insecure Communications", *Communications of the ACM*, v.24, n.11, Nov. 1981, pp 770-772



In figura la continua modifica della chiave è stata graficamente evidenziata con due frecce oblique sui blocchi E, D.

In questo caso sia **A** che **B** hanno il vantaggio di poter memorizzare solo la PSW che hanno inizialmente concordato.

La continua modifica della trasformazione **D** consente infatti a B di accettare per buone solo le prove d'identità "fresche", cioè prodotte da un A che modifica parallelamente la trasformazione **E** operata sulla PSW.

L'uso del Cifrario con chiave di sessione **trasforma** dunque il metodo di identificazione tramite password da **statico** a **dinamico**, ma questo solo al momento in cui la prova d'identità deve essere trasferita sul canale insicuro.

Può però venire il dubbio che la modifica, continua ed in segreto, delle due trasformazioni costituisca un carico di lavoro eccessivo richiesto ai due corrispondenti.

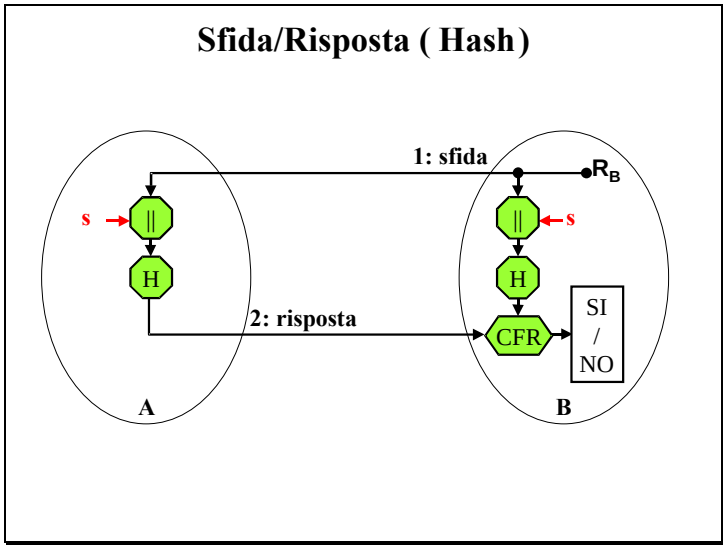
Non è così e lo vedremo: tale delicata incombenza può essere presa totalmente in carico dai **servizi di rete**, senza richiedere alcun intervento diretto da parte degli utenti.

ESEMPIO – All'inizio di ogni collegamento, i protocolli di rete **SSL** e **IPv6** aiutano i due terminali a concordare in segreto la chiave di sessione che impiegheranno per cifrare ogni loro successiva comunicazione.

2 – Il protocollo a sfida/risposta

Il metodo d'identificazione attiva oggi più usato è il protocollo **a sfida e risposta** (*challenge/ response*)[18]; per eseguirlo i due corrispondenti possono avvalersi

- o di **funzioni hash** sicure,
- o di un **cifrario**, simmetrico o asimmetrico,
- o di uno schema di **firma digitale**.



Nel primo caso, molto usato per la sua efficienza, **A** e **B** scelgono una funzione **H** sicura e concordano un segreto **s**.

Quando **A** chiede a **B** di essere identificato inizia l'esecuzione del seguente protocollo a tre passi:

1. **B:** invia ad **A** un dato di sfida R_B che non ha mai impiegato prima (tale numero usato una sola volta è detto **nonce**);
2. **A:** calcola $c = H(R_B \,||\, s)$ e trasmette **c** come risposta alla sfida;
3. **B:** calcola $c' = H(R_B \,||\, s)$ con i dati a sua disposizione ed esamina se $c' = c$.

Se si vuole un'identificazione **mutua** occorrono due dati di sfida (R_A, R_B):

1. $B \rightarrow A: R_B$;
2. $A \rightarrow B: c_A = R_A \,||\, H(R_B \,||\, s)$;
3. $B \rightarrow A: c_B = H(R_A \,||\, s)$.

ESEMPIO – Nei **telefoni cellulari**, **B** è la stazione della rete fissa che serve la cella in cui si trova l'utente mobile **A**.

Il segreto di **A** è un numero di 128 bit, memorizzato nella smart card, detta SIM, contenuta all'interno del suo apparato mobile. Anche l'Ente che ha registrato l'utente **A** (HLR o *Home Location Register*) conosce **s**.

Per consentire l'identificazione di A, ovunque esso sia, e per difendere la riservatezza di s, HLR si preoccupa di inviare in modo sicuro una coppia **R** e $H(R \,||\, s)$ ad ogni B che lo richiede.

In realtà, per rendere sicura l'identificazione reciproca, occorre usare un protocollo più complesso:

[18] All'inizio degli anni '50 i militari americani usavano una tecnica similare per distinguere con il radar, durante una battaglia aerea, gli aerei "amici" da quelli "nemici".

1. **B→A: R_B**;
2. **A→B: $c_A = R_A \| H(R_A \| R_B \| B \| s)$**;
3. **B→A: $c_B = H(R_A \| R_B \| A \| s)$**.

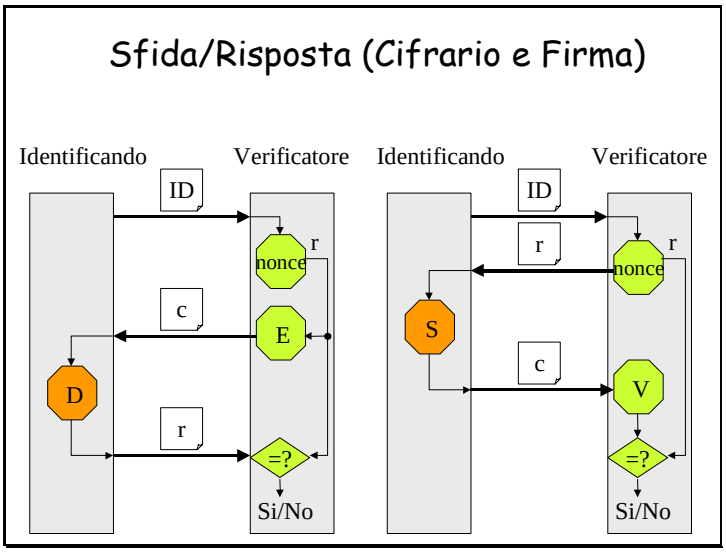
La giustificazione è fornita dal seguente problema.

Il problema del Gran Maestro di scacchi – *Il signor A vuole spacciarsi per un grande esperto di scacchi pur non conoscendo il gioco. A sfida due Gran Maestri B e C, che sistema, senza che se ne accorgano, in due camere contigue: a B assegna i "bianchi", a C i "neri". Preso nota della prima mossa di B, A corre nell'altra stanza e la riproduce sulla scacchiera di C. Successivamente prende nota della contromossa di C e corre a riprodurla sulla scacchiera di B. Continuando così ottiene o due patte o un'incredibile vittoria.*

Bisogna dunque impedire all'intruso di intromettersi nel protocollo, spacciandosi per **A** con **B** e per **B** con **A**: l'inserzione dell'identificativo del destinatario nelle risposte è un primo utile accorgimento difensivo.

Bisogna anche non concedere all'intruso il tempo per svolgere contemporaneamente due protocolli: a tal fine al posto, o a fianco, di $\mathbf{R_A}$ e di $\mathbf{R_B}$ sono spesso impiegate delle **marche temporali** (*time-stamp*) per controllare quanto tempo intercorre tra il lancio della sfida e l'arrivo della risposta.

Un ulteriore accorgimento è quello di numerare le identificazioni tra i due interlocutori.



Per lanciare la sfida e controllare poi la risposta, può essere impiegato anche un **Cifrario**, o uno **Schema di firma**.

Nel primo caso (v. lo schema sulla sinistra in figura) il verificatore cifra il nonce, si mette in attesa della sua decifrazione e confronta infine il risultato con il dato in suo possesso.

Nel secondo caso (v. lo schema sulla destra) il verificatore trasmette il nonce in chiaro, si mette in attesa della sua firma e poi la verifica.

La **segretezza** della trasformazione **D** nel primo caso e della trasformazione **S** nel secondo impediscono all'intruso di rispondere correttamente alla sfida. La continua variazione di **r** rende inutile intercettazione e replica. In entrambi i casi è comunque utile il time-stamp.

3 – La prova d'identità con conoscenza zero

Il protocollo challenge/response ha un difetto non eliminabile: qualsiasi sia la trasformazione impiegata, il dato segreto che identifica un'entità deve partecipare alla formazione della risposta e non può quindi mai essere perfettamente protetto.

Per porre rimedio anche a questa vulnerabilità sono stati studiati i **protocolli** "**a conoscenza zero**", che richiedono alla parte che deve essere identificata di dare solo una **prova**, o **testimonianza**, della conoscenza del segreto senza fornire all'altra parte **nulla di più** di quello che serve per verificarla.

Nei protocolli a *zero knowledge* l'identificando dichiara di saper risolvere un **problema difficile** (per il quale però esiste un algoritmo di verifica con tempo polinomiale) e lo dimostra al verificatore facendogli vedere di essere in grado di risolvere un diverso ed isomorfo problema difficile che ha in precedenza concordato con una **terza parte fidata**.

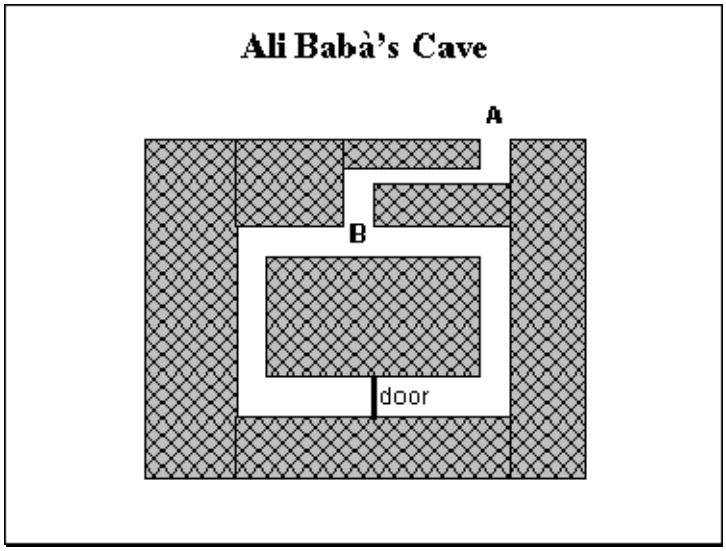
L'interazione tra i due corrispondenti prevede tipicamente tre passi:

1. l'identificando fornisce una **testimonianza** di quello che asserisce di saper fare;
2. il verificatore gli lancia una **sfida** che consente di giudicare la veridicità della testimonianza;
3. l'identificando invia la **risposta** alla sfida ed il verificatore controlla se è corretta.

Al termine dei tre passi può esserci il dubbio che l'identificando abbia dato la risposta corretta "tirando ad indovinare": per rendere piccola la probabilità di questo evento i tre passi devono essere **ripetuti più volte**.

Più avanti (v. 5.6.2) esamineremo con un certo dettaglio il protocollo che ha aperto questo importante filone di ricerca.

Per ora dobbiamo dunque accontentarci di un aneddoto abbastanza ben rappresentativo di una prova a conoscenza zero[19].



La caverna di Alì Babà – *Una caverna è formata da due tunnel separati da una porta, che può essere aperta solo usando una frase segreta. Alì Babà vuole convincere il califfo Omar di essere a conoscenza del segreto senza rivelargli quale è.*

A tal fine Alì Babà dice al califfo di restare nel punto A ed entra in uno dei due tunnel. A questo punto Omar avanza fino al punto B e chiede ad Ali di uscire o dal tunnel posto alla sua destra o da quello a sinistra.

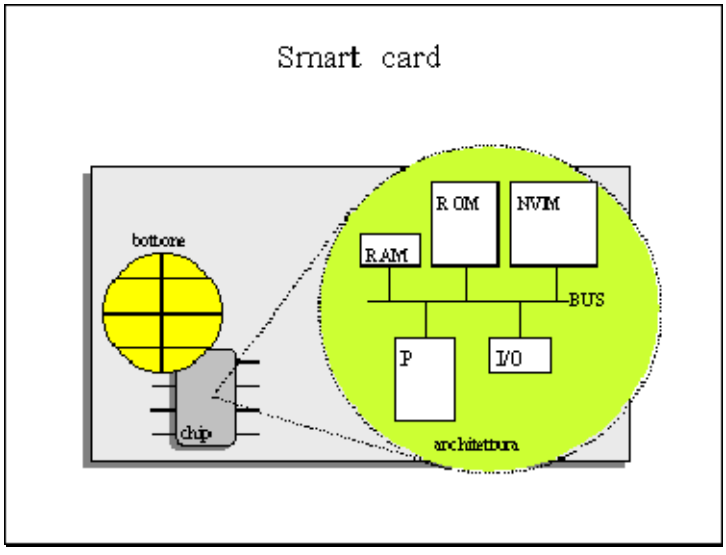
Alì ci riesce, aprendo se necessario la porta con la frase segreta. Un impostore ha solo il 50% di probabilità di trovarsi già nel tunnel richiesto e se la prova è ripetuta non potrà, prima o poi, trovarsi dalla parte giusta.

*Dopo **n** positive ripetizioni della prova la probabilità che Alì sia un impostore è **2^{-n}**.*

2.4.3 Smart Card

L'archivio dei segreti personali è ancora più sicuro se ogni utente si può portare dietro il suo: la **smart card**, una piccola scheda di plastica avente a bordo il chip di un micro-calcolatore, ha proprio questa funzione.

ESEMPI – I telefoni cellulari contengono una smart card rimovibile, in cui sono alloggiati i dati del proprietario.
Molte applicazioni di home banking identificano l'utente remoto tramite una smart card.
I docenti della nostra Università impiegano una smart card per firmare e registrare i verbali d'esame.
E' previsto che nel 2009 (ma più realisticamente sarà il 2011) tutti i cittadini italiani avranno una carta d'identità elettronica (www.innovazione.gov.it/).



Il chip è dotato di **controllo d'accesso** (per attivarlo occorre digitare un PIN), di **dispositivi anti-intrusione** (i dati segreti sono distrutti se qualcuno tenta di condurre un'analisi della struttura interna) e di **risorse** (memoria, processore, I/O).

Nelle cosiddette **carte a contatto** un bottone metallico, suddiviso in otto sezioni elettricamente isolate una dall'altra, consente ad un apposito terminale (*card reader/writer*) di alimentare il chip sottostante e di fargli scambiare dati con l'esterno.

La scheda non è quindi alimentata se non quando è introdotta nel terminale.

Caratteristica peculiare del chip è dunque la presenza al suo interno di 5-10 KB di **memoria non volatile** (EEPROM, FLASH), dedicata a mantenere i dati segreti dell'utente.

La *NV-Memory* è gestita da un **sistema operativo** alloggiato nella ROM (10-16 KB) e selettivamente attivato da comandi provenienti dal lettore o, suo tramite, da un calcolatore esterno; nella memoria a sola lettura sono alloggiati anche i programmi che consentono al **µP** (inizialmente un semplice CISC a 8 bit, poi un CISC affiancato da un Coprocessore crittografico ed attualmente un potente RISC a 32 bit) di eseguire complesse funzioni di sicurezza.

[19] J.J. Quisquater, L. Guillou, T. Brenson: "How to explain zero-knowledge protocols to your children" Advances in Cryptology- CRYPTO 92 Proceedings, Springer-Verlag 1993, pp.31-53.

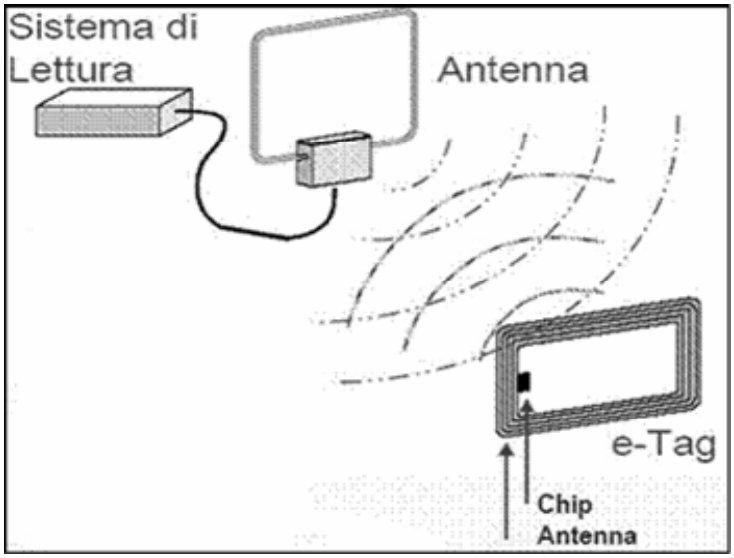
Il prezzo di vendita dipende dalla quantità, ma è comunque molto basso (5-10 $). I volumi di produzione sono sempre più grandi (200 milioni di schede nel solo anno 2000).

ESEMPIO – Qualsiasi cittadino italiano può ottenere oggi , con una spesa di circa 25 euro, una smart card (detta Carta Servizi) che gli consente di essere identificato da tutti i fornitori di servizi pubblici e privati.

Sono in commercio anche carte ***contact-less*** (che devono essere solo avvicinate al terminale) e carte ***proximity*** (che impiegano una piccola antenna per poter comunicare a distanza con il loro terminale).

Smart card dello stesso tipo e di Costruttori diversi sono oggi **interoperabili**: a tal fine sono stati definiti diversi standard "de iure" e "de facto" (ISO 7816, Microsoft Crypto API, PKCS#11 e #15, PC/SC).

Smart Tag



Anche gli oggetti "fisici" devono poter essere univocamente identificati dalle macchine che svolgono servizi nella Società della informazione. A tal fine è in uso la tecnologia **RFID** (*Radio Frequency Identification*), basata su un microcircuito che comunica via radio ed in serie i bit di un'**etichetta** posta al suo interno.

Un apparato, detto *reader*, genera un campo elettromagnetico intorno a lui.

Un chip inserito nell'oggetto e dotato di antenna, riflette o no il segnale del reader secondo il valore dei bit della sua etichetta.

Il lettore rileva il fenomeno, estrae la stringa di bit e la comunica ad un server.

Quest'ultimo accede ad un database ed associa all'oggetto identificato tutti i dati che possono servire per sottoporlo ad uno o più servizi.

Esiste uno standard per la codifica dei dati inseriti in un'etichetta (**EPC** da *Electronic Product Code*). Le applicazioni, sperimentate o ipotizzate, sono moltissime.

ESEMPI – Ogni prodotto prelevato da un bancone di un supermercato potrebbe essere rapidamente sostituito da uno di magazzino; il suo costo potrebbe inoltre aggiornare on-line il conto della spesa contenuta in un carrello e facilitare il pagamento automatico del tutto. Si potrebbe sapere in ogni momento quanti soldi abbiamo nel portafoglio, o dove sono le mucche al pascolo, o qual'è la classifica provvisoria di una maratona.

Molta perplessità sollevano le applicazioni che prevedono di applicare i chip all'uomo o a suoi oggetti personali (ad. esempio il passaporto). Un lettore può infatti ottenere informazioni riservate senza il consenso del loro proprietario; la violazione della privacy è ancora più pericolosa se è stato un avversario a predisporre il lettore.

Per fronteggiare queste minacce si stanno sperimentando tecniche di autenticazione e di cifratura.

ESEMPIO – Molte recenti ed innovative applicazioni nei settori degli armamenti, della produzione industriale e della salute si basano sull'uso di sensori mobili che comunicano via radio la loro identità ed i dati che hanno raccolto (*sensor networks*). Tecniche efficienti di autenticazione mediante crittografia a chiave pubblica sono già state, ad esempio, messe a punto dalla Ditta Certicom (www.certicom.com).