Autenticazione

A cura di

Mara Corona

Francesco Scarano

Ilaria Scarano

Anno Accademico 2004-2005

Introduzione

E' possibile trattare il problema dell'autenticazione seguendo svariati punti di vista. Quello che a noi è sembrato più chiaro è stato quello che suddivideva tale problema in due sottoproblemi:

- la message authentication;
- l'entity authentication.

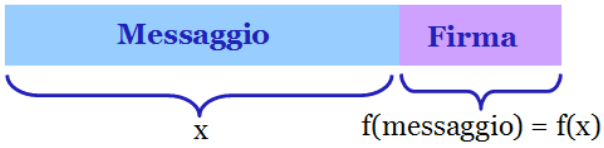
Vediamo quindi quali sono le soluzioni proposte per garantire queste due tipi di autenticazione.

1. Message Authentication

Con il termine autenticazione del messaggio andiamo ad intendere la possibilità di verificare che un messaggio non sia stato modificato lungo la sua trasmissione sul canale di comunicazione. Più in dettaglio possiamo dire che tale tipo di autenticazione permette, da un lato la verifica di colui che ha originato il messaggio e, dall'altro, l'integrità del messaggio stesso.
Per risolvere questo problema vengono presentate le tre soluzioni più diffusamente utilizzate.

1.1 Firma digitale (Cenni)

La firma digitale è il risultato di algoritmi molto complessi o, dal punto di vista dell'utente una funzione generata da un programma.
L'idea che c'è alla base è simile a quella di una normale firma, anche se quest'ultima è molto semplice da contraffare, dove un dato messaggio x è caratterizzato da un'unica firma digitale che è funzione del messaggio e attaccata da esso.

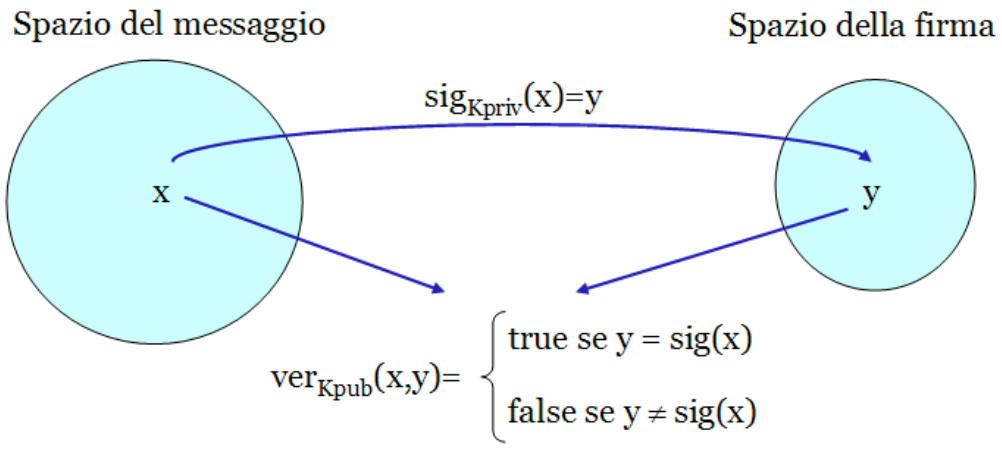


La principale differenza tra firma autografa e firma digitale sta nel fatto che la prima è direttamente riconducibile all'identità di colui che la appone, poiché la calligrafia è un elemento identificativo della persona, mentre la seconda non possiede questa proprietà.

Per coprire questa deficienza si ricorre *all'autorità di certificazione*[1], il cui compito è quello di stabilire, garantire e pubblicare l'associazione tra firma digitale e soggetto che l'ha apposta.
Per contro, mentre l'associazione tra il testo di un documento e la firma autografa è ottenuta esclusivamente attraverso il supporto cartaceo, la firma digitale è intrinsecamente legata al testo a cui è apposta, tanto che i due oggetti possono essere fisicamente separati senza che per questo venga meno il legame esistente tra loro. Conseguenza di ciò è l'unicità della firma digitale, nel senso che a testi diversi corrispondono firme diverse e quindi, nonostante la sua perfetta replicabilità, è impossibile trasferirla da un testo ad un altro.

I meccanismi di firma digitale poggiano essenzialmente su algoritmi crittografici a chiavi pubbliche, che sono detti anche a chiavi asimmetriche poiché utilizzano chiavi diverse per le operazioni di cifratura e decifratura.
Un calcolo statistico ha dimostrato che è 600 milioni di volte più difficile contraffare la firma elettronica rispetto alla tradizionale firma autografa.

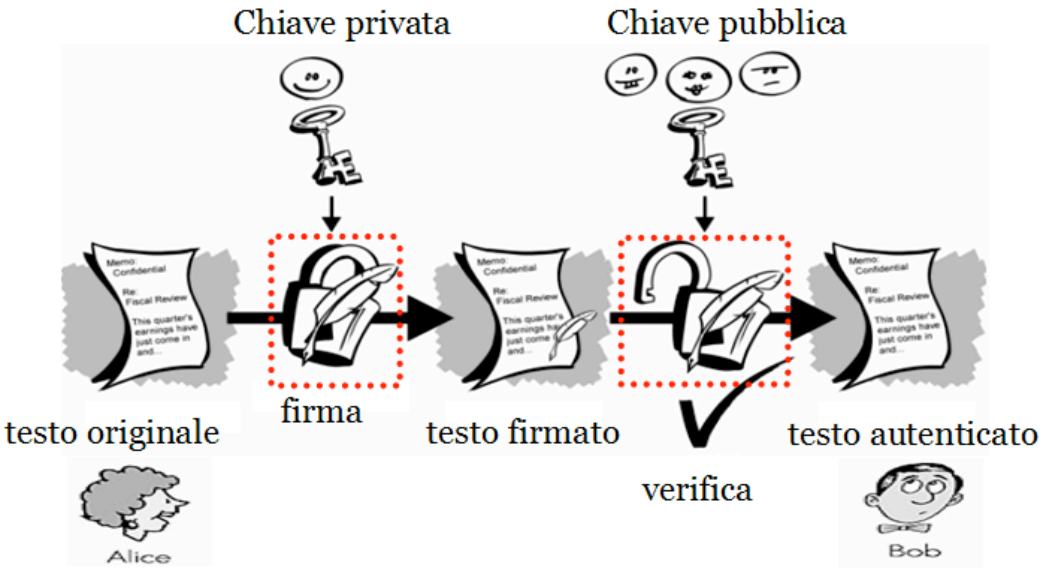
In sostanza la firma digitale non è altro una funzione che, preso un messaggio x appartenente allo spazio dei messaggi permette di ottenere una y appartenente lo spazio della firma.



Ciò che si può notare attraverso le rappresentazioni insiemistica è che se il mittente per firmare un messaggio utilizza oltre ad un algoritmo di firma la sua chiave privata, il destinatario utilizza per verificare la validità della firma una chiave pubblica.

[1] Soggetto assolve alla funzione di terza parte fidata e rappresenta una sorta di notaio telematico che garantisce sull'autenticità della firma, col rilascio di un certificato relativo alla chiave asimmetrica di cifratura con la quale la firma stessa è stata generata.

Tale funzionamento può essere più chiaro utilizzando un esempio:



Si supponga che il mittente Alice debba inviare un messaggio x al destinatario Bob.
Dal suo punto di vista, Bob vuole essere sicuro che il messaggio che riceverà sia effettivamente stato inviato Alice (autenticità) e non sia stato modificato successivamente (integrità).
Per ottenere tutto ciò, Alice invierà il messaggio x apponendovi la sua firma, attraverso l'utilizzo di un algoritmo di firma e della propria chiave privata. In questo modo Bob può decifrare la firma digitale con la chiave pubblica di Alice e verificarne l'autenticità.
Più schematicamente ciò che avviene è che:

1. Alice firma il suo messaggio x con la sua chiave privata $\mathbf{K_{priv} = sig_{Kpriv}(x)}$
2. Alice invia la coppia **(x,y)** a Bob
3. Bob applica la funzione di verifica $\mathbf{ver_{Kpub}(x,y)}$ con la chiave pubblica di Alice.

Quindi negli schemi di firma digitale vengono utilizzate le chiavi (pubblica e privata) del **mittente**.
L'aspetto sicuramente più interessante in questa breve trattazione della firma digitale è rappresentato dalle proprietà che essa offre, che possono essere riassunte in cinque punti fondamentali:

1. solo Alice può firmare il suo documento con la sua chiave privata K_{priv};
2. chiunque può verificare la firma con la chiave pubblica K_{pub};
3. **Autenticazione**: Bob è sicuro che sia stata Alice a firmare il messaggio;

4. **Integrità**: il messaggio x non può essere alterato dal momento che ciò verrebbe scoperto attraverso l'applicazione della funzione di verifica;
5. **Non ripudiabilità**: nessun utente deve poter ripudiare o negare i messaggi da lui spediti.

Questo meccanismo garantisce anche l'autenticità e, dunque, la non ripudiabilità del messaggio da parte del suo autore, una volta che si è sicuri che la chiave pubblica relativa a quella privata con cui è apposta la firma sul messaggio sia effettivamente la chiave pubblica del mittente. Tale sicurezza è garantita dell'autorità di certificazione.

La firma digitale diviene dunque una garanzia del messaggio o documento elettronico, alla stregua della sottoscrizione di un documento cartaceo, attestandone con certezza l'integrità, l'autenticità e la non ripudiabilità anche dal punto di vista legale, poiché la legislazione italiana attribuisce ad un documento elettronico con firma digitale lo stesso valore dello stesso in forma cartacea sottoscritto con firma autografa.

Ci sono diversi schemi di firma digitale fra cui:
(1) RSA
(2) El Gamal
(3) Rabin

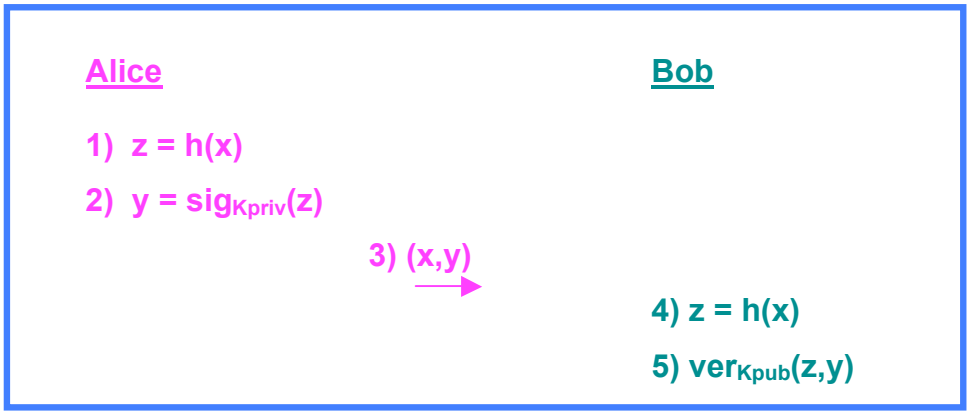
1.2 Funzioni di hash

Un altro strumento che gioca un ruolo fondamentale nella cartografia moderna è rappresentato dalle funzioni di hash. Queste funzioni sono usate per garantire l'integrità dei dati in congiunzione con schemi di firma digitale, in particolare un messaggio viene sottoposto prima ad un'operazione di hashing e successivamente il valore di hash è firmato al posto del messaggio originale.

Una funzione di hash è una funzione che, dato un qualunque messaggio di lunghezza arbitraria, ne produce un'impronta (detta digest) di lunghezza prefissata (di solito dell'ordine di 100-200 bit).

L'utilità di una funziona di hash sta nel poter utilizzare l'impronta come rappresentazione compatta del messaggio stesso, tipicamente firmando l'impronta anziché l'intero messaggio perchè questo possa avvenire è desiderabile che la funzione presenti due proprietà particolari, sia cioè *senza collisioni* e *unidirezionale*.

Prima di vedere però che cosa permettono di garantire le proprietà di cui accennato sopra è utile, per comprendere come avviene l'utilizzo delle funzioni di hash fornire un esempio.



1. Il mittente, ovvero Alice, applica la funzione di hash h al messaggio x da inviare a Bob, ottenendo così l'impronta z.
2. Successivamente Alice calcola il y ovvero il valore che si ottiene in output all'algoritmo di firma (uso della chiave privata) fornendogli in input l'impronta z e non più, come nel caso della firma digitale, il messaggio x.
3. Alice invia così la coppia messaggio - firma (x, y) al destinatario.
4. Bob ricevuta la coppia (x, y) applica, a sua volta, la funzione di hash sul messaggio ricevuto da Alice ottenendo così l'impronta z.
5. Bob confronta il valore da lui calcolato con quello inviatogli da Alice applicando la funzione di verifica (uso della chiave pubblica).
   Se tale verifica ha esito positivo allora il messaggio è accettato dal destinatario, in caso contrario viene scartato.

Si è gia accennato al fatto che una funzione di hash deve garantire alcune proprietà fondamentali, vediamole in dettaglio.

- **One-way (unidirezionale):**
  - se dato un qualsiasi x è computazionalmente semplice calcolare **h(x)**

- se data un'impronta z, è computazionalmente impraticabile trovare un messaggio x tale che **h(x)=z**

L'idea è che l'hash di un messaggio sia in sostanza una corrispondenza pseudocasuale per cui non è possibile prevedere in alcun modo il risultato se non calcolando l'hash (messaggi diversi anche per un solo bit devono generare hash non correlati).

- **Weak collision resistant:** se dato un messaggio x è è computazionalmente impraticabile trovare un messaggio qualsiasi x' tale che **h(x) = h(x ')**.

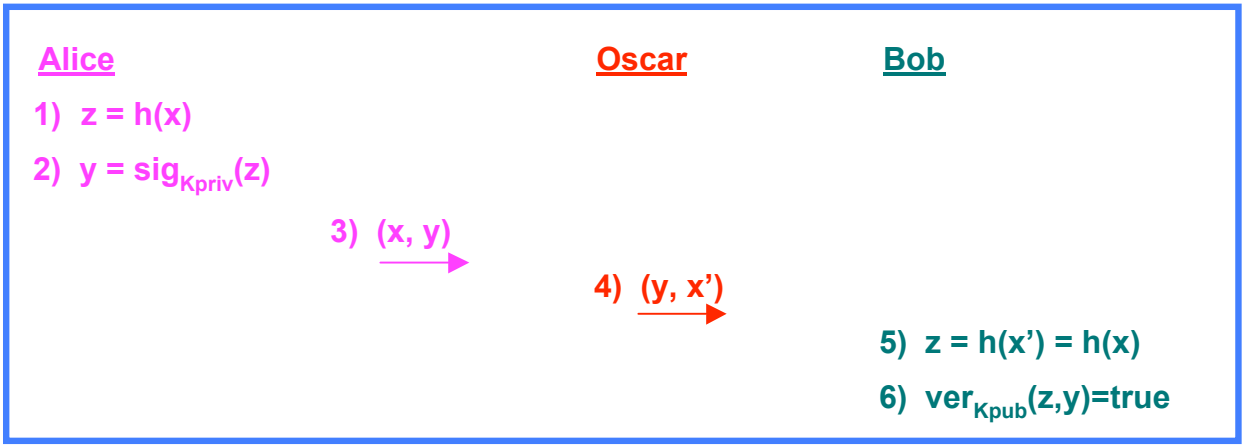
- **Strong collision resistant:** se è computazionalmente impraticabile trovare due messaggi x e x' tali che **h(x)=h(x')**.

L'idea è che, anche se necessariamente esistono messaggi diversi che producono lo stesso hash, non è praticabile trovarli.

Ovviamente, laddove tali proprietà non fossero garantite le funzioni di hash, e quindi il messaggio x, sarebbero attaccabili. In particolare è possibile distinguere per ogni proprietà un particolare tipo di attacco.

- Se h(x) non è unidirezionale, un assalitore può calcolare x da h(x) nei casi in cui x è cifrato.

- **Attacco a forza bruta "semplice"**: Se h(x) non è Weak collision free, un assalitore può sostituire il messaggio x con x' (trovare un messaggio che produca lo stesso hash del messaggio inviato), ovvero:



- **Attacco del compleanno**: Se h(x) non è Strong collision free, l'assalitore può trovare un messaggio che produca lo stesso hash indipendentemente dal valore di quest'ultimo, ovvero:

  a) Oscar sceglie il messaggio legittimo x_1 ed il messaggio fraudolento x_2;

  b) altera x_1 e x_2 a una posizione "non-visibile", cioè sostituisce i tabs con gli spazi, collega ritorni, ecc., fino a che $h(x'_1) = h(x'_2)$.

  Per avere un'idea della complessità computazionale basti pensare che 64 alterazioni di posizione permettono 2^{64} versioni di un messaggio con 2^{64} valori differenti di hash).

  c) Se Alice firma $\mathbf{x'_1 \rightarrow (x'_1, sig_{Kpriv}(h(x'_1)))}$

  d) Oscar sostituisce $\mathbf{x'_1 \rightarrow x'_2}$ **e** $\mathbf{(x'_2, sig_{Kpriv}(h(x'_2)))}$.
  Non permettendo così al destinatario di renderesi conto dell'avvenuta corruzione del messaggio.

La complessità degli ultimi due e tipi di attacco è molto diversa. Se l'hash è lungo m bit, la complessità del primo è $\mathbf{2^m}$ mentre quella del secondo è $\mathbf{2^{m/2}}$.

Gli algoritmi più utilizzati per la cifratura con funzioni hash sono i seguenti:

MD5 (MESSAGE DIGEST 5) creato da R. Rivest nel 1991, è nato per sostituire MD4.
Produce valori hash di 128 bit, accettando messaggi di input con lunghezza massima di 264 bit e scomponendoli in blocchi di 512 byte.

SHA (SECURE HASH ALGORITHM) è un algoritmo sviluppato nel 1994 dal NIST (National Istitute of Standard and Tecnology), e dalla NSA (National Security Agency), molto simile a MD4.
Fornisce valori hash di 160 bit. La cifratura viene effettuata su messaggi con lunghezza massima di 264 bit, scomponendoli in blocchi di 264 bit. Risulta un po' più lento di MD5, ma produce un message digest più grosso che lo rende più sicuro agli attacchi per forza bruta.

SHA-1 è una revisione del 1995 dell'algoritmo SHA

SHA-2 è una nuova versione dell'algoritmo SHA, pubblicata nel 2002, che fornisce valori hash di 256 bit.

RIPEMD (Race Integrity Primitives Evaluation Message Digest) è un algoritmo sviluppato nel 1996 per rimpiazzare MD4. Ne esistono varie versioni: RIPEMD-128, RIPEMD-160, RIPEMD-256 e RIPEMD-320 con lunghezza della chiave rispettivamente di 128, 160, 256 e 320 bit. Si tratta essenzialmente di un algoritmo con caratteristiche e velocità paragonabili a quelle di SHA-1.

Come si può notare la lunghezza dei valori di *hash* varia a seconda degli algoritmi.
Quelli a 128 bit sono i più comuni e i preferiti, per il maggiore numero dei bit che ne assicura una maggiore resistenza agli attacchi. Si pensi, infatti, che una funzione *hash* a 4 bit non è di alcuna utilità per una verifica dell'integrità perché 1/16 dei possibili messaggi sono mappati su uno dei 16 possibili valori di *hash*. Un hacker che modifica un messaggio può farlo agevolmente in modo tale da lasciare lo stesso valore di *hash*.
Se invece si usa un valore di *hash* da 160 bit, l'*hacker* dovrebbe modificare il messaggio in 2^{159} maniere diverse per ottenere il corretto valore di *hash*.

In conclusione va sottolineato un ultimo aspetto fondamentale riguardante le funzioni di hash ovvero che *un hash non è né una firma né un MAC, di per sé quindi non garantisce autenticazione e/o integrità. Nel calcolo di un hash non si introduce alcuna informazione segreta per cui chiunque può generare l'hash corretto di un qualunque messaggio. Per questi motivi, tale soluzione, deve essere utilizzata, come già ribadito precedentemente, in combinazione con altri meccanismi, quali quelli di firma digitale.*

1.3 Message Authentication Codes (MACs)

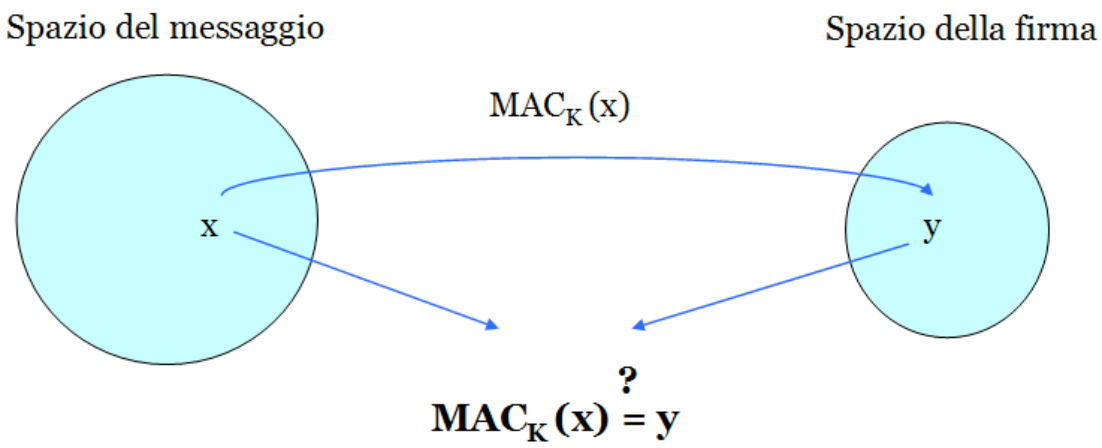
Una classe distinta di funzioni di hash è chiamata Message Authentication Codes (o keyed hash function) e permette L'autenticazione dei messaggi con tecniche simmetriche.
In particolare gli algoritmi MAC possono essere visti con funzioni di hash che prendono in ingresso due input distinti, un messaggio è una chiave segreta, per produrre un output di lunghezza fissata, con l'intento di rendere impossibile la produzione dello stesso output senza conoscere la chiave k.
I MAC può essere usato per garantire **integrità dei dati** e **autenticazione dell'origine** dei dati stessi.

Più precisamente, possiamo definire un algoritmo di MAC come una funzione h_k (x), parametrizzata da una chiave k, caratterizzata dalle seguenti proprietà:

- Dati una chiave k e un input x, è facile calcolare $h_k(x)$. Tale valore è detto brevemente MAC.
- Dato un input x di lunghezza qualsiasi la funzione genera un output $h_k(x)$di lunghezza prefissata.
- Noto un certo numero di coppie $(x_i, h_k(x_i))$ è computazionalmente impraticabile calcolare qualunque altra coppia $(x, h_k(x))$ per un qualsiasi nuovo input $x \neq x_i$ senza conoscere k.

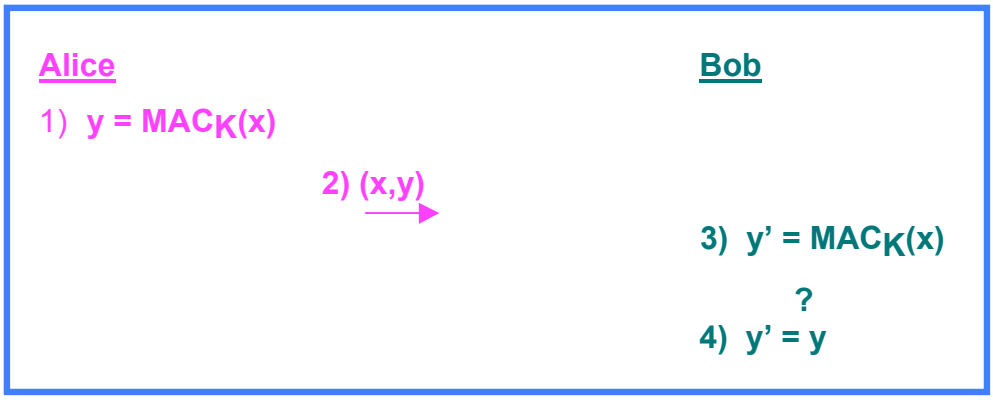
Un aspetto importanente che differenzia il MAC dalla firma digitale sta nel fatto che questi ultimi sono generati e verificati con la stessa chiave (simmetrica).

Quindi il MAC lo possiamo vedere come il risultato dell'applicazione di una funzione che dato un elemento appartenente allo spazio del messaggio lo trasforma in un elemento (y) appartenete allo spazio della firma. Come si vede dalla rappresentazione , il confronto effettuato dal destinatario verrà fatto sul valore del MAC.



Anche nel caso dell'uso di MAC vediamo con un esempio come avviene la trasmissione del messaggio e la sua verifica:

1. Alice applica la funzione MAC sul messaggio x utilizzando la chiave K.
2. Alice invia a Bob la coppia (x,y) con y è impronta (authentication tag) del messaggio x calcolata dall'algoritmo di autenticazione MAC.
3. Quando Bob riceve la coppia (x,y) esegue un algoritmo di verifica per accettare o rifiutare il messaggio.



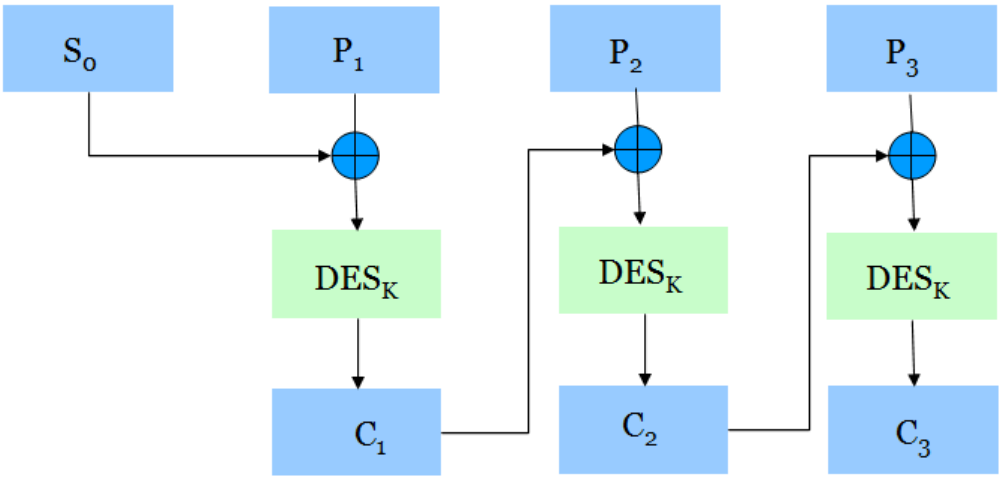
OSSERAVZIONE

E' un meccanismo basato su **chiave privata**: colui che firma il messaggio e chi ne verifica l'autenticità devono condividere la stessa chiave segreta.

Esistono diversi tipi di MAC, quelli più usati sono il MAC-CBC e l'HMAC. Vediamoli in dettaglio.

- **MAC basato su modalità di codifica CBC (Cipher Block Chaining)**

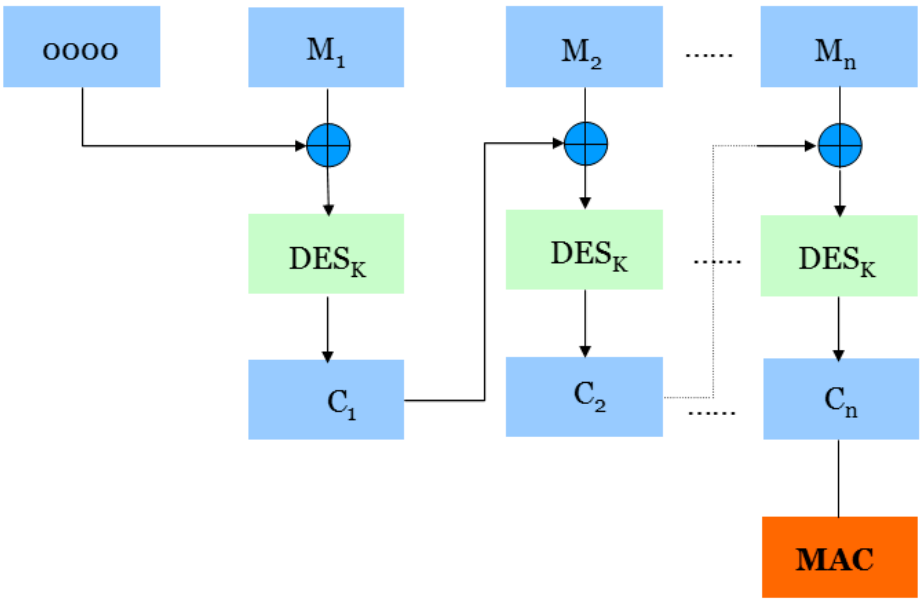
Il messaggio viene cifrato con un algoritmo simmetrico a blocchi (es. DES-CBC) e il valore dell'ultimo blocco viene aggiunto al messaggio utilizzandolo come MAC.
Per comprenderne appieno il funzionamento è utile rivedere brevemente il funzionamento di un algoritmo di codifica a blocchi.



Si determina un vettore di inizializzazione S_0 (seed – seme), consistente in un numero random tra 0 e $2^{b/8}$(con b dimensione dei blocchi in cui è suddiviso il messaggio).

Si utilizza tale vettore, combinandolo in exclusive or con il testo in chiaro nel primo blocco di codifica. Il risultato viene poi cifrato con un cifrario a piacere (per esempio il DES), ottenendo il testo cifrato.
Nel blocco successivo, l'operazione viene ripetuta, ma invece del vettore di inizializzazione, si impiega il primo testo cifrato. Nel terzo blocco si impieghera' quello del secondo e cosi' via, concatenando (da qui il termine "chaining") il risultato finale di ogni blocco con la cifratura del successivo.
L'operazione di decifratura avviene invertendo il percorso: si sottopone cioè il testo cifrato alla funzione di decifratura e ne si combina il risultato con il testo cifrato in ingresso al blocco precedente. Per il primo blocco di decifratura si utilizza S_0.

Rivisto quindi il funzionamento di un algoritmo di codifica CBC è semplice comprendere come da questo si riesce a generare un MAC. Il MAC sarà l'ultima porzione di messaggio cifrato (in figura indicato con Cn).



Vediamo quindi, schematicamente, quali sono in tal caso le operazioni da effetture prima di effettuare la trasmissione del messaggio:

- si fissa il seme di tutti 0;
- dato un messaggio di n blocchi M_1, M_2, ..., M_n si applica CBC usando la chiave segreta K;
- si scartano i primi n-1 blocchi cifrati C_1, C_2, . . . , e si usa C_n;
- si invia M_1, M_2, ..., M_n e il tag di autenticazione $MAC_k(M) = C_n$.

Una volta ricevuto il messaggio, il destinatario, che conosce la chiave di cifratura, applichera' il procedimento ad ogni parte del testo in chiaro, ottenendo l'ultima porzione di codice cifrato Cn: se questa sara' identica a quella ricevuta insieme al messaggio, l'integrita' sara' garantita.

- **MAC hash function-based (HMAC)**

E' un MAC che si basa sull'utilizzo di una chiave e di una funzione hash vista come scatola nera.

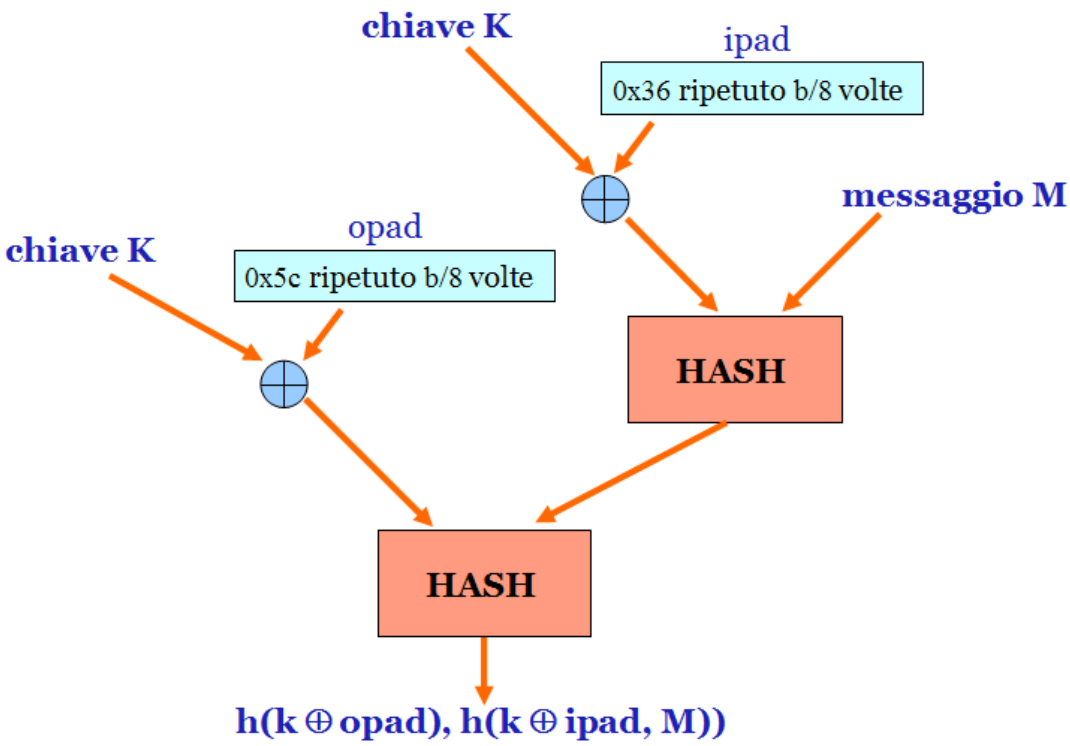
L'HMAC riceve in input un messaggio m, una chiave k ed una funzione hash h e calcola il MAC come:

$$HMACK(m,h) = h(k \oplus opad), h(k \oplus ipad, m))$$

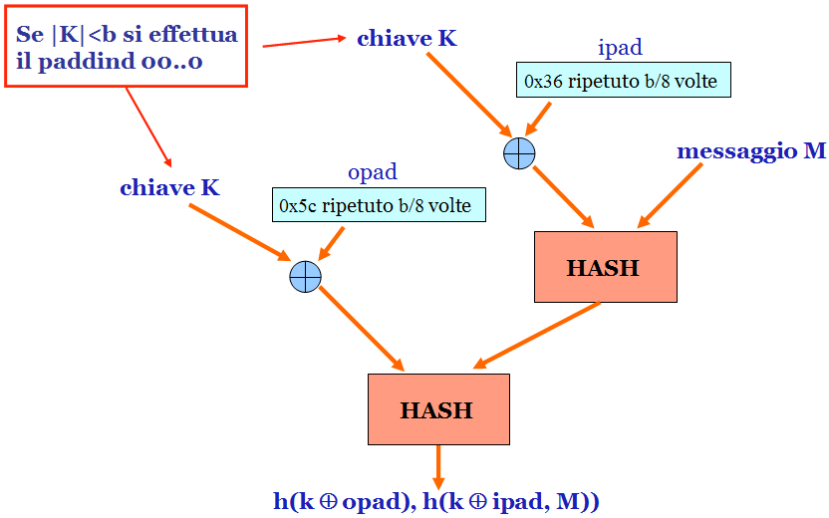
dove opad e ipad rappresentano, rispettivamente, 2 stringe fissate di 8 bit:

- opad = 01011010 ripetuto b/8 volte (con b dimensione del blocco)
- ipad = 00110110 b/8 volte (con b dimensione del blocco)

Il funzionamento è meglio spiegato da uno schema:



Nel caso i cui la chiave ha una lunghezza minore di b (per esempio 512) allora si aggiungono ad essa tanti zeri a sinistra quanti ne servono per arrivare alla lunghezza richiesta (si effettua un'operazione di padding).



Nel caso in cui, invece, la chiave ha una lunghezza superiore alla dimensione del blocco allora si effettua prima un hash della chiave.

Riassumendo, il messaggio viene dato in input ad una funzione hash che genera il message digest relativo, quindi un algoritmo di cifratura, cifra il message digest per mezzo di una chiave ed infine, il valore ottenuto viene aggiunto al messaggio stesso.
Chi riceve il messaggio può verificarne l'integrità decifrando il HMAC con l'apposita chiave e confrontando il message digest ricevuto con quello calcolato sul momento relativamente al messaggio in chiaro ricevuto.
Spesso nella pratica l'hash che si ottiene in output viene troncato ovvero se ne usano solo i primi t bit. Questo è un vantaggio perché fornisce meno informazioni a chi vuole sferrare un attacco ma nel contempo è uno svantaggio perché l'attaccante ha meno bit da predire.
Un esempio di algoritmo che effettua tale troncamento è l'HMAC-SHA1-80 che usa solo i primi 80 bit dei 160 forniti in output
e che pertanto si contrappone all' HMAC-MD5 che usa tutti i 128 bit dell'hash.
In ogni caso comunque il numero di bit t scelti deve essere $t \geq b/2$ per una funzione hash di b bit, in modo tale da garantire l'integrità d3el messaggio.
Ultimo aspetto rilevante è che gli HMAC sono più veloci dei cifrari a blocchi.

2. Entity Authentication

Conclusa, dunque la trattazione delle soluzioni che permettono di ottenere l'autenticazione del messaggio, passiamo definire il problema dell'entity authentication e le soluzioni atte a risolverlo.
Con il termine autenticazione dell'entità si intende l'identificazione corroborata da prove di un'entità coinvolta in un protocollo. Ovviamente esistono molte soluzioni, fra loro assai differenti, che permettono di garantire tale tipologia di identificazione. Una classificazione di tali tecniche, che di solito è usuale adoperare, è quella che le suddivide in tre categorie, in base agli strumenti usati per garantire la sicurezza. Le tre categorie sono:

- **Qualcosa che si conosce**: Password, PIN (Personal Identification Numbers) e chiavi pubbliche e private.
- **Qualcosa che si possiede**: Smart Card (della carte contenenti un microprocessore o un circuito integrato) e generatori di password.
- **Qualcosa che si è**: impronte digitali, riconoscimento dell'iride, della voce, della calligrafia (biometria).

Idealmente, per garantire una corretta autenticazione, dovrebbero essere combinati insieme due o più di questi meccanismi.

Prima di passare alla trattazione delle soluzioni adoperate in tale campo è interessante considerare due aspetti che diversificano fortemente l'entity authentication dalla message authentication. In primo luogo la caratteristica di real time: nell'Entity Authentication si rafforza l'identità del chiamante in tempo reale attraverso l'applicazione del protocollo mentre nella Message Authenticatin la verifica dell'autenticazione del messaggio non deve necessariamente avvenire in tempo reale (es. per firma digitale la verifica può avvenire parecchio tempo dopo rispetto a quando la firma è stata apposta). In secondo luogo la caratterizzazione dl messaggio: nell'Entity Authentication non si coinvolge alcun messaggio significativo se non l'accertamento di un particolare soggetto, cosa che ovviamente non si verifica nella Message Authenticatin.

Infine vediamo quali sono le proprietà fondamentali che i protocolli trattati qui di seguito dovranno garantire.

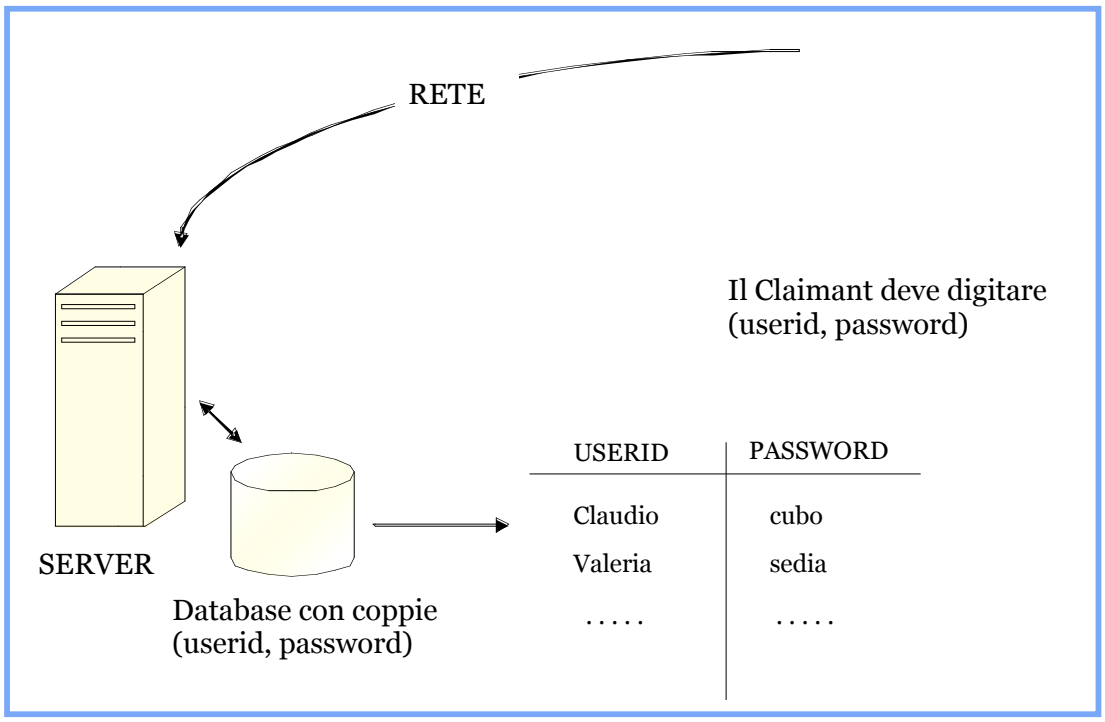
- Reciprocità dell'autenticazione: una delle due parti o entrambe devono fornire rispettivamente un'autenticazione unilaterale o mutua;
- Efficienza computazionale: il numero di operazioni che il protocollo richiede che eseguire;
- Efficienza della comunicazione: il numero di passaggi (cambiamenti del messaggio) e l'ampiezza di banda richiesta;
- Necessità di una terza parte
- Necessità di coinvolgere una terza parte in tempo reale: es. i servizi per la distribuzione di chiavi pubbliche certificate, supportati da un'autorità di certificazione;
- Natura della fiducia richiesta dalla terza parte
- Quali garanzie di sicurezza sono offerte
- Come e dove sono mantenute e conservate le chiavi e/o i segreti (dischi locali, software, ecc.).

A questo punto possiamo quindi passare a parlare delle diverse tipologie di entity authentication ovvero l'autenticazione debole e quella forte, nonché dei protocolli a conoscenza zero (zero-knoledge) e, per concludere di un noto protocollo di autenticazione fa client e server: Kerberos.

2.1 Autenticazione debole

I tradizionali schemi di password sono caratterizzati da password stabili nel tempo, che determinano la così detta autenticazione debole. L'idea di base è la seguente. Una password, associata con ogni utente (entità), è tipicamente una stringa lunga da 6 a 10 caratteri. Essa funge da segreto condiviso tra il sistema e l'utente. (I tradizionali schemi di password rientrano così nella categoria di tecniche a chiave simmetrica che forniscono l'autenticazione unilaterale). Per ottenere l'accesso ad una risorsa del sistema (per esempio, computer account, stampante, o applicazioni software), l'utente inserisce una coppia del tipo <userid, password>; l'userid (user identificator) è una dichiarazione di identità, e la password è una prova a supporto dell'identificazione. Il sistema controlla che la password abbia una corrispondenza tra i dati memorizzati, confrontando la stringa digitata con la password dell'userid specificato, a quel punto l'identità stabilita è autorizzata

ad accedere alla risorsa. L'insieme delle idee presentate nelle seguenti sezioni motiva le decisioni progettuali realizzate in schemi di password tipici. Una sezione successiva riassume gli attacchi standard che questi progetti neutralizzano. Le minacce da cui dobbiamo proteggerci includono: rivelazione di password (al di fuori del sistema) ed intercettazione (nel sistema), entrambe consentono attacchi replay.



2.1.1 Schemi a password fissa

2.1.1.1 File di password memorizzati

L'approccio più semplice per il sistema è quello di memorizzare in un file di sistema non cifrato, protetto sia in lettura che in scrittura (per esempio attraverso il sistema operativo si accede ad istruzioni privilegiate di controllo), la coppia <userid, password> relativa ad ogni utente. Quando l'utente vuole identificarsi al sistema immette la sua coppia <userid, password>. Il sistema ricerca nel file l'userid inserito, confronta la password con quella memorizzata relativa a tale userid. Se le password sono identiche, l'utente accede al sistema altrimenti viene respinto. Un inconveniente di tale metodo è che non fornisce alcuna protezione contro gli utenti privilegiati o superuser (userid speciali che godono di accessi privilegiati ai file di sistema e alle risorse).

2.1.1.2 File di password “cifrate”

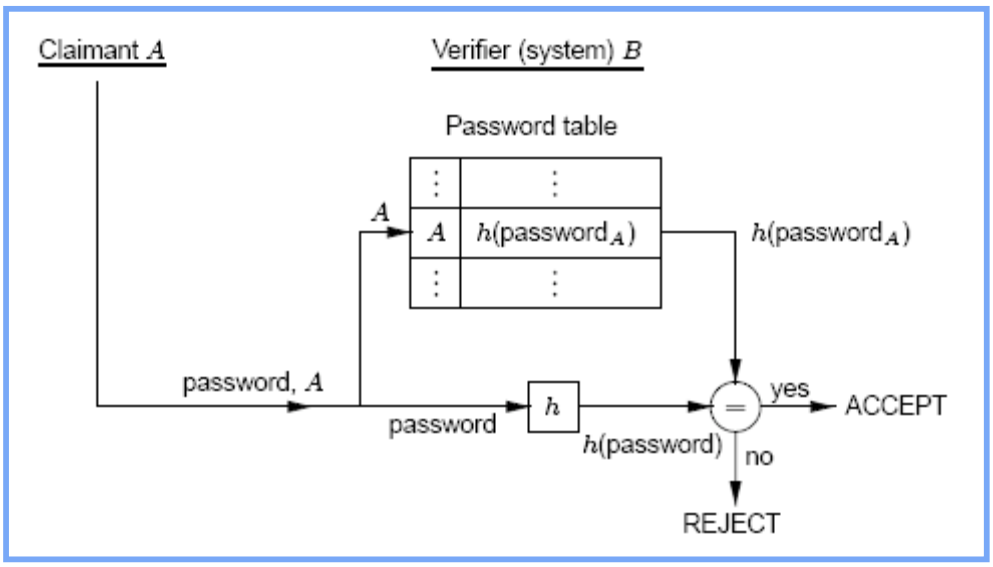


Figura 1: Uso di una funzione one-way per password-checking

Piuttosto che memorizzare la password dell'utente in chiaro in un file, si memorizza una funzione one-way della password (Figura 1). Con questo metodo il sistema memorizza nel file di password una coppia <userid, h(password)>, dove h(password) è una funzione hash della password. L'utente che vuole accedere al sistema inserisce la sua coppia <userid, password>. Il sistema ricerca l'userid nel file, calcola una funzione hash della password inserita e la confronta con quella memorizzata; se sono uguali l'identificazione ha successo, altrimenti viene rifiutata.

2.1.1.3 Regole della password

Le caratteristiche fondamentali che le password dovrebbero rispettare sono:

- difficili da indovinare;
- facili da ricordare.

In realtà è arduo combinare queste due proprietà, in quanto qualcosa che è facile da ricordare risulta anche prevedibile! Dato che gli attacchi di tipo dizionario hanno successo con le password prevedibili, alcuni semplici accorgimenti adottati nella loro scelta consentono di evitare gli attacchi o comunque renderli più complessi. Alcuni sistemi impongono delle "regole della password" per scoraggiare o impedire l'uso di password "deboli" da parte degli utenti. Regole della password tipiche richiedono un limite inferiore alla lunghezza della password (per esempio, 8 o 12 caratteri); inoltre si richiede che ogni

password contenga almeno un carattere di ciascun insieme delle categorie (per esempio, maiuscole, numerici, non-alfanumerici); si controlla che le password candidate non siano presenti in dizionari disponibili on-line e che non siano composte da informazioni relative all'account (userid o sue sottostringhe). In effetti, conoscendo tali regole, un avversario potrebbe usare una strategia di attacco di tipo dizionario modificata tenendo conto delle regole, proponendosi di trovare la più debole forma di password che le soddisfi.

Alcuni aspetti che rendono le password vulnerabili derivano dai criteri di scelta.

Infatti le password sono per lo più costituite da:

- parole comuni (in qualsiasi lingua);
- nomi comuni (nomi TV, nomi di musicisti, vezzeggiativi, nomi di amici, membri della famiglia, soprannomi);
- informazioni facili da ottenere (date di nascita, numeri di telefono, codice fiscale, targa della macchina);
- tasti consecutivi di tastiere ("qwerty");
- password su altri sistemi;
- permutazioni di queste lettere (lettere al contrario).

Alcune soluzioni per risolvere il problema della vulnerabilità potrebbero essere:

- mescolare lettere minuscole e maiuscole;
- usare qualcosa di impronunciabile;
- includere caratteri non alfanumerici;
- mescolare lettere e numeri;
- eseguire sostituzioni sistematiche, come o—>0 oppure i—>1;
- scegliere lettere da un periodo o da una frase;
- generazione di una sequenza casuale da parte di un computer. Questo sistema genera combinazioni difficili da ricordare, ma anche tali programmi possono avere dei bug. Per esempio, si sono avuti casi in cui la stessa combinazione apriva più lucchetti.

Un'altra tecnica procedurale intesa a migliorare la sicurezza della password è l'invecchiamento della password. Un periodo di tempo è definito come limite di ciascuna password (per esempio 30 o 90 giorni). Ciò richiede che le password siano cambiate periodicamente.

2.1.1.4 Rallentare il mapping della password

La funzione di verifica della password (per esempio, la funzione one-way) può essere resa intensa dal punto di vista computazionale; per esempio, iterando una semplice funzione per t volte, con t>l, con l'output dell'iterazione i usato come input all'iterazione i + 1. Il numero totale di iterazioni deve essere limitato in modo da non imporre ai legittimi utenti un ritardo notevole o inammissibile.

2.1.1.5 Salting password

Per rendere gli attacchi di tipo dizionario meno efficaci, ogni password, nella lista iniziale, può essere aumentata con una stringa random di s bit chiamata salt prima di applicare la funzione one-way. Sia le password modificate che i bit di salt sono registrati nel file di password. Quando l'utente introdurrà una password, il sistema cercherà il salt, aggiungerà alla password tale valore e applicherà una funzione one-way alla stringa ottenuta. La difficoltà di una ricerca esaustiva su qualsiasi password di un particolare utente non è modificata dal salting (siccome il salt è dato in chiaro nel file delle password); comunque, il salting incrementa la complessità di un attacco di tipo dizionario effettuato simultaneamente su un insieme di password. Il dizionario richiede di contenere 2S variazioni di ciascun tentativo, implicando una grande richiesta di memoria per registrare un dizionario cifrato, e in concomitanza molto tempo per la sua preparazione. Da notare che con il salting due utenti che scelgono la stessa password hanno differenti entry nel file di password. In alcuni sistemi può essere appropriato usare lo stesso userid dell'entità come salt.

2.1.1.6 Passphrase

Per consentire una maggiore entropia tenendo conto della capacità di memoria degli utenti, le password possono essere estese in passphrase; in questo caso, gli utenti digitano una frase o un detto, piuttosto che una semplice " parola ". La passphrases viene troncata se supera la lunghezza fissata ed ha lo stesso ruolo della password; è importante che la passphrase non sia facilmente troncata dal sistema, come viene fatto per le password in alcuni sistemi.

L'idea è che l'utente possa ricordare le frasi più facilmente rispetto ad una sequenza di caratteri casuali. Se le password derivano dal testo inglese (in cui ogni carattere contiene solo circa 1.5 bit di entropia) una passphrase fornisce una sicurezza maggiore attraverso

l'incremento dell'entropia rispetto ad una breve password. Un inconveniente è il dover digitare una quantità maggiore di testo.

2.1.2 Attacchi agli schemi a password fissa

2.1.2.1 Reply alle password fisse

Una debolezza dello schema che usa password fisse riproponibili, è la possibilità che un avversario capisca la password di un utente osservando il modo in cui viene digitata. Un secondo problema di sicurezza è che le password inserite dall'utente (o funzioni one-way) siano trasmesse in chiaro sulla linea di comunicazione tra l'utente ed il sistema, e siano disponibili sempre in chiaro temporaneamente durante la verifica del sistema. Un avversario curioso potrebbe registrare questi dati ed effettuare successive impersonificazioni. Schemi di password fissa sono usati perciò quando la password è trasmessa su linee di comunicazione affidabili, sicure dal monitoraggio, ma non sono adatti nel caso in cui le password siano trasmesse su reti di comunicazioni aperte. Per esempio nella Figura 1, il richiedente A potrebbe essere un utente che si connette da casa con un modem sulla linea telefonica, ad un ufficio B, lontano due (o migliaia) di chilometri; la password in chiaro potrebbe viaggiare su una linea non sicura (includendo la possibilità di un collegamento via etere). Se si effettua una verifica remota dell'identità per accedere ad una risorsa locale, per esempio uno sportello bancario automatico con verifica dell' identità on-line, la risposta del sistema (accettazione/rifiuto) dovrebbe essere protetta insieme alla password.

2.1.2.2 Ricerca esaustiva di password

Un attacco molto semplice richiede che un avversario provi le password una alla volta, con un sitema verificatore nella speranza di trovare la password corretta. Questo tipo di attacco può essere reso più difficile ampliando lo spazio in cui esse vengono scelte, limitando il numero di tentativi non validi (on-line) consentiti in determinati periodi di tempo e rallentando il mapping della password. Gli attacchi off-line che non richiedono alcuna diretta interazione con il sistema verificatore sono di maggiore interesse. Dato un file di password contenente funzioni one-way delle password utente, un avversario può tentare di sconfiggere il sistema testando le password una alla volta e confrontando la funzione one-way di ciascuna con le password presenti nel file di password. Questo è teoricamente possibile siccome sia il mapping one-way che il testo (ipotetico) sono noti (ciò potrebbe

essere evitato gestendo i dettagli di un mapping one-way o tenendo segreto il file delle password, ma non è considerato prudente basare la sicurezza del sistema sull'assunzione che tali dettagli rimangano segreti per sempre). La flessibilità dell'attacco dipende dal numero di password che devono essere controllate prima di trovare quella giusta e dal tempo richiesto per testarne una. Infine conta il mapping di password usato, la sua implementazione, il tempo di esecuzione dell'istruzione del processo host e il numero di processori disponibili.

2.1.2.3 Attacchi dizionario

Per migliorare la probabilità di successo di una ricerca esaustiva, piuttosto che cercare nello spazio di tutte le possibili password, un avversario può cercare in uno spazio ridotto. Idealmente le stringhe arbitrarie di N caratteri potrebbero essere tanto probabili quanto le password selezionate dall'utente, di solito però la maggior parte degli utenti seleziona le password da un piccolo sottoinsieme dell'intero spazio delle password (per esempio, password brevi, parole del dizionario, nomi propri, stringhe in minuscolo). Tali password deboli, sono facilmente prevedibili.

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	23.5	25.9	29.8	32.9
6	28.2	31.0	35.7	39.4
7	32.9	36.2	41.7	46.0
8	37.6	41.4	47.6	52.6
9	42.3	46.5	53.6	59.1
10	47.0	51.7	59.5	65.7

Table 10.1: *Bitsize of password space for various character combinations. The number of n-character passwords, given c choices per character, is c^n. The table gives the base-2 logarithm of this number of possible passwords.*

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	0.67 hr	3.4 hr	51 hr	430 hr
6	17 hr	120 hr	130 dy	4.7 yr
7	19 dy	180 dy	22 yr	440 yr
8	1.3 yr	18 yr	1400 yr	42000 yr
9	34 yr	640 yr	86000 yr	4.0×10^6 yr
10	890 yr	23000 yr	5.3×10^6 yr	3.8×10^8 yr

Table 10.2: *Time required to search entire password space. The table gives the time T (in hours, days, or years) required to search or pre-compute over the entire specified spaces using a single processor (cf. Table 10.1). $T = c^n \cdot t \cdot y$, where t is the number of times the password mapping is iterated, and y the time per iteration, for $t = 25$, $y = 1/(125\ 000)$ sec. (This approximates the* UNIX *crypt command on a high-end PC performing DES at 1.0 Mbytes/s – see §10.2.3.)*

Le password trovate in qualsiasi lista di parole on-line potrebbero essere scoperte da un avversario che tenta tutte le parole in questa lista usando un cosiddetto attacco dizionario. Oltre i tradizionali dizionari, ne esistono altri di parole in lingue straniere, o su argomenti specializzati come musica, film etc. Gli attacchi di tipo dizionario non hanno successo in genere nel trovare la password di un particolare utente, ma trovano molte password nella maggior parte dei sistemi.

2.1.3 One-time password

Un progresso naturale dagli schemi a password fisse ai protocolli di identificazione challenge-response può essere osservato considerando gli schemi one-time password. La maggiore sicurezza rispetto agli schemi di password fisse è nello scovare e successivamente ripetere le password. Una parziale soluzione è lo schema password one-time: ogni password è usata solo una volta. Tali schemi sono protetti da avversari passivi che scoprono le password e poi tentano di usarle.
Le variazioni a tale schema includono:

1. *Liste condivise di password one-time*. L'utente ed il sistema usano una sequenza o un insieme di t password segrete (ognuna valida per una singola autenticazione), distribuite come una lista pre-condivisa. Uno svantaggio è il dover mantenere una lista-condivisa.

Se la lista non fosse usata sequenzialmente, il sistema potrebbe controllare le password inserite rispetto a quelle inutilizzate.

2. *One-time password aggiornate sequenzialmente*. Inizialmente solo un'unica password segreta è password segreta è condivisa. Durante l'autenticazione usando la password i, l'utente crea e trasmette al sistema una nuova password (password i+1) cifrata con una chiave derivata dalla password i. Questo metodo diventa difficile se si verifica un crollo della comunicazione.

3. *Sequenze di one-time password basate su una funzione one-way*. Lo schema della one-time password di Lamport è descritto successivamente. Questo metodo è più efficiente rispetto all'aggiornamento sequenziale delle one-time password, e può essere visto come un protocollo challenge-response dove la sfida (challenge) è implicitamente definita dalla posizione corrente nella sequenza delle password.

2.1.3.1 Schema di Lamport

Nello schema one-time password di Lamport l'utente inizia con un segreto w. Una funzione one-way H è usata per definire la sequenza di password: H(w), H(H(w)), ..., $H^t(w)$.
La password per l'i-esima sessione di identificazione, 1 □ i □ t, e definita come:

Sia A un utente che si vuole identificare ad un sistema B. L'identificazione avviene usando una one-time password estrapolata da una sequenza.
Descriviamo di seguito il protocollo di Lamport basato su one-time password.

1. *One-time setup*
    - L'utente inizia con un segreto *w*. Sia H una funzione one-way.
    - Un vincolo t è fissato (per esempio t = 100 oppure 1000), definendo il numero di identificazioni che devono essere eseguite(Il sistema è poi riavviato con un nuovo *w*, per evitare attacchi di replay).
    - A trasferisce $w_0 = H^t(w)$ (il segreto condiviso), in modo da garantire la sua autenticità al sistema B. B inizializza il suo contatore per A ad $i_A = 1$.

2. *Messaggi inviati*

   Durante l'i-esima identificazione 1 □ i □ t, A invia a B un messaggio composto da:

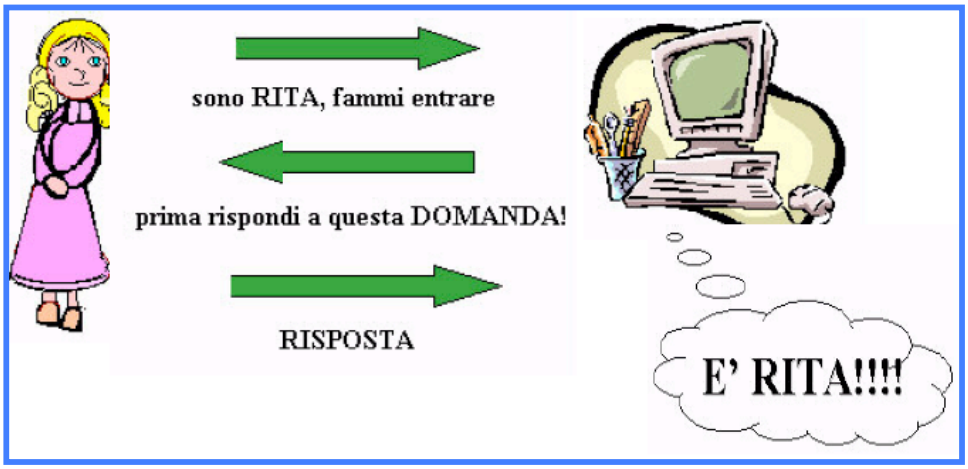
   - identificatore utente (A),
   - numero dell'identificazione (i) e
   - la password per l' i-esima identificazione (w_i)

3. *Computazioni effettuate.*

   Per identificarsi per la sessione i:

   - A calcola w_i facilmente derivato sia da w stesso o da un appropriato valore intermedio salvato durante la computazione di $H^t(w)$;
   - B controlla che i = i_A, e che la password ricevuta w_i soddisfi: $H(w_i) = w_{i-1}$.
     - Se entrambi i controlli hanno successo:
       - B accetta la password,
       - aggiorna $i_A = i_A + 1$,
       - salva w_i per la successiva sessione di verifica.

2.2 Autenticazione forte



L' idea del protocollo challenge-response, è che un' entità (il claimant) debba provare la propria identità ad una seconda entità (il Verifier), dimostrando la conoscenza di un segreto noto e associato ad entrambe le entità, senza rivelarlo sul canale di trasmissione durante l' esecuzione del protocollo.

L' attuazione di quanto detto avviene mediante un meccanismo che prevede l' utilizzo di un parametro time-variant per la rappresentazione del challenge che deve essere inviato

dal del Verifier, al quale deve corrispondere una response che dipende sia dal segreto (la chiave) che dal challenge stesso.
Il challenge è tipicamente un numero scelto dal Verifier all' inizio del protocollo stesso. Anche se un intruso dovesse catturare una coppia (challenge, response), sarebbe per questo impossibile violare il sistema perché ad ogni autenticazione dell' entità, i challenge variano e con essi anche i rispettivi response.
Prima di affrontare la presentazione dei diversi tipi di paradigmi challenge-response, affrontiamo una breve analisi dei parametri time-variant menzionati.

2.2.1 Parametri Time-variant

Esistono tre diversi tipi di parametri time-variant:

- Random Numbers (nonce);
- Sequence Numbers (serial numbers o counter values);
- Timestamps (stampe del clock allegate al messaggio originale);

Analizziamo in breve ciascuno di questi:

- **RANDOM NUMBERS:**

  Sono dei semplicissimi numeri estratti casualmente mediante un generatore casuale di numeri che però in un certo senso rappresenta un po' l' handicap nell' utilizzo di questo tipo di parametri (è infatti necessario associare ad ogni sistema di autenticazione uno di questi generatori, appesantendo la struttura).

  COME UTILIZZARLI NEI PROTOCOLLI DI AUTENTICAZIONE ?

  Una entità include un (nuovo) random number nel messaggio che deve spedire. Il messaggio contenente tale valore, arriva alla seconda entità coinvolta.

  Questa, invia un secondo messaggio (appartenente alla stessa istanza di protocollo) la cui costruzione richiede la conoscenza del numero random che lega i due messaggi.

- **SEQUENCE NUMBERS:**

  Sono delle sequenze di valori che servono ad identificare come unico un messaggio. Ciascuna di tali sequenze è specifica per ogni coppia di entità e deve essere, implicitamente o esplicitamente, associata ad entrambe.

  COME UTILIZZARLE NEI PROTOCOLLI DI AUTENTICAZIONE ?

  Quando un messaggio viene inviato, esso è indissolubilmente legato a tale sequenza.

Ciascuna sequenza viene accettata solamente nel caso in cui non sia mai stata utilizzata prima e soddisfi delle politiche di costruzione decise preventivamente (ad esempio se inizia per 0 e viene aggiornata sequenzialmente ad ogni successivo messaggio con un valore superiore di 1 a quello ricevuto o se semplicemente è una sequenza monotona e crescente).

Lo svantaggio che si ha nell' utilizzo di questo tipo di parametro nasce dalla esigenza di dover mantenere più liste di questi da parte del claimant, una per ciascun verifier con il quale comunica e deve autenticarsi.

- **TIMESTAMP:**

  I timestamps sono delle semplici variabili che contengono al momento della loro richiesta da parte dell' utente, la stampa del clock di sistema relativo allo host sul quale l'utente lavora.

  COME UTILIZZARLI NEI PROTOCOLLI DI AUTENTICAZIONE ?

  Quando un utente deve autenticarsi con un Verifier, richiede una stampa del clock del proprio sistema e la allega in modo crittografico al messaggio che deve inviare. Nel momento in cui il verifier riceve il messaggio, richiede esso stesso una stampa del clock del proprio sistema e lo confronta con quello ricevuto. Solamente se la differenza tra i due è inferiore ai 10-20 ms e non è stato ricevuto un messaggio con lo stesso timestamp in precedenza, il messaggio viene accettato. In caso contrario esso viene respinto.

  Lo svantaggio è che i due clock relativi alle due entità che devono comunicare devono essere sempre rigorosamente sincronizzati per evitare che messaggi corretti siano respinti.

Tutti questi tipi di parametri possono servire per:

- prevenire attacchi: usando combinazioni di random numbers concatenati con timestamps o sequence numbers;
- garantire tempestività e unicità:
  - indirettamente con serial number o time clock;
  - direttamente i random numbers;
- distinguere istanze: usando random numbers di grandezza sufficiente.

2.2.2 Protocollo Challenge-Response

Esistono tre tipi di protocollo challenge-response:

- Basato su tecniche a chiave simmetrica, di cui vedremo le seguenti tre tecniche:
  - Simmetric-key encryption;
  - One-way function;
  - Hand-held passcode generator;
- Basato su tecniche a chiave pubblica, di cui vedremo:
  - Decrittaggio a chiave pubblica con witness;
  - Needham-Schroedr PK modificato;
- Basato sullo schema della firma digitale.

Analizziamo qui di seguito le diverse tecniche.

2.2.2.1 Challenge-Response con tecniche a chiave simmetrica

Questa tecnica è alla base dei protocolli di Kerberos e Needham-Shroeder e presuppone che Claimant e Verifier condividano una chiave simmetrica.

Per piccoli sistemi chiusi, con pochi utenti, ciascuna coppia di users può condividere una chiave a priori, mentre per sistemi più grandi, i protocolli di identificazione spesso utilizzano un server di fiducia on-line con il quale ogni parte condivide una chiave. Tale server fornisce una chiave per ogni sessione alle due parti non appena l'una vuole autenticarsi all' altra.

Nella presentazione delle tre semplici tecniche descritte di seguito, si assume che la chiave venga scelta anticipatamente e condivisa dalle due entità e che il Claimant sia in grado di autenticarsi con il Verifier crittando un challenge attraverso tale chiave.

NOTAZIONE UTILIZZATA

- r_A : random number generato da A;
- t_A : timestamp generato da A;
- E_k : algoritmo di crittaggio simmetrico con la chiave K condivisa da A e B;
- "," : indica la concatenazione di due stringhe;
- "*" : indica l' opzionalità del parametro.

2.2.2.1.1 Autenticazione unilaterale basata su timestamp

$$A \rightarrow B : E_K(t_A, B^*)$$

Una volta ricevuto e decrtittato il messaggio, B verifica che il timestamp sia accettabile e opzionalmente controlla che l' identificativo ricevuto sia effettivamente il suo.

2.2.2.1.2 Autenticazione unilaterale basata su random numbers

$$\begin{aligned} A &\leftarrow B : r_B \\ A &\rightarrow B : E_K(r_B, B^*) \end{aligned}$$

Per evitare di fare affidamento sul timestamp per gli svantaggi visti, questo può essere sostituito da un numero random con l' insersione di una nuova istruzione.

A invia a B un messaggio costruito mediante l' algoritmo di crittaggio E_k sulla base del numero random inviatogli da B e sulla base della chiave segreta condivisa.
B decritta il messaggio ricevuto e controlla che il numero random ricevuto sia effettivamente quello da lui inviato nell' istruzione precedente.
Solo se i due numeri random sono coincidenti, il messaggio viene accettato.

2.2.2.1.3 Mutua autenticazione basata su random numbers

$$\begin{aligned} A &\leftarrow B : r_B \\ A &\rightarrow B : E_K(r_A, r_B, B^*) \\ A &\leftarrow B : E_K(r_B, r_A) \end{aligned}$$

E' un meccanismo sostanzialmente simile al precedente, con l' aggiunta di una terza istruzione che prevede l' invio di un random numbers anche da parte di A (da qui mutua autenticazione).

2.2.2.2 C-R basato su one-way functions

$$\begin{aligned} A &\leftarrow B : \quad r_B \\ A &\rightarrow B : \quad r_A,\ h_K(r_A, r_B, B) \\ A &\leftarrow B : \quad h_K(r_B, r_A, A) \end{aligned}$$

Del tutto simile al precedente con l' eccezione del tipo di crittaggio. In questo caso la funzione E_k viene sostituita da una funzione MAC h_k. Inoltre, va sottolineato che il Verifier

in questo caso calcola per proprio conto la funzione MAC e la controlla con quella ricevuta.

2.2.2.3 C-R con l'utilizzo di un hand-held passcode generator

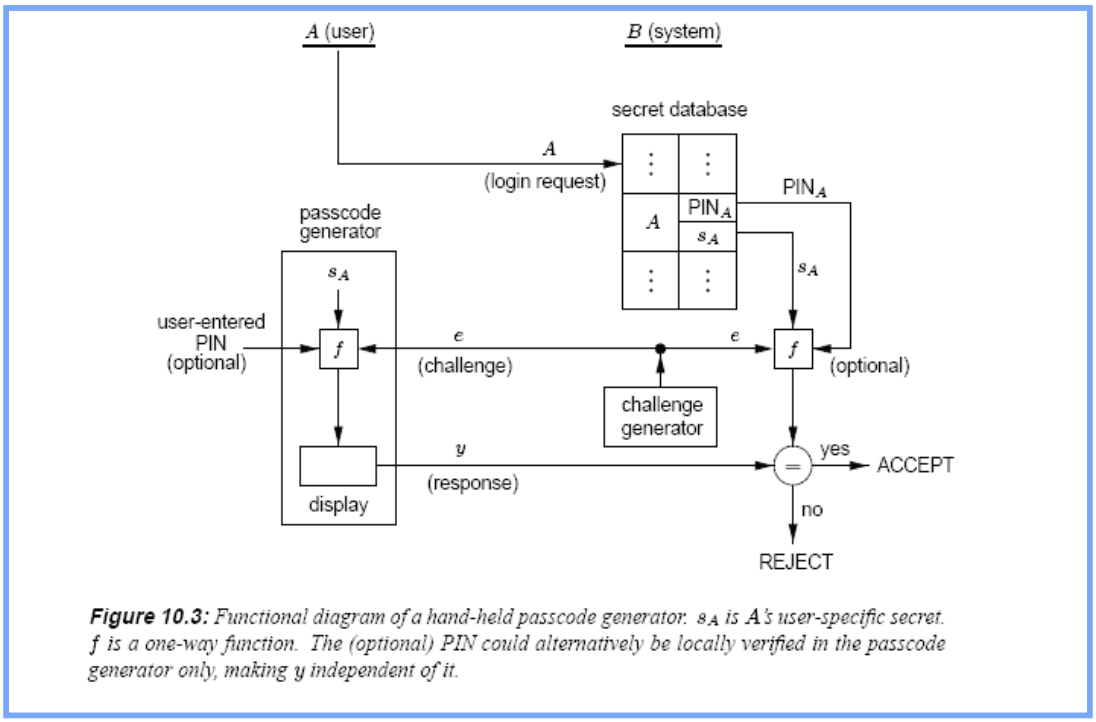


Figure 10.3: *Functional diagram of a hand-held passcode generator. s_A is A's user-specific secret. f is a one-way function. The (optional) PIN could alternatively be locally verified in the passcode generator only, making y independent of it.*

Quando un utente A vuole autenticarsi ad un utente B, invia il proprio identificativo sulla rete. B, ricevuto tale identificativo se ne serve per selezionare nel database le informazioni relative all' identificativo ricevuto. In base a tali informazioni e con un challenge generato da un challenge generator, calcola attraverso la funzione f, il valore che confronterà poi con quello inviatogli da A.

A, calcola tale valore utilizzando un passcode generator, mandando in input alla stessa funzione f , in esso memorizzata, le stesse informazioni utilizzate da B.

Solo se i due valori coincidono l' utente viene accettato altrimenti viene rigettato.

2.2.3 C-R con tecniche a chiave pubblica

Con il CR a chiave pubblica il client dimostra di conoscere il segreto attraverso la decrittazione di una sfida precedentemente crittata con la propria chiave pubblica.

Esistono due tipologie che andremo ad analizzare:

- Decrittaggio a chiave pubblica con witness (testimone)
- Needham-Schroedr PK modificato

Idealmente la coppia di chiavi pubbliche usate in tali meccanismi non dovrebbe essere riutilizzata per altri scopi dal momento che il suo riuso poterebbe compromettere la sicurezza delle informazioni.
Un'ulteriore cosa a cui fare attenzione è che il sistema a chiave pubblica utilizzato non sia suscettibile ad attacchi sul testo cifrato, come ad esempio l'estrazione di informazioni da parte di un avversario.
Una soluzione per risolvere entrambi questi problemi è quella di usare un generatore casuale di numeri detto anche **CONFOUNDER.**
Tali dati devono essere disponibili al verifier (verificatore) nel testo in chiaro per permettere la verifica alla conclusione del processo di autenticazione.

2.2.3.1 C-R basato su decrittazione a chiave pubblica

- Autenticazione basata su *Decrittaggio a chiave pubblica con witness (testimone)*

$$A \longleftarrow B : h(r), B, P_A(r, B)$$
$$A \longrightarrow B : r$$

Il confounder del verificatore **B** sceglie un numero casuale **r** e attraverso la funzione di hash **h** in cui è introdotto il valore casuale **r** calcola il testimone **x**.
Calcola la sfida **e = P_A (r, B),** attraverso l'algoritmo di crittaggio a chiave pubblica per **A**, in cui vengono introdotti i parametri **r** e **B** (identificatore).
L'algoritmo di crittaggio non è altro che una funzione in cui introdurre dei valori che permette di avere in output un valore desiderato, in tal caso 1 oppure 0.
Successivamente viene inviato il messaggio che comprende il testimone, l'identificatore e la sfida.

A riceve il messaggio e lo decritta grazie alla conoscenza della chiave privata corrispondente alla chiave pubblica usata dall'algoritmo **P_A** ed estrae due informazioni: **r'** e **B'** che non sa ancora se sono uguali a **r** e **B**. Procede al calcolo di **x'** e, in caso di differenza tra i due testimoni chiude la comunicazione, altrimenti invia **r'** a **B** che verificando l'uguaglianza tra **r** ed **r'** autentica **A**.

L'uso del testimone preclude la possibilità che si verifichino attacchi al messaggio.

Questo perché un utente esterno anche intercettando il testimone non può risalire al valore di **r** generato dal verificatore e quindi non può generare alcun **r' = r**

- Autenticazione basata su *Protocollo Needham-Schroeder a chiave pubblica modificato*

$$A \longrightarrow B: P_B(r1, A)$$
$$A \longleftarrow B: P_A(r1, r2)$$
$$A \longrightarrow B: r2$$

Questo protocollo fornisce la possibilità di trasportare due chiavi distinte $\mathbf{k_1}$ e $\mathbf{k_2}$, rispettivamente, da **A** a **B** e da **B** ad **A** per permettere la mutua autenticazione.
Se la caratteristica di definizione della chiave non è richiesta, $\mathbf{k_1}$ e $\mathbf{k_2}$ possono essere omesse.

- Con $\mathbf{P_B}$ denotiamo l'algoritmo di crittaggio a chiave pubblica per **B** con cui **A** genera la sfida per **B**
- Con $\mathbf{P_A}$ denotiamo l'algoritmo di crittaggio a chiave pubblica per **A** con cui **B** genera la sfida per **A**

2.2.3.2 Challenge Response basato su firma digitale

Vediamo tre meccanismi di identificazione di tipo challenge-response basati su firma digitale. Per quanto riguarda la notazione $\mathbf{r_A}$ e $\mathbf{t_A}$ sono rispettivamente un numero random e il timestamp generato da **A**. Il timestamp è un vero e proprio cartellino, che si allega al messaggio, dove è indicato l'orario di invio. Il dispositivo ricevente può così capire se quel messaggio ha un ritardo sopportabile oppure se deve essere cestinato in quanto portatore di un ritardo eccessivo.
$\mathbf{S_A}$ è la firma del client **A**.
$\mathbf{Cert_A}$ è il certificato contenente la firma di **A** con chiave pubblica, permette al verificatore di conoscere la chiave pubblica di **A**. Infatti **B** legge il certificato e vede quale chiave corrisponde ad una determinata firma. In tale meccanismo se il verifier già possiede la chiave pubblica del client l'utilizzo dei certificati può essere omesso.

- *Autenticazione unilaterale con timestamps*

$$A \longrightarrow B: Cert_A, t_A, B, S_A(t_A, B)$$

B, a ricezione avvenuta, verifica che il timestamp sia valido, che l'identificatore B sia proprio il suo, e (usando la chiave pubblica di A estratta dal certificato) controlla che la firma su questi due campi sia corretta.

- *Autenticazione unilaterale con numeri random*

I timestamp possono essere sostituiti da numeri random a costo però di inserire un messaggio addizionale.

$$A \longleftarrow B : r_B$$

$$A \longrightarrow B : Cert_A, r_A, B, S_A(r_A, r_B, B)$$

B genera un numero casuale e lo invia ad **A**.
Quest ultimo genera un altro numero casuale e firma sia quello ricevuto $\mathbf{r_B}$ che quello generato $\mathbf{r_A}$, firmando anche l'identificatore di **B** e spedisce il pacchetto completo.
B verifica che l'identificatore sia proprio il suo e (usando la chiave pubblica di **A** estratta dal certificato) controlla che la firma sui due numeri random e sull'identificatore sia quella di **A**.

- *Muta Autenticazione unilaterale con numeri random*

$$A \longleftarrow B : r_B$$

$$A \longrightarrow B : Cert_A, r_A, B, S_A(r_A, r_B, B)$$

$$A \longleftarrow B : Cert_B, r_B, A, S_B(r_B, r_A, A)$$

Il procedimento è analogo al precedente con la differenza che non è solo **A** ad autenticarsi a **B**, mediante l'utilizzo di numeri random, ma anche **B**.

2.3 Zero-Knowledge

Purtroppo, di solito, l'unico modo per dimostrare di conoscere una certa cosa, consiste nel dirla; a quel punto però, anche altri la vengono a sapere.Usando la funzione unidirezionale, invece, un utente può fornire una zero-knowledge proof cioè una prova a conoscenza zero. Questo tipo di protocollo si basa sull'autenticazione forte del Challenge-Response sebbene sia un passo avanti rispetto a quel tipo di protocollo. L'utente A

dimostra di avere una certa informazione, ma non concede a B la possibilità di conoscere tale informazione perché non viene rivelato nulla né sul segreto stesso né sulla relazione che c è tra sfida e risposta.

Questo tipo di protocollo si basa sulla prova interattiva. B (sistema di autenticazione) pone ad A (utente) un certo numero di domande: se A conosce il segreto è in grado di rispondere correttamente a tutte le domande; altrimenti può solo indovinare. L'obbiettivo è proprio quello di convincere B di un'asserzione, detta prova (o dimostrazione) di conoscenza (proof of knowledge) senza che vengano trasferite delle informazioni riguardo tale asserzione. Naturalmente il sistema B può accettare o meno tale dimostrazione e concedere o no l'accesso.

Un buon protocollo (completo e valido) deve fare in modo che, dopo un certo numero di passi, B sia certo circa l'identità di A con una probabilità il più possibile vicina ad 1. Proprio su questo progressivo convincimento si basa il concetto di AUTENTICAZIONE PROBABILISTICA che è fondamentale per lo Zero-Knowledge.
Ad ogni passo successivo B è sempre più convinto dell'identità di A.

Tuttavia la tradizionale nozione matematica di dimostrazione è alterata a dispetto di un "gioco interattivo" dove le dimostrazioni sono probabilistiche invece che assolute; una riprova, in questo contesto, deve essere corretta solo con una limitata probabilità, sebbene, come detto, possibilmente vicina ad 1.
Una dimostrazione interattiva è chiamata anche una dimostrazione di protocollo.

2.3.1 Proprietà del protocollo

- ***Completezza:*** un protocollo interattivo è completo se, dati un utente onesto ed un sistema onesto, il protocollo ha successo con schiacciante probabilità.
- ***Validità:*** un protocollo interattivo è valido se un utente A, che riesce ad autenticarsi, è in possesso dei dati (pubblici e privati) che gli consentono di eseguire il protocollo; allora potrà usare quei dati per autenticarsi in utilizzi successivi del protocollo.

Se queste due condizioni sono rispettate un utente esterno che vuole impersonare l'utente A deve assolutamente conoscerne il segreto s.

- ***Zero-Knowledge proprietà:*** un protocollo ha le proprietà di zero-knowledge se è descrivibile in questo modo: deve esistere un simulatore che è in grado di produrre, sulla base dell'input dell'affermazione da provare ma senza interagire con il vero prover, copie indistinguibili da quelle provenienti da un interazione col vero prover.
  Cioè deve esistere la possibilità di simulare, con assoluta fedeltà all'originale, un certo processo di autenticazione.

- ***Computazionalità:*** se considero un osservatore C che assiste ad una ZK dimostrazione che coinvolge A (prover) e B (verificatore), la dimostrazione in questione non fornisce alcun informazione utile a C. Allo stesso tempo una registrazione della comunicazione non fornisce alcuna garanzia a C sulla riproducibilità della stessa.
  Il protocollo raggiunge un livello di computazionalità se l'osservatore C non riesce a distinguere le trasmissioni reali da quelle simulate.
  Si dice computazionalmente perfetto se le probabilità tra transazioni reali e simulate sono identiche.

2.3.2 Vantaggi e svantaggi del protocollo

- Il protocollo mantiene sempre lo stesso livello di sicurezza poiché non vengono inviate informazioni. Quindi anche dopo un certo numero n di esecuzioni è come se si stesse eseguendo il protocollo per la prima volta.
- Molte tecniche ZK evitano l'uso di algoritmi di crittazione espliciti riuscendo ad essere più leggeri dal punto di vista esecutivo.

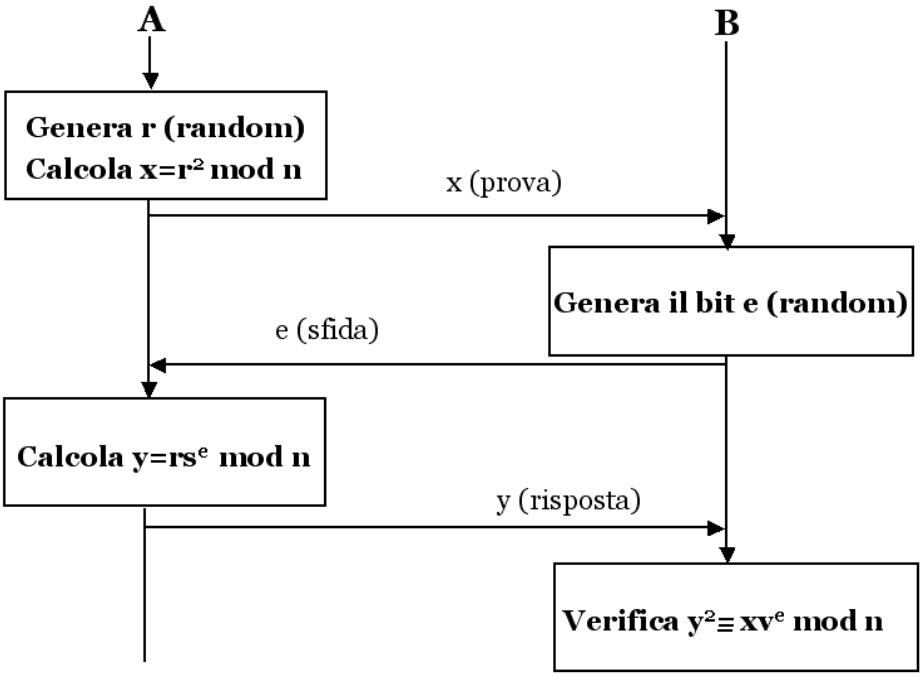
- **Costi:** le tecniche basate sullo ZK sono estremamente efficienti, ma hanno dei costi di esecuzione e memoria usata elevati.
- Alcuni protocolli ZK (dimostrazione di conoscenza) sono basati su assunzioni matematiche non completamente dimostrate(come ad esempio la fattorizzazione di un numero), ma solo accettate grazie alla conoscenza preliminare di altri teoremi dimostrati.

2.3.3 Protocollo di Fiat-Shamir

Si basa sull'idea di ZK proof dove **B** pone una singola sfida per ogni iterazione. La sua sicurezza è ottenuta dalla difficoltà di trovare la radice quadrata di un numero modulo **n**.

FASE DI SET-UP

- Una terza parte **T** genera casualmente i numeri primi **p** e **q** e calcola il modulo **n = p q**
- **T** rende pubblico **n** e mantiene segreti **p** e **q**
    - **A** (prover) genera casualmente la chiave privata **s (0<s<n)** tale che sia coprimo con **n**.
    - **A** calcola **v** = $\mathbf{s^2}$ **mod n** che **T** registra come chiave pubblica di **A**.



FUNZIONAMENTO

Il confounder di **A** sceglie un numero casuale **r** (**0< r < n**) e tramite questo numero il client calcola la prova (testimone) **x = r_ mod n**.

Viene spedita al verificatore **B** tale prova e successivamente **B** genera il bit di sfida **e=0** oppure **e=1** che viene inviato al prover **A**.

A calcola la risposta a partire dalla formula **y = r •s^e mod n** e la invia a **B**.

Il processo sta per chiudersi: **B** verifica la congruenza **y_= x•v^e mod n** e, in caso di verifica positiva, autentica il client **A**.

2.3.3.1 Attacchi

Esistono due possibili attacchi a questo sistema di autenticazione:

- Se **C** sceglie **r**, calcola $\mathbf{r^2}$ riuscirà a superare la sfida solo se **e = 0** poiché, in tal caso, la forma della risposta deve essere del tipo **y = r** che è facilmente intuibile poiché è stato **C** stesso a generare **r**.Se invece **e = 1**, **C** non sarà in grado di superare la prova poiché non può calcolare una risposta del tipo **y = rs** in quanto il valore di **s** è computazionalmente incalcolabile a partire da $\mathbf{s^2}$ **mod n**

- Se **C** sceglie **x** in modo tale che valga la relazione $\mathbf{r^2 = x^2/s^2}$ riuscirà a superare la sfida solo se **e=1** in quanto può calcolare una risposta del tipo **y = rs** facendo la radice quadrata **x**. Se invece riceve **e=0** non sarà in grado di superare la prova poiché per trovare **y = r** deve calcolare la radice quadrata di **s mod n**.

Perciò quando un utente tenta di autenticarsi senza conoscere **s** potrà rispondere ad una sola delle due domande.Ad ogni esecuzione dell'algoritmo la sua possibilità di autenticarsi si dimezza: dopo **t** passi avrà $\mathbf{(1/2)^t}$ possibilità di autenticarsi.
Perciò al crescere di **t** cresce anche la sicurezza del sistema.

Come si vede durante ogni esecuzione di un protocollo ZK il verificatore **B** non apprende nulla su **s**.
Se **e = 0**, **y** non contiene **s**
Se **e = 1**, nella risposta **y** il segreto **s** è mascherato dal fattore **r** che né **B** né altri possono ricavare.

2.3.4 Altri protocolli ZK

Esistono comunque, anche se meno conosciuti, anche altri tipi di protocollo a conoscenza zero, in particolare ricordiamo:

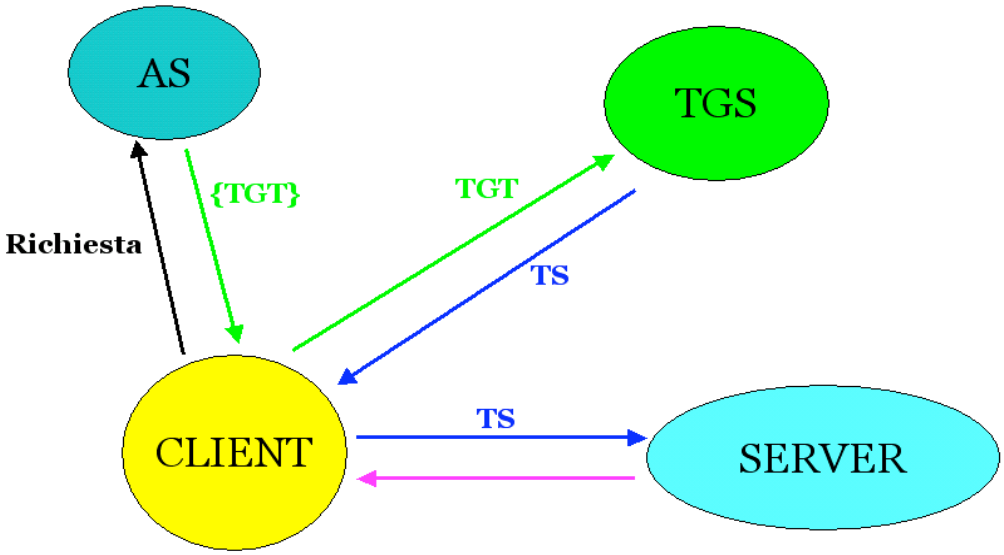
- **Feige-Fiat-Shamir:** si basa sull'idea di ZK proof dove **B** pone k sfide per ogni iterazione. La probabilità che un utente esterno possa autenticarsi è data da $\mathbf{2^{-kt}}$
- **Guillou – Quisquater**: è un estensione di Fiat-Shamir con riduzione del numero di messaggi scambiati e della memoria occupata.
- **Schnorr:** è basato sull'intrattabilità del logaritmo discreto.

2.4 Kerberos

Kerberos è un protocollo di autenticazione dei servizi di rete creato dal MIT che si serve della crittografia a chiave segreta evitando così la necessità di inviare password attraverso la rete. Autenticare mediante Kerberos impedisce agli utenti non autorizzati di intercettare le password inviate attraverso la rete.
Lo scopo principale di Kerberos è quello di eliminare la trasmissione delle informazioni di autenticazione attraverso la rete. Il corretto utilizzo di Kerberos vi permette di ridurre drasticamente la possibilità di intercettazione da parte dei packet sniffer.

2.4.1 Funzionamento

Kerberos differisce dagli altri metodi di autenticazione. Al posto di attuare un'autenticazione tra ogni dispositivo del client e ogni server, Kerberos utilizza la cifratura simmetrica e un sistema fidato di terze parti — noto come Key Distribution Center o KDC — per autenticare gli utenti su una rete e consentire loro di accedere ai servizi desiderati. Sia gli utenti che i servizi (detti ambedue principal) possiedono una chiave che deve essere conosciuta dal KDC.
Il termine principal è molto importante a livello concettuale.Infatti permette di capire come sia i client che i server vengono considerati allo stesso livello in questo tipo di protocollo. Ambedue devono sottostare alle stesse regole e hanno gli stessi diritti.



Il blocco AS (Autentication Server) e il blocco TGS (Ticket Granting Server) fanno parte di un unico KDC, ma sono stati scissi per comodità.

Il client fa un'autenticazione iniziale (detta di bootstrap) con l'AS del KDC chiedendogli un TGT (Ticket Granting Ticket).
Il KDC registra nel proprio database la chiave pubblica del client e i suoi dati utili, concede questo TGT crittografato con la chiave pubblica del client stesso che era stata registrata precedentemente (durante ilo bootstrap) nel database e codificata con la chiave del KDC.
In questo modo, ogni volta che il client deve utilizzare un servizio, presenta il proprio TGT al TGS del KDC specificando il servizio che vuole utilizzare.
Il KDC (o meglio il suo TGS) invia al client un ST (Service Ticket), crittografato con la chiave del servizio scelto.
Tale ST fornisce all'utente un ulteriore garanzia della sua autenticità; poiché il server vedendo che il ticket proviene da un'entità kerberizatta fornisce l'accesso ai propri servizi, fidandosi ciecamente. Ad ogni utilizzo successivo di quel servizio basta che il client presenti il Service Ticket e otterrà il nuovo accesso.

Il TGT ha una durata limitata in modo tale che un TGT compromesso possa essere utilizzato solo per un periodo limitato.Dopo la scadenza, un utente deve inserire nuovamente la propria password per ottenere un nuovo TGT e quindi ripetere tutto il processo a partire dall'autenticazione di bootstrap.
Generalmente la durata di un ticket è di 8 ore oppure meno; infatti se si esce dalla sessione durante queste 8 ore, si deve rieffettuare l'autenticazione inziale di bootstrap

Quanto detto vale per i servizi che compongono la rete realm. Questa rete è composta da un insieme di server e client che usano Kerberos come sistema di autenticazione.

2.4.2 Vataggi di Kerberos

- elimina la trasmissione di informazioni di autenticazione attraverso la rete
- login unico per i server appartenenti alla realm
- meccanismo di Ticket utile per connessioni intermittenti (ISDN, portatile)
- crescente supporto commerciale (Windows 2000)

2.4.3 Svantaggi di Kerberos

- richiede la sincronizzazione dei clock dei dispositivi sulla rete. Infatti in caso di asincronismo un ticket ancora valido per un client A può risultare già scaduto per un server B.

- l’accesso remoto al KDC per l’autenticazione di bootstrap richiede l’invio di una password in chiaro (non crittata)
- per avere una rete sicura si dovrebbero “kerberizzare” tutte le applicazioni che inviano password in testo
- configurare un’applicazione di rete affinché possa usare Kerberos richiede molta programmazione

2.4.4 Versione 4 contro versione 5

Quanto detto fino ad ora vale per la versione di Kerberos 4. Alla metà degli anni 90 ne è stata sviluppata anche una versione 5 che presenta differenze e miglioramenti sia dal punto di vista tecnico che dal punto di vista organizzativo.

Le differenze tra le due versioni sono le seguenti:

- la 4 è ristretta ad un unico ambiente, permette autenticazioni solo tra elementi della stessa realm
- la 4 usa il DES mentre la 5 non solo
- nella versione 5 i Ticket sono più lunghi a livello di dati ed estesi a livello temporale
- nella versione 5 i byte di messaggio hanno diverso ordine
- la 5 permette autenticazioni tra elementi di diversi ambienti
- la 5 si basa sul protocollo Internet
- la 5 è specificata in dettaglio nell’ RFC1510 e usata da molti servizi